SIAM J. COMPUT.
Vol. 22, No. 1, pp. 1-10, February 1993

IMPLICIT $O(1)$ PROBE SEARCH*

AMOS FIAT† AND MONI NAOR‡

Abstract. Given a set of n elements from the domain $\{1, \cdots, m\}$, this paper investigates how to arrange them in a table of size n, so that searching for an element in the table can be done in constant time. Yao [*J. Assoc. Comput. Mach.*, 28(1981), pp. 615-628] has shown that this cannot be done when the domain is sufficiently large as a function of n.

This paper gives a constructive solution when the domain m is polynomial in n, the number of elements, as well as a nonconstructive proof for m no larger than exponential in poly (n). The authors improve upon a result of Yao and give better bounds on the maximum m for which implicit $O(1)$ probe search can be done. The results are achieved by showing the tight relationship between hashing and certain encoding problems called rainbows.

Key words. hashing, perfect hashing, spatial complexity, Ramsey theory, randomness in computation

AMS(MOS) subject classifications. 68P05, 68P10, 68Q05, 68R05, 68R10

1. Introduction. The problem addressed in this paper is that of searching a full table: A set $S \subset \{1, \cdots, m\}$ of size n is to be stored in a table T of size n, where every table entry holds a single element of S. Given $x \in \{1, \cdots, m\}$, the goal is to locate x in the table or to indicate that $x \notin S$, while probing the table as few times as possible. We assume that n and m are known to the searcher.

Yao [8] has shown that if no storage is available in addition to the table T, then there is no table organization that enables an element to be located in less than $\log n$ probes. We refer to a table organization that requires no additional storage as an *implicit* scheme. Yao's proof assumes that the domain size m is much larger than the number of elements n. This immediately raises the following two questions.

(1) For what values of m (relative to n) does an implicit $O(1)$ probe search scheme exist?

(2) Given that an implicit scheme does not exist, how much additional storage is required to ensure $O(1)$ search?

In [4] Fiat, Naor, Schmidt, and Siegel show that if $m = O(n)$, then search can be performed in $O(1)$ time without any additional storage. As for the second question, Fredman, Komlós, and Szemerédi [5] show that one probe search can be performed with $O(n\sqrt{\log n} + \log\log m)$ additional bits of storage. In [4] an $O(1)$ probe scheme is given that requires only $O(\log n + \log\log m)$ additional bits of storage.

We give an implicit $O(1)$ probe search scheme for a domain of size m that is polynomial in the number of elements n. We prove that an implicit scheme exists whenever m is bounded by $2^{\text{poly}(n)}$, based upon a probabilistic construction. It then follows from [4] that $O(\log\log m)$ additional bits are sufficient for any m.

We provide a refinement to Yao's theorem mentioned above that yields a better bound on the maximum m for which implicit $O(1)$ probe search schemes exist. Our proof technique gives a lower bound tradeoff between the number of probes and the

* Received by the editors March 14, 1991; accepted for publication (in revised form) September 9, 1991. Most of this work was performed while both authors were at the University of California, Berkeley, California 94720.

† Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. The work of this author was supported by a Weizmann Postdoctoral Fellowship and by National Science Foundation grant DCR 84-11954.

‡ IBM Research Division, Almaden Research Center, 650 Harry Rd, San Jose, California 95120. The work of this author was supported by National Science Foundation grant DCR 85-13926 and CCR 88-13632.

size of the domain that allows implicit search. In particular, if $O(1)$ probe implicit search is possible, then we can show that

$$m \leqq 2^{2^{2^{\cdot^{\cdot^{2^n}}}}}\Big\}O(1).$$

Yao's proof technique obtains a tower whose height depends on n.

These results are obtained by mean of a class of structures we call *rainbows*. A t-sequence over a set U is a sequence of length t, without repetitions, of elements in U. A (c, m, n, t)-rainbow is a coloring of all t-sequences over $\{1, \cdots, m\}$ with c colors so that for any set $S \subset \{1, \cdots, m\}$, $|S| = n$; all c colors occur in the coloring of the t-sequences over S. We show that the existence of $(c = n, m, n, t = O(1))$-rainbows is essentially equivalent to the implicit $O(1)$ probe search problem for a set of n elements chosen from the domain $\{1, \cdots, m\}$.

The relationship between rainbows and implicit $O(q)$ probe search schemes is specified by the following theorems.

THEOREM 1. *For m, n, let $c = \max(n, \log m)$. The existence of a $(c, m, n, t = O(1))$-rainbow yields an implicit $O(1)$ probe search scheme for n elements from the domain* $\{1, \cdots, m\}$.

THEOREM 2. *Given an implicit $O(1)$ probe search scheme for n elements chosen from the domain $\{1, \cdots, m\}$, we can construct an $(n, m, n, t = O(1))$-rainbow.*

To motivate rainbows, we start in § 2 by showing how they can be utilized to provide virtual memory.

Theorems 1 and 2 are proved in §§ 3 and 4. Section 5 contains a probabilistic construction for a $(c = n, m = 2^{\text{poly}(n)}, n, t = O(1))$-rainbow and an explicit construction for a $(c = n, m = \text{poly}(n), n, t = O(1))$-rainbow. Thus, by Theorem 1, we achieve the bounds claimed in the beginning of this section.

From their definition it is apparent that rainbows are related to Ramsey Theory. Indeed, the impossibility results we have are derived from Ramsey Theory are expressed in terms of Ramsey numbers.

In § 6 we show bounds on the maximal m, as a function of n, for which an $(n, m, n, t = O(1))$-rainbow exists. Thus, Theorem 2 gives bounds on m for which implicit $O(1)$ probe search schemes exist. Section 6 also discusses the connection between rainbows and colorings of the uniform hypergraph.

Section 7 deals with the relationship between the rainbow structure and other structures, called dispersers, proposed in the literature (for completely different applications). Specifically, we show that if an explicit construction for a certain kind of dispersers is possible, then we can find an explicit construction for an implicit $O(1)$ probe search scheme for m that is $n^{\log n}$. For these dispersers, Sipser [7] gave a probabilistic construction.

2. Rainbows provide virtual memory. We now show how Rainbows can be used to simulate additional memory. The virtual memory problem with parameters c, n', t, l is defined below:

Virtual memory problem.
- A set $R = \{r_1, r_2, \cdots, r_{n'}\}$, where $1 \leqq r_j \leqq m$ for $1 \leqq j \leqq n'$.
- A series of values $v_1, v_2, \cdots, v_l$, where $0 \leqq v_i \leqq c-1$ for $1 \leqq i \leqq l$.

Arrange the elements of R in an array A of size n' (putting each element of R in a different location), so that given $1 \leqq j \leqq l$, v_j can be reconstructed (decoded) quickly, via t accesses to A.

Note that we do not require anything about locating elements of R, only that they will reside somewhere in A.

The next lemma shows the relationship between this problem and the existence of rainbows.

LEMMA 1. *Given a (c, m, n, t)-rainbow $\mathscr{C}$, the virtual memory problem for parameters c, n', t, l such that $n - tl \geqq n$ can be solved.*

Proof. Divide the first $t \cdot l$ locations of the array A into blocks of size t. The elements of R should be arranged in A so that the color assigned by $\mathscr{C}$ to the jth block, i.e., to the sequence $\langle A[jt+1], \cdots, A[(j+1)t]\rangle$, is v_j. To achieve that, a greedy algorithm can be applied.

Greedy encoding.

```
• Set $U = R$.
• For $j = 1$ to $l$
    - find a sequence $s$ colored $v_j$ in $U$,
    - put the sequence $s$ in the $j$th block of $A$,
    - $U \leftarrow U \setminus s$.
• Arrange $U$ in the rest of $A$ arbitrarily.
```

Throughout the execution of the loop, the number of elements in U is $n' - jt \geqq n$. Hence there is a sequence in U colored by $\mathscr{C}$, and the find step in the algorithm always succeeds.

This arrangement means that in order to reconstruct v_j, we have to determine the color of the jth block under $\mathscr{C}$, and this can be done via t probes to A. This method is constructive if the color of a sequence under $\mathscr{C}$ can be determined effectively. □

3. Rainbows yield implicit $O(1)$ probe search. Our goal in this section is to prove Theorem 1. This section is strongly dependent on [4], which is the source of our techniques. A reader not familiar with the paper should be able to understand the major steps explained hereafter.

We note the following theorem from [4].

THEOREM 3 [4]. *n elements from the domain $\{1, \cdots, m\}$ can be arranged in a table of size n so that $O(1)$ probe search is possible, provided $O(\log n + \log\log m)$ additional bits of storage are available.*

We will concentrate on proving a slightly weaker version of Theorem 1.

THEOREM 4. *For m, n, let $c = \max(n, \log m)$. The existence of a $(c, m, n, t = O(1))$-rainbow yields an implicit $O(1)$ probe search scheme for $4n$ elements from a domain of size m.*

Later, we will show that the rainbow construction is robust in that the constants ($4n$) are irrelevant, a $(c, m, n, t = O(1))$-rainbow can be translated into another $(c^{e_1}, m^{e_2}, n^{e_3}, t' = O(1))$-rainbow for arbitrary fixed exponents e_1, e_2, $e_3 > 0$.

It now might seem trivial to prove Theorem 1, given Theorem 3. Given $4n$ elements from a domain of size m, we encode the $O(\log n + \log\log m)$ bits required by the (FNSS) scheme above by choosing $O(1)$ groups of t elements and ordering the elements in some fixed set of locations in the table as in the greedy encoding. During the search, the $O(1)$ special elements chosen for the encoding are read; if the search value is not found, then the elements are interpreted under the rainbow interpretation as representing the extra bits of storage required by the scheme in [4].

Unfortunately, moving the required elements to their position as required by the encoding ruins the original order suggested in [4]. To prove our claim, we must start afresh.

Given a set of n keys, $S \subset \{1, \cdots, m\}$. Fredman, Komlós, and Szemerédi [5] describe how to find a perfect hash function $f: \{1, \cdots, m\} \mapsto \{1, \cdots, n\}$ with the property that f is one-to-one and onto when limited to the domain S. This function requires a description of $O(\log\log m)+o(n\log n)$ bits. The description is split into $O(1)$ words of size $O(\log\log m+\log n)$ bits, plus an additional $o(n)$ words of $O(\log n)$ bits each. Evaluating the function f requires reading only $O(1)$ of these words.

We say that S is in the natural order in the table T relative to f if $T[f(x)]=x$, $x\in S$. The natural order is easy to search, given f's description. Another order that is easy to search is obtained by applying an arbitrary permutation $\tau:\{1,\cdots,n/2\}\mapsto\{1,\cdots,n/2\}$ to the first half of the table and applying τ^{-1} to the second half. (For $1\leqq i\leqq n/2$, set $T[\tau(i)]:=T[i]$; for $n/2<i\leqq n$, set $T[\tau^{-1}(i-n/2)+n/2]:=T[i+n/2]$.) The idea is that both τ and τ^{-1} are easy to compute. To compute $\tau^{-1}(i)$, simply evaluate $f(T[i])$; to compute $\tau(i)$, evaluate $f(T[i+n/2])-n/2$. As both τ and τ^{-1} can be computed with one probe to the table, search can be done by performing two probes to the table.

This is a variation of Feldman's involution trick, as presented in [2].

The method.

- Find a perfect hash FKS-function f for the $4n$ elements, as described in [5].
- Divide the elements into two sets, depending on whether $f(x)\leqq 2n$ or $f(x)>2n$.
- Encode the description of f by arranging the elements x with $f(x)\leqq 2n$ in the first half of T using the greedy encoding of § 2. By Lemma 1, this is possible.
- The arrangement defines a permutation τ of the elements in the first half of T, so organize the elements x with $f(x)>2n$ in the second half of the table as required under τ^{-1}.

Searching for an element now requires decoding $O(1)$ words of the description of the FKS-function. Each decoding requires probing the table at $O(1)$ locations. The natural order can be reestablished by appropriately computing either τ or τ^{-1}, each of which requires one probe plus $O(1)$ probes to read the [5] function description. Overall, search requires $O(1)$ probes.

4. Implicit $O(1)$ probe search yields rainbows. In this section we show that rainbows and $O(1)$ probe search schemes relate in the other direction as well; i.e., given a search scheme we show how to construct a rainbow. More specifically, we prove a refined version of Theorem 2.

THEOREM 5. *Given an implicit t-probe search scheme for n elements from the domain* $\{1,\cdots,m\}$, *an* $(n, m, n, t+2\log t)$*-rainbow can be constructed.*

Proof. The sequences are assigned colors based on simulating a search scheme. The colors $1,\cdots,n$ correspond to locations $1,\cdots,n$ in some imaginary search array. The idea is that in a t-sequence there is enough information to simulate a t probe search; i.e., given a t-sequence over $\{1,\cdots,m\}$, $e_1, e_2,\cdots,e_t$, we simulate a search for e_1 in the imaginary array, where e_{i+1}, $1\leqq i\leqq t-1$ is the element probed at Step i. Since the location probed at Step i is determined by the search value and the elements probed in Steps 1-$i-1$, we know the location in the imaginary array at each step of the simulation. The color assigned to the sequence is the last location we are to probe.

The only problem with this description is that e_1 might be probed at any of the t steps, not necessarily the last, but our sequences do not have repetitions. We can use $\log t$ bits to indicate the step number, j, at which e_1 is probed. This can be done by dedicating a pair of elements is allocated for each bit of j. If the elements are in order, they encode 0; otherwise 1. We assume that these elements are at the end of the sequence, that is, elements $e_{t+1}, e_{t+2},\cdots,e_{t+2\log t}$.

To summarize, the color assigned to the sequence

$$e_1, e_2,\cdots,e_t, e_{t+1},\cdots,e_{t+2\log t}$$

is the location of e_1 in the array for which the search is being simulated, where e_1 is encountered in the step encoded by $e_{t+1}, \cdots, e_{t+2\log t}$. Sequences that cannot be interpreted in such a fashion are colored arbitrarily.

CLAIM 1. *Given a set $S \subset \{1, \cdots, m\}$ of size n, and any color $1 \leqq c \leqq n$, there is a $t+2\log t$-sequence over S that is colored c.*

Proof. Assume that the set S is arranged in the array A so that implicit t-probe search is possible. Consider the sequence consisting of the elements probed in A when searching for $A[c]$, concatenated to $2\log t$ elements in S not appearing in the probe sequence whose order encodes the step number at which cell c is probed. This sequence is colored c, and consists only of elements in S. □

5. Rainbow construction. This section provides an explicit construction of rainbows when the number of colors $c = n$ and the length of the sequence t is a constant. We start with a construction for a domain m that is quadratic in the number of elements n (Lemma 2). The ideas behind this construction are later used in showing how to reduce a problem with domain m to another problem with domain $\sqrt{m}$ (Lemma 3). This yields an explicit recursive construction for any m that is polynomial in n (Theorem 6). Theorem 6 yields as a corollary that implicit $O(1)$ probe search scheme is possible when m is polynomial in n. We conclude the section by showing that a probabilistic construction is good even when m is exponential in n (Theorem 7).

LEMMA 2. *For any prime p, there is an explicit construction of a $(c = n,\ m = p^2,\ n = p+1,\ t = 2)$-rainbow.*

Proof. Consider a one-to-one mapping from all elements $e \in \{1, \cdots, m\}$ to pairs (x, y) such that $0 \leqq x,\ y \leqq p-1$. (For instance, $x = e \pmod p$, $y = (e-x)/p \pmod p$.) Given an element in $\{1, \cdots, m\}$, we will set its value of the mapping.

Color the sequence $\langle u, v\rangle$, $u = (x_1, y_1)$, $v = (x_2, y_2)$, with the color $(y_2 - y_1)/(x_2 - x_1) \pmod p$. If $x_2 = x_1$, then color the sequence $\langle u, v\rangle$ with the color p. We have colored all edges of the full directed graph on m vertices. Note that the sequence $\langle u, v\rangle$ is colored as the sequence $\langle v, u\rangle$; hence we can consider the coloring as that of a complete undirected graph. To prove that this is a good coloring, we need the following.

CLAIM 2. *Consider the edge induced subgraph G_i obtained by choosing all edges of color i. G_i consists of p vertex disjoint cliques of size p.*

Proof. First, note that every vertex $u = (x, y)$ has exactly $p-1$ directed edges $(u, v_j = (x_j, y_j))$ colored i, for all $0 \leqq i \leqq p$. For $i = p$, these are simply pairs (x, y_j), $y_j \neq y$; for $i < p$, the x_j and values are the $p-1$ solutions to the equation $(y_j - y)/(x_j - x) = i \pmod p$.

To show that the undirected induced subgraph consists of cliques, assume that the $\langle u, v\rangle$ and $\langle v, w\rangle$ sequences are colored i: then the $\langle u, w\rangle$ sequence must also be colored i. If $u = (x_1, y_1)$, $v = (x_2, y_2)$, and $w = (x_3, y_3)$, either $i = p$ in which case $y_1 = y_2 = y_3$ and (u, w) is also colored p or $i < p$, in which case $(y_2 - y_1)/(x_2 - x_1) = (y_3 - y_2)/(x_3 - x_2) = i \pmod p$. It now follows that $(y_3 - y_1)/(x_3 - x_1) = i \pmod p$. □

Remark. Note that all vertices u_j, $u_j = (x_j, y_j)$, belonging to the same clique in G_i, have the same value $y_j - ix_j \pmod p$. This means that we can identify the clique in G_i containing a vertex u.

We can now resume the proof of the lemma. Given a set $S \subset \{1, \cdots, m\}$ of size $n = p+1$, for all $0 \leqslant i \leqslant p$, at least two elements $u,\ v \in S$ belong to the same clique in G_i. This means that both sequences $\langle u, v\rangle$ and $\langle v, u\rangle$ are colored i.

To construct a rainbow for m polynomial in n, we use a recursive construction. We explain how to use the construction above to transform the problem from a domain of size m to a domain of size $\sqrt{m}$ by concatenating two elements to each sequence in the $\sqrt{m}$ domain.

LEMMA 3. *Given a construction of a $(c=n,\ m=p,\ n-2,\ t)$-rainbow, p a prime, a $(c=n,\ p^2,\ n,\ t+2)$-rainbow can be constructed.*

Proof. Let $\mathscr{C}_1$ be a $(p+1,\ p^2,\ p+1,\ 2)$-rainbow as described in Lemma 2, and let $\mathscr{C}_2$ be an $(n,\ p,\ n-2,\ t)$-rainbow that exists by assumption. Our goal is to construct an $(n, p^2, n, t+2)$-rainbow. Given a $t+2$-sequence $\bar{e}=e_1, e_2, \cdots, e_{t+2}$, over $\{1, \cdots, m\}$ we use e_1 and e_2 as indicators. If $e_1 > e_2$, then color $\bar{e}$ with the color assigned to (e_1, e_2) under $\mathscr{C}_1$.

Given a set $S \subset \{1, \cdots, m\}$, $|S| = n$, if all $p+1$ colors occur in the coloring of the 2-sequences over S under $\mathscr{C}_1$, then we are done. (In fact, the rainbow contains more colors than required.)

Otherwise, at least one color is missing under $\mathscr{C}_1$, but there is at least one color that appears (we assume $n \geqq 2$); therefore, there is a color i such that no pair in S is colored by $\mathscr{C}_1$ with i, but there exist $u, v \in S$ such that (u, v) is colored $i-1 \pmod{p+1}$ under $\mathscr{C}_1$.

Consider G_i, the edge induced graph defined by edges colored i and introduced above. Every element in S is in a different clique of G_i; otherwise there would have been a pair colored i. The cliques of G_i can easily be indexed as described by the remark at the end of Lemma 2.

If $e_1 < e_2$, we translate $\bar{e}$ to a t-sequence, $\bar{d}=d_1, d_2, \cdots, d_t$, over $1, \cdots, \sqrt{m}$. We color $\bar{e}$ by the color assigned to $\bar{d}$ by $\mathscr{C}_2$. Let d_j be the index of the clique of G_i containing e_{j+2}, $1 \leqq j \leqq t$, and $i-1 \pmod{p+1}$ is the color assigned to (e_1, e_2) under $\mathscr{C}_1$.

By the discussion above, it follows that for every $S \subset \{1, \cdots, m\}$, and for every $1 \leqq k \leqq c$ there is a $t+2$-sequence over S that is colored k. Thus we have described a construction for an $(n,\ p^2,\ n,\ t+2)$-rainbow. □

Since for any integer x there is a prime in $(x, 2x)$, we can apply Lemma 3 recursively, each time reducing the domain from m to $2\sqrt{m}$. Using Lemma 2 as the base case provides us for any $d \geqq 1$ with an explicit construction of an $(c=n,\ m=n^d,\ n,\ 2\lceil \log d \rceil + \lceil \log\log d \rceil)$-rainbow. Thus we have the following.

THEOREM 6. *For any domain m polynomial in the set size n there exists an $(n, m, n, O(1))$-rainbow. Given a sequence, its color can be determined in $O(1)$ time assuming modular arithmetic in unit time.*

Remark. Note that the proof implies that the existence of rainbows is a robust property, meaning that if p_1, p_2, p_3 are polynomials, and m is as a function of n such that a $(c, m, n, O(1))$-rainbow exists, then a $(p_1(c), p_2(m), p_3(n), O(1))$-rainbow exists as well.

It now follows from Theorem 1.

COROLLARY 1. *For any domain m polynomial in the set size n there exists an implicit $O(1)$ probe search scheme for which search requires $O(1)$ time, assuming modular arithmetic in unit time.*

Probabilistic constructions. We now turn to probabilistic constructions of rainbows for m that is *exponential* in n. Suppose $m = 2^{n^l}$, and consider a random coloring with n colors of all $l+2$ sequences over $\{1, \cdots, m\}$. For a set $S \subset \{1, \cdots, m\}$, $|S| = n$, the probability that a specific color is missing in the $l+2$ sequences over S is less than

$$(1-1/n)^{n(n-1)\cdots(n-l-1)}.$$

There are n colors and $\binom{m}{n}$ sets; hence the probability that there exists a set and a color such that the color is missing over the set is less than

$$\binom{m}{n} \cdot n \cdot (1-1/n)^{n(n-1)\cdots(n-l-1)} \leqq 2^{n^{l+1}} \cdot n \cdot e^{-n^{l+1}} \cdot e^{l^2 n^l} \ll 1;$$

therefore, we have the following.

THEOREM 7. *For any domain m exponential in the set size n there exists an $(n, m, n, O(1))$-rainbow.*

COROLLARY 2. *For any domain m exponential in the set size n there exists an implicit $O(1)$ probe search scheme.*

6. Bounds on rainbows. In this section we give bounds on the maximum m, as a function of n and t, for which a $(c = n, m, n, t)$-rainbow can exist. We will do that by showing the connection between rainbows and colorings of the t-uniform hypergraph. Consider a coloring of all t-subsets (subsets of size t) of $\{1, \cdots, m\}$ with c colors. Ramsey Theory tells us that there exists a function $R(n, t, c)$ such that if $m > R(n, t, c)$, then for any coloring of the t-subsets of $\{1, \cdots, m\}$ with c colors there exists a set $S \subset \{1, \cdots, m\}$ of size n such that all the t-subsets over S are colored with the same color. (See the book by Graham, Rothschild, and Spencer [6] for details on Ramsey Theory.)

THEOREM 8. *If there exists a (c, m, n, t)-rainbow and $c > t!$, then*

$$m \leqq R(n, t, t!+1).$$

Proof. Let $\mathscr{C}$ be a (c, m, n, t)-rainbow. Define a coloring of the t-subsets of $\{1, \cdots, m\}$ $\mathscr{D}$: for each subset $H \subset \{1, \cdots, m\}$ of size t consider all possible orderings of H. Each of the $t!$ possible orderings receives a color in the rainbow. Since there are more than $t!$ colors in the rainbow, we know that there is a color i, $1 \leqq i \leqq t!+1$, which none of the orderings receives. $\mathscr{D}$ colors H with the least such i. From Ramsey Theory it follows that if $m > R(n, t, t!+1)$, then there will be a set $S \subset \{1, \cdots, m\}$ of size n such that all of S subsets of size t are colored under $\mathscr{D}$ with the same color i. Hence under $\mathscr{C}$ none of the t-sequences over S are colored i, and thus $\mathscr{C}$ is not a rainbow. $\square$

How fast does $R(n, t, t!+1)$ grow? Let the tower functions $h_i(x)$ be defined as $h_1(x) = x$ and $h_{i+1}(x) = 2^{h_i(x)}$ for $i \geqq 1$. That is,

$$h_i(x) = \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^x}}}}}}\right\}i-1.$$

The Stepping Up Lemma in [6, p. 91] yields the following: $h_{j-1}(c_1 \cdot n^2) \leqq R(n, j, 2) \leqq h_j(c_2 \cdot n)$ for some fixed c_1 and c_2. By the method of the proof of Ramsey's Theorem, increasing the number of colors from 2 to $t!+1$ does not add more than $\log t!+1$ to the height, i.e., $R(n, t, t!+1) < h_{t+\lceil\log(t!+1)\rceil}(c_2 n)$. Hence we can conclude that for a $(c \geqq t!+1, m, n, t = O(1))$-rainbow to exist we must have

$$m \leqq \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}\right\}O(1).$$

Applying Theorem 2, on the connection between rainbow and t-probe search, we get that an implicit t-probe scheme can exist only if $m < R(n, t', t'!+1)$, where $t' = t + 2\log t$. Thus, for an implicit O(1) probe search to exist we must have

$$m \leqq \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}}\right\}O(1).$$

This constitutes a new proof of Yao's theorem [8] with better bounds. His bounds imply that $m < R(2n-1, n, n!)$, which grows much faster. Yao's proof has the advantage that it implies that whenever $m \geqq R(2n-1, n, n!)$, the lower bound on the search time is $\lceil \log n \rceil$. Our proof cannot give better bounds than $\Omega(\log n/\log\log n)$, since $t!+1$ must be less than n.

Any improvement on the lower bounds for rainbows would yield a better lower bound for implicit $O(1)$ probe search. Conversely, constructive implicit $O(1)$ probe search schemes for higher bounds imply better rainbow constructions. The reader can interpret this as either an optimistic or a pessimistic statement.

Undirected rainbows. We now show that the existence of rainbows is closely related to that of undirected rainbows defined as follows: *A* (c, m, n, t)*-undirected rainbow* is a coloring of all t-subsets over $\{1, \cdots, m\}$ with c colors, so that for any set $S \subset \{1, \cdots, m\}$, $|S| = n$, all c colors appear in the t-subsets over S.

Since the order itself in directed rainbows can determine $t!$ different colors, we know that $(c = t!, m, n, t)$-rainbows exist for any m and n such that $m \geqq n$. However, by Ramsey theory, this is not true for undirected rainbows. On the other hand, the next theorem shows that in order to give bounds on the maximum m for which $(c = n, m, n, O(1))$-rainbows exist, it is enough to consider undirected rainbows.

THEOREM 9. *For every t there exists a constant b_t, dependent only upon t, such that a construction for a $(c = n, m, n, t)$-rainbow yields a construction for a $(c = n, m, \lceil \log (t!) + 1 \rceil \cdot n, b_t)$-undirected rainbow.*

Proof. The idea is to provide enough information in the b_t-subset so as to simulate an ordered set. If, in addition to a t-subset, $\lceil \log (t!) \rceil$ bits are provided to determine the order in the t-subset, then the color of the subset will be the color of the corresponding t-sequence in the $(c = n, m, n, t)$-rainbow.

Let $\mathscr{C}$ be a $(c = n, m, n, t)$-rainbow. From Theorem 8 we know that $m < R(n, t, t!+1)$, and thus $m < h_{t'}(c_2 n)$ for some t', depending only on t. From the lower bound on $R(n, j, 2)$ of the Stepping Up Lemma there exists a $(2, n, m, t')$ undirected rainbow $\mathscr{A}$. Let $b_t = t + t' \cdot \lceil \log (t!) \rceil$. Define $\mathscr{C}'$, a coloring of b_t-subsets, as follows. Sort the b_t-subset and partition it into $\lceil \log (t!) \rceil + 1$ subsets of consecutive elements such that the subset of the largest elements is of size t, and all the rest are of size t'. Compute the coloring under $\mathscr{A}$ of each of the t'-subsets. Each of the $\lceil \log (t!) \rceil$ t'-subsets supplies one bit under its 2-coloring, and together those bits determine an ordering of the t-subset. The color $\mathscr{C}'$ assigns is the one $\mathscr{C}$ assigns the t-sequence, resulting from the the t-subset when it is ordered by the encoding given by the smaller subsets.

To see that $\mathscr{C}'$ is indeed a $(c = n, m, \lceil \log (t!) + 1 \lceil \cdot n, b_t)$-undirected rainbow, consider any $S \subset \{1, \cdots, m\}$ of size $\lceil \log (t!) + 1 \rceil \cdot n$. Partition S into $\lceil \log (t!) + 1 \rceil \cdot$ subsets $S_1, S_2, \cdots$, such that each S_i is of size n and all the elements of S_i are smaller than those of S_{i+1}. For any color $1 \leqq j \leqq c$, $\mathscr{C}$ colors at least one t-ordered subset of $S_{\lceil \log (t!) + 1 \rceil}$ with j. The order of this subset determines $\lceil \log (t!) \rceil$ bits $b_1, b_2, \cdots, b_{\lceil \log (t!) \rceil}$. In each subset S_i there is a t'-subset colored b_i under $\mathscr{A}$. The b_t subset of S that is the union of all these subsets is colored j under $\mathscr{C}'$. $\square$

7. Construction through dispersers. In this section we show how an explicit construction for dispersers, defined below, yields an explicit construction for rainbows with $m = n^{\log n}$. An (m, n, d, a, b)*-disperser* is a bipartite graph with m nodes on the left side, each with degree at most d, n nodes on the right side with the property that every subset of a nodes in the left side is connected to at least b of the nodes of the right side. These graphs have been used, for instance, by Ajtai, Komlós, and Szemerédi [1] and Sipser [7] to remove randomness in probabilistic algorithms. Cohen and Wigderson [3] provide a survey of constructions and applications.

Let $m = n^{\log n}$. We first show how to construct a rainbow with $\log n$ colors for such m and n, and then we show how to apply it with a $(m, n, \log^2 n, n, n/2)$-disperser to get an $(c = n, m, n, O(1))$-rainbow.

LEMMA 4. *There exists an explicit construction for a* $(c=\log n, m, n, O(1))$*-rainbow if* m *is* $n^{\text{polylog}(n)}$.

Proof. For $1 \leqq x \leqq m$ let x_i denote the ith bit of x. Consider the coloring of pairs that assigns the pair (x, y) $\min_{1\leqq i \leqq \log m} x_i \neq y_i$, i.e., the first bit in which x and y differ.

CLAIM 3. *In any set* $S \subset \{1, \cdots, m\}$ *of size* n, *the pairs must be colored with at least* $\log n$ *different colors.*

To see that the claim is true, consider organizing the elements of S in a trie, i.e., in a binary tree where each element appears as a leaf and its value is determined by the path from the root. If a node in the ith level of the trie has two children, then there is a pair $\langle x, y\rangle$, where x is a descendent of the left child and y a descendent of the right child, that is, colored i. There must be at least $\log n$ levels in which there is a node with 2 children, since each level at most doubles the number of nodes from the previous one and there are n leaves.

The claim shows that rather than having a set of size n out of a domain of size m, the problem can be reduced to that of a set of size $\log n$ from a domain of size $\log m$. If m is $n^{\text{polylog}(n)}$, then $\log m$ is polynomial in $\log n$, and hence the construction of Theorem 6 can be applied to obtain the required rainbow.

Sipser [7] gave a probabilistic construction for an $(n^{\log n}, n, \log^2 n, n, n/2)$-disperser. Given such a disperser D, we now show how to use such dispersers to amplify rainbows and construct $(c=n,\ m=n^{\log n},\ n,\ O(1))$-rainbows. Let $\mathscr{C}$ be a $(c=\log^2 n,\ m,\ n-1,\ t)$-rainbow whose existence is assured by the previous lemma and the remark following Theorem 6. Consider a coloring of $t+1$-tuples over $\{1, \cdots, m\}$ defined as follows. The first t elements are used to obtain a color $e \in \{1 \cdots \log^2 n\}$ via $\mathscr{C}$. The $t+1$st element, $v \in \{1, \cdots, m\}$ is treated as a node on the left side of D. e specifies a neighbor of v on the right side of D. The neighbor is the color of the t-tuple. Since any set of n nodes on the left side is adjacent to at least half the nodes on the right side, and since $\mathscr{C}$ is a $(\log^2 n, m, n, t)$-rainbow, it follows that for any set $S \subset \{1, \cdots, m\}$ of size n there is a set $T \subset \{1, \cdots, n\}$ of size at least $n/2$ such that we can specify in this manner any member of T. Using the construction of Lemma 2, this can be amplified to include all the n nodes on the right. This construction gives a $(c=n,\ m=n^{\log n},\ n,\ 2t+2)$-rainbow. Therefore, an explicit construction for a disperser with those parameters yields an implicit $O(1)$ probe search scheme for $m=n^{\log n}$.

No explicit construction with parameters close to the ones given in [7] is known. The best explicit construction for such expanders is given in Ajtai, Komlós, and Szemerédi [1].

8. Conclusions and open problems.

As a consequence of the results of this paper, the maximal m for which implicit $O(1)$ probe is possible lies between $2m^{\text{poly}(n)}$ and a constant height tower of powers. One obvious, open problem is to close this gap. Finding an explicit construction for rainbows with m superpolynomial in n is another obvious research direction. A different question is whether rainbows are useful for implicit data representation in other settings.

Note added in proof. Zuckerman [9] has found some constructions related to those in § 7.

Acknowledgments. We thank Noga Alon, Joel Friedman, Nati Linial, Mike Luby, Jeanette P. Schmidt, Alan Siegel, and Avi Wigderson for helpful discussions and advice and are grateful to the two anonymous referees for their diligent reading and many useful remarks.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi, *Deterministic simulation in* LOGSPACE, in Proc. 19th ACM Symposium on Theory of Computing, 1987, pp. 132-140.

[2] A. Borodin, F. E. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson, *Tradeoff between search and update time for the implicit dictionary problem*, Theoret. Comput. Sci., 58 (1988), pp. 57-68.

[3] A. Cohen and A. Wigderson, *Multigraph amplification*, 1989, manuscript.

[4] A. Fiat, M. Naor, J. P. Schmidt, and A. Siegel, *Non-oblivious hashing*, in Proc. 20th ACM Symposium on Theory of Computing, Chicago, IL, pp. 367-376.

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

[6] R. L. Graham, B. L. Rothschild, and J. H. Spencer, *Ramsey Theory*, Wiley, New York, 1980.

[7] M. Sipser, *Expanders, Randomness or time versus space*, J. Comput. System Sci., 36 (1988), pp. 379-383.

[8] A. C. Yao, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615-628.

[9] D. Zuckerman, *Stimulating* BPP *using a general weak random source*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, October 1991, pp. 79-89.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 11–28, February 1993

MAINTAINING THE 3-EDGE-CONNECTED COMPONENTS OF A GRAPH ON-LINE*

ZVI GALIL† AND GIUSEPPE F. ITALIANO‡

Abstract. The problem of maintaining the 3-edge-connected components of a graph undergoing repeated dynamic modifications, such as edge and vertex insertions, is studied. This paper shows how to answer the question of whether or not two vertices belong to the same 3-edge-connected component of a connected graph that is undergoing only edge insertions. Any sequence of q query and updates on an n-vertex graph can be performed in $O((n + q)\alpha(q, n))$ time.

Key words. analysis of algorithms, dynamic data structures, edge connectivity, vertex connectivity

AMS(MOS) subject classifications. 68P05, 68Q20, 68R10

1. Introduction. Given an undirected graph $G = (V, E)$, and an integer $k \geq 2$, a pair of vertices $\langle u, v\rangle$ is said to be *k-edge-connected* if the removal of any $k - 1$ edges in G leaves u and v connected. This is an equivalence relationship, and we denote it by $\equiv_k$, i.e., if a pair of vertices $\langle x, y\rangle$ is k-edge-connected we write $x \equiv_k y$. The vertices of a graph G are partitioned by this relationship into equivalence classes called *k-edge-connected components*. An edge set $E' \subseteq E$ is an *edge-cut* for vertices x and y if the removal of all the edges in E' disconnects x and y. The cardinality of an edge-cut E', denoted by $|E'|$, is given by the number of edges in E'. An edge-cut E' for x and y is said to be a *minimum cardinality edge-cut* or in short a *min-edge-cut* if there is no other edge-cut E'' for x and y such that $|E''| < |E'|$. Then $x \equiv_k y$ if and only if a min-edge-cut for x and y contains at least k edges. A graph G is said to be *k-edge-connected* if all its pairs of vertices are k-edge-connected. A min-edge-cut of cardinality 1 is called a *bridge*. Similarly, a vertex set $V' \subseteq V - \{x, y\}$ is a *vertex-cut* for vertices x and y if the removal of all the vertices in V' disconnects x and y. The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for x and y is said to be a *minimum cardinality vertex-cut* or in short a *min-vertex-cut* if there is no other vertex-cut V'' for x and y such that $|V''| < |V'|$. Then x and y are k-vertex-connected if and only if a min-vertex-cut for x and y contains at least k vertices. A graph G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected. A min-vertex-cut of cardinality 1 is called an *articulation point*.

The following theorems, due to Menger; Ford and Fulkerson; Elias, Feinstein, and Shannon (see, for instance, [17]), give another characterization of k-vertex and k-edge connectivity.

THEOREM 1.1 (Menger). *Given a graph G and two vertices x and y in G, x and y are k-vertex-connected if and only if there are at least k vertex-disjoint paths between x and y.*

THEOREM 1.2 (Ford and Fulkerson; Elias, Feinstein, and Shannon). *Given a graph G and two vertices x and y in G, x and y are k-edge-connected if and only if there are at*

*Received by the editors March 8, 1991; accepted for publication (in revised form) August 12, 1991. This work was partially supported by National Science Foundation grants CCR-8814977 and CCR-9014605, by the ESPRIT II Basic Research Actions Program of the EC under contract 3075 (Project ALCOM), and by the Italian MURST Project "Algoritmi e Strutture di Calcolo." Portions of this paper appear in the Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1991, pp. 317–327.

†Department of Computer Science, Columbia University, 450 Computer Science Building, New York, New York 10027, and Tel-Aviv University, Tel-Aviv, Israel.

‡Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Rome, Italy. Present address, IBM T. J. Watson Research Center, Yorktown Heights, New York 10598. Part of this work, which was done while the author was at Columbia University, was partially supported by an IBM Graduate Fellowship.

least k edge-disjoint paths between x and y.

Vertex connectivity and edge connectivity problems arise naturally in many applications and have been extensively studied. Much attention has been given to the 2-vertex-connected (or biconnected) components and 2-edge-connected (or bridge-connected) components of a graph. Tarjan [32] gives optimal linear-time sequential algorithms. Tarjan and Vishkin [34] give logarithmic-time parallel algorithms. Recently, Westbrook and Tarjan [36] considered the problem of maintaining biconnected components and bridge-connected components undergoing any sequence of edge and vertex insertions. They presented algorithms that run in a total of $O(q\alpha(q,n))$ time, where q is the total number of operations, n is the number of vertices, and α is a functional inverse of Ackermann's function. As for 3-vertex-connectivity, Hopcroft and Tarjan [18] showed how to compute the triconnected components of a graph in linear time. Di Battista and Tamassia [6] showed how to maintain the triconnected components of a graph during edge insertions in a total of $O(q + n\log n)$ time[1], where q is the total number of operations performed and n is the number of vertices. Their algorithm achieves an $O(q\alpha(q,n))$ bound only in the case of an initially biconnected graph.

In the last decade there has been a growing interest in dynamic problems on graphs. In particular, much attention has been devoted to (among others) the dynamic maintenance of connected components [10], [11], [26], [28]; 2- and 3-connectivity [6], [15]; [35], [36]; transitive closure [2], [19], [20], [21], [24], [31], [37]; planarity [5], [6], [30]; shortest paths [1], [4], [9], [25], [27], [29], [37]; and minimum spanning trees [3], [8], [11], [29]. In these problems we would like to answer queries on graphs that are undergoing a sequence of updates, for instance, insertions and deletions of edges and vertices.

A problem is said to be *fully dynamic* if the update operations include both insertions and deletions of edges. On the other hand, a problem is called *partially dynamic* if only one type of update, i.e., either insertions or deletions, is allowed. The goal of a (fully or partially) dynamic graph algorithm is to update efficiently the solution of a problem after dynamic changes, rather than having to recompute it from scratch each time. Given their powerful versatility, it is not surprising that dynamic algorithms and dynamic data structures are usually more difficult to design than their static counterparts.

In this paper we study the partially dynamic problem of maintaining the 3-edge-connected components of a connected graph during edge insertions. We wish to maintain the graph under an intermixed sequence of operations of the following kinds.

Same3EdgeBlock(u,v): Return *true* if vertices u and v are in the same 3-edge-connected component. Return *false* otherwise.

InsertEdge(x,y): Insert a new edge between the two vertices x and y.

AddVertex(u,v): Add a new vertex u to the graph, and connect it to G by means of the edge (u,v).

We give algorithms that support any sequence of q *InsertEdge*, *AddVertex*, and *Same3EdgeBlock* operations on an initially connected graph with n vertices in a total of $O((n+q)\alpha(q,n))$ time. Notice that the best-known bound for the twin problem of maintaining in a partially dynamic fashion the triconnected components of a graph is $O(q + n\log n)$ [6], and the algorithms given in [6] achieve the $O(q\alpha(q,n))$ bound only in the case of an initially biconnected graph.

Our techniques combine a variety of graph properties, and data structures. We represent the 3-edge-connected components of a graph by means of a tree structure, and

[1]In this paper all the logarithms are assumed to be base two.

update and query this tree instead of the original graph. Indeed, we represent a graph G by a tree of 2-edge-connected components, each of which is itself represented as a tree of 3-edge-connected components. We then develop efficient data structures for updating this representation during edge and vertex insertions.

The remainder of the paper consists of four sections. Section 2 gives some graph-theoretical properties of k-edge-connectivity. Section 3 gives our bounds for 2-edge-connected graphs, and §4 extends these bounds to connected graphs. In §5 we list some concluding remarks.

2. Some properties of k-edge connectivity. Let $G = (V, E)$ be an undirected graph. Let $V_1, V_2, \dots, V_p$ be a partition of the vertices of $G = (V, E)$ such that if $x, y \in V_i$, $1 \leq i \leq p$, then $x \equiv_k y$. Notice that $V_1, V_2, \dots, V_p$ are not necessarily the k-edge-connected components of G since we can have $u \equiv_k v$ for $u \in V_i$ and $v \in V_j$, $i \neq j$. Let $\varphi(G) = (\varphi(V), \varphi(E))$ be the graph obtained from G by shrinking each V_i, $1 \leq i \leq p$, into one super-vertex $\varphi(V_i)$. If $e = (u, v)$ is an edge of G, $u \in V_i$, $v \in V_j$, let $\varphi(e) = (\varphi(V_i), \varphi(V_j))$ be the corresponding edge in $\varphi(G)$. Notice that $\varphi(G)$ can contain multiple edges and self-loops. For any $E' \subseteq E$, let $\varphi(E') = \{\varphi(e) | e \in E'\}$. As a consequence of this definition, $|\varphi(E')| = |E'|$ for any edge set $E' \subseteq E$. Similarly, if $x \in V_i$ we define $\varphi(x) = \varphi(V_i)$. In the sequel such a φ will be referred to as a *k-edge-connectivity obeying mapping*.

The following lemmas characterize some combinatorial properties of the k-edge-connected components of a graph.

LEMMA 2.1. *Let φ be a k-edge-connectivity obeying mapping. A set of edges E', $|E'| \leq k - 1$, is an edge-cut for x and y in G if and only if $\varphi(E')$ is an edge-cut for $\varphi(x)$ and $\varphi(y)$ in $\varphi(G)$.*

Proof. Assume $\varphi(E')$ is not an edge-cut for $\varphi(x)$ and $\varphi(y)$ in $\varphi(G)$. Thus, there is a path $\varphi(\pi)$ in $\varphi(G)$ between $\varphi(x)$ and $\varphi(y)$ containing no edges in $\varphi(E')$. Let $\varphi(e_1)$, $\varphi(e_2), \dots, \varphi(e_p)$ be the edges in $\varphi(\pi)$, and let $e_1, e_2, \dots, e_p$ be the corresponding edges in G. Clearly, $e_i \notin E'$, $1 \leq i \leq q$. However, $e_1, e_2, \dots, e_p$ does not give a path in G. Let $e_i = (u_i, v_i)$, $1 \leq i \leq q$. Denote x by v_0 and y by u_{p+1}. By definition of $\varphi(G)$, $v_i \equiv_k u_{i+1}$, $0 \leq i \leq p$. By Theorem 1.2, there are at least k edge-disjoint paths in G between v_i and u_{i+1}, $0 \leq i \leq p$. Since $|E'| \leq k - 1$, there is at least one path π_i between v_i and u_{i+1}, $0 \leq i \leq p$, containing no edges in E'. As a result, $\pi_0 \cdot e_1 \cdot \pi_1 \cdot e_2 \cdot \dots \cdot e_p \cdot \pi_p$ gives a path in G between x and y, avoiding edges of E'. Hence E' is not an edge-cut for x and y in G.

Conversely, assume E' is not an edge-cut for x and y in G. That is, there is a path $\pi = \{x = v_0, v_1, \dots, v_{q-1}, v_q = y\}$ in G containing no edges of E'. Then $\varphi(\pi) = \{\varphi(x), \varphi(v_1), \dots, \varphi(v_{q-1}), \varphi(y)\}$ gives a path between $\varphi(x)$ and $\varphi(y)$ in $\varphi(G)$ that uses no edges of $\varphi(E')$. Therefore, $\varphi(E')$ cannot be an edge-cut for $\varphi(x)$ and $\varphi(y)$ in $\varphi(G)$. □

LEMMA 2.2. *Let φ be a k-edge-connectivity obeying mapping. Let $\varphi(V_i)$ and $\varphi(V_j)$ be any two vertices in $\varphi(G)$, and let x and y be any two vertices of G, $x \in V_i$, $y \in V_j$. Given any integer h, $1 \leq h \leq k$, $\varphi(V_i)$ and $\varphi(V_j)$ are h-edge-connected in $\varphi(G)$ if and only if x and y are h-edge-connected in G.*

Proof. If $V_i = V_j$, the lemma is trivial.

Assume $i \neq j$ and let V_i and V_j be such that $\varphi(V_i) \not\equiv_h \varphi(V_j)$ in the contracted graph $\varphi(G)$. This means that there is a min-edge-cut for $\varphi(V_i)$ and $\varphi(V_j)$ in $\varphi(G)$ containing at most $h - 1$ edges. Let A be such such a min-edge-cut, $|A| \leq h - 1$. Let x be any vertex in V_i, and let y be any vertex in V_j. Let $\bar{E} = \{e \in E | \varphi(e) \in A\}$. By Lemma 2.1, $\bar{E}$ is an edge-cut for x and y in G. Since $|\bar{E}| = |A| \leq h - 1$, $x \not\equiv_h y$ in G.

Let x and y be vertices in G such that $x \not\equiv_h y$. Let $x \in V_i$, $y \in V_j$, $i \neq j$. Let E', $|E'| \leq h - 1$, be an edge-cut for x and y in G. Then by Lemma 2.1, $\varphi(E')$, $|\varphi(E')| = |E'| \leq h - 1$, is an edge-cut for $\varphi(V_i)$ and $\varphi(V_j)$ in $\varphi(G)$. Therefore, $\varphi(V_i) \not\equiv_h \varphi(V_j)$ in $\varphi(G)$. □

3. Maintaining 3-edge-connected components of 2-edge-connected graphs.

In this section we show how to maintain efficiently the 3-edge-connected components of a graph G under any sequence of the operations *InsertEdge*, *AddVertex*, and *Same3EdgeBlock*. We recall here that *InsertEdge*(x, y) adds a new edge between vertices x and y, *AddVertex*(u, v) inserts a new vertex u and connects it to vertex v, and *Same3EdgeBlock* (u, v) returns true if vertices u and v are in the same 3-edge-connected component of G, and it returns false otherwise. We consider first 2-edge-connected graphs and then generalize the results to connected graphs.

Let G be a 2-edge-connected graph with n vertices subject to *Same3EdgeBlock* and *InsertEdge* operations. In this section we do not consider *AddVertex* operations since they do not preserve G 2-edge-connected; we show how to deal with them in the next section.

For any vertex x in G, denote by $C(x)$ the 3-edge-connected component containing x. We show now how to maintain information about 3-edge-connected components of a 2-edge-connected graph G during edge insertions. Let G' be the graph obtained by contracting each 3-edge-connected component of G into a super-vertex. Notice that there can be multiple edges in G'. We use interchangeably the terms 3-edge-connected component of G and super-vertex in G'. By hypothesis G is 2-edge-connected, and by Lemma 2.2 G' is 2-edge-connected, too. As a result, any two super-vertices of G' lie on a common cycle. The following lemma states that G' consists of simple cycles such that any two of them share at most one super-vertex (see also [7], [22]).

LEMMA 3.1. *Two simple cycles of G' can intersect in at most one super-vertex.*

Proof. We proceed by contradiction. Assume there exist two different simple cycles Λ_1 and Λ_2 of G' intersecting at two super-vertices σ_1 and σ_2. Then there must be a subpath of Λ_2 between two vertices σ' and σ'' of Λ_1 (not necessarily σ_1 and σ_2) that is edge-disjoint from Λ_1. As a result, there are three edge-disjoint paths between σ' and σ'' in G'. By Theorem 1.2, $\sigma' \equiv_3 \sigma''$ in G'. By Lemma 2.2, any two vertices x and y of G such that $x \in \sigma'$ and $y \in \sigma''$ are 3-edge-connected. Consequently, σ' and σ'' cannot be 3-edge-connected components of G, a contradiction. □

LEMMA 3.2. *Each biconnected component of G' is a simple cycle.*

Proof. We proceed by contradiction. Let B be a biconnected component of G' that is not a simple cycle. This implies that there are three edges e_1, e_2, and e_3 in B that do not lie in a simple cycle. However, since two edges belong to the same biconnected component if and only if they lie in a same simple cycle, there must be a simple cycle Λ_1 containing both e_1 and e_2, and a simple cycle Λ_2 containing both e_2 and e_3. Since e_2 belongs to both Λ_1 and Λ_2, Λ_1 and Λ_2 must intersect in at least two vertices, which contradicts Lemma 3.1. □

Define $\mathcal{T}$ as the block tree of G'. We recall here that given a graph G, the block tree of G is defined as follows (see, for instance, [17]). The nodes of the block tree are partitioned into *square* and *round* nodes: square nodes correspond to the vertices of G, while round nodes correspond to the biconnected components of G. There is an edge between a square node u and a round node ρ if and only if vertex u of G is in biconnected component ρ. Also, there is an edge between a square vertex u and round vertices ρ_1 and ρ_2 if and only if u is an articulation point separating biconnected components ρ_1 and ρ_2 in G.

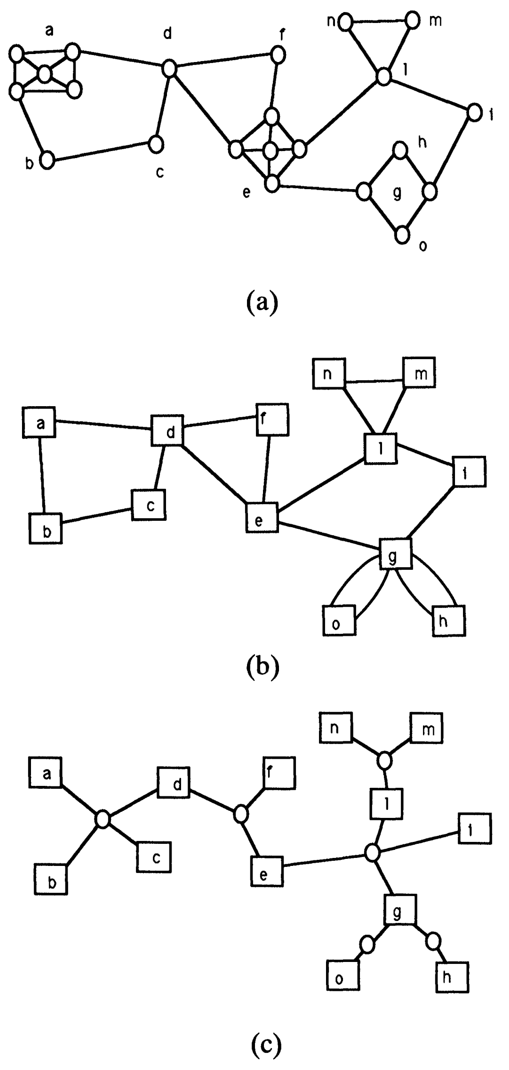


FIG. 1. (a) *A* 2*-edge-connected graph G*; (b) *the graph G'*; *and* (c) *the tree of* 3*-edge-connected components of G.*

In the following, we refer to $\mathcal{T}$ as the *tree of* 3*-edge-connected components* of G. Notice that each 3-edge-connected component of G corresponds uniquely to a super-vertex of G' and to a square node in $\mathcal{T}$. In the remainder of this section we use interchangeably the terms 3-edge-connected component of G, super-vertex of G', and square node of $\mathcal{T}$. Figure 1 shows graphs G and G', as well as the block tree $\mathcal{T}$.

We root the tree $\mathcal{T}$ arbitrarily at any square node. Edges of $\mathcal{T}$ are considered directed from children to parent. Furthermore, for each simple cycle in G', we maintain the same cyclic order of vertices in $\mathcal{T}$ (see Fig. 1). In other words, let $\sigma_1, \sigma_2, \ldots, \sigma_p$ be the vertices of G' encountered in a counterclockwise walk around around a cycle ρ. Then in $\mathcal{T}$ there is one square node σ_ℓ, $1 \leq \ell \leq p$, that is the parent of ρ, and all the children of ρ from left to right are $\sigma_{\ell+1}, \sigma_{\ell+2}, \ldots, \sigma_p, \sigma_1, \sigma_2, \ldots, \sigma_{\ell-1}$. The following corollary follows easily from Lemma 3.2 and the definition of block tree of a graph.

COROLLARY 3.3. *Let x and y be any two vertices in G, $x \not\equiv_3 y$. Let $C(x)$ and $C(y)$ be the 3-edge-connected components containing x and y, respectively. Let $\pi_{x,y}$ be the path in $\mathcal{T}$ between $C(x)$ and $C(y)$ containing vertices $C(x) = \sigma_1, \rho_1, \sigma_2, \ldots, \sigma_{q-1}, \rho_{q-1}, \sigma_q = C(y)$, $q \geq 2$ (σ_i are square nodes and ρ_i are round nodes). Then $\sigma_2, \sigma_3, \ldots, \sigma_{q-1}$ are all the articulation points separating $C(x)$ and $C(y)$ in G', and all the simple paths between $C(x)$ and $C(y)$ in G' are given by the $q-1$ simple cycles $\rho_1, \rho_2, \ldots, \rho_{q-1}$.*

Besides maintaining $\mathcal{T}$, we maintain the actual 3-edge-connected components of G as disjoint sets subject to *union* and *find* operations. The name of any such set gives a pointer to the corresponding square node in $\mathcal{T}$. For each vertex x, *find*(x)=$C(x)$ (i.e., *find*(x) returns the 3-edge-connected component containing vertex x in G). With this data structure, a *Same3EdgeBlock*(x, y) operation can be simply performed by checking whether *find*(x) = *find*(y). The *union* operations will be used to update efficiently the 3-edge-connected components during *InsertEdge* operations. We recall that using the fast set-union data structures of [33] yields that any sequence of q *union* and *find* operations on a collection of n elements can be performed in $O(q\alpha(q, n))$ worst-case time.

Let x and y be any two vertices in G such that $x \not\equiv_3 y$. Denote by $C(x)$ and $C(y)$ the square nodes in $\mathcal{T}$ corresponding to the 3-edge-connected components of x and y, respectively, and let $\pi_{x,y}$ be the path in $\mathcal{T}$ between $C(x)$ and $C(y)$. We now describe the updates needed in $\mathcal{T}$ because of the insertion of a new edge (x, y). As the following lemma shows, the path in $\mathcal{T}$ between $C(x)$ and $C(y)$ plays a crucial role in the update.

LEMMA 3.4. *Let G be a 2-edge-connected graph. After the insertion of (x, y), all the vertices of the 3-edge-connected components corresponding to square nodes in $\pi_{x,y}$ become 3-edge-connected, while there is no change in the other 3-edge-connected components of G.*

Proof. Let $V_1, V_2, \ldots, V_p$ be the 3-edge-connected components of G before the insertion of edge (x, y). Let $V'_1, V'_2, \ldots V'_q$, $1 \leq q \leq p$, be the 3-edge-connected of G corresponding to the square nodes of $\mathcal{T}$ in $\pi_{x,y}$, as they are met while going from $C(x)$ to $C(y)$ in $\mathcal{T}$. Notice that $C(x) = V'_1$ and $C(y) = V'_q$. Let $G^{new} = G \cup \{(x, y)\}$. Similarly, let $(G^{new})'$ be the graph obtained from G^{new} by contracting its 3-edge-connected components.

Consider the graph $G'_{temp} = G' \cup \{(C(x), C(y))\}$. Notice that G'_{temp} is not necessarily $(G^{new})'$. However, since any two vertices that were 3-edge-connected in G are still 3-edge-connected in G^{new}, $V_1, V_2, \ldots, V_q$ gives a partition of the vertices of G^{new} such that if $u, v \in V_i$, then $u \equiv_3 v$. Therefore, G'_{temp} can be derived from G^{new} by applying a 3-edge-connected obeying mapping φ to G^{new}: namely, $G'_{temp} = \varphi(G^{new})$.

Assume first that $C(x) = C(y)$. Since by definition no two vertices of G' are 3-edge-connected, then no two vertices of $\varphi(G^{new})$ are 3-edge-connected either. Therefore, by Lemma 2.2 there is no change in the 3-edge-connected components of G. In this case $\pi_{x,y} = \{C(x)\}$ and the lemma is trivially true.

Assume now $C(x) \neq C(y)$. By Corollary 3.3, all the simple paths between $C(x) = V'_1$ and $C(y) = V'_q$ in G' are given by $q - 1$ simple cycles $\rho_1, \rho_2, \ldots, \rho_{q-1}$ in G' such that $V'_1 \in \rho_1$, $V'_p \in \rho_{q-1}$, $\rho_{i-1} \cap \rho_i = \{V'_i\}$, $2 \leq i \leq q-1$, and $\rho_i \cap \rho_j = \emptyset$, $i \leq j-2$ or $i \geq j+2$. Each ρ_i, $1 \leq i \leq q-1$, is composed of two disjoint paths between V'_i and V'_{i+1}, say, $\rho_i^{(1)}$ and $\rho_i^{(2)}$.

Then there are three edge-disjoint paths between V'_i and V'_j, $1 \leq i < j \leq q$, in $\varphi(G^{new})$. The first is given by $\rho_i^{(1)}, \rho_{i+1}^{(1)}, \ldots, \rho_{j-1}^{(1)}$; the second by $\rho_i^{(2)}, \rho_{i+1}^{(2)}, \ldots, \rho_{j-1}^{(2)}$; and, finally, the third is given by $\rho_{i-1}^{(1)}, \rho_{i-2}^{(1)}, \ldots, \rho_1^{(1)}, (C(x), C(y)), \rho_{q-1}^{(1)}, \rho_{q-2}^{(1)}, \ldots, \rho_j^{(1)}$. Thus $V'_i \equiv_3 V'_j$.

We now show that only the vertices V_i', $1 \le i \le q$, become 3-edge-connected in $\varphi(G^{new})$. Let $\widetilde{V}$ be any vertex of G' different from V_i', $1 \le i \le q$, and let V_i, $1 \le i \le p$, be any vertex of G' different from $\widetilde{V}$. Since vertices of G' correspond to 3-edge-connected components of G, $\widetilde{V} \not\equiv_3 V_i$ in G'. We show that also $\widetilde{V} \not\equiv_3 V_i$ in $\varphi(G^{new})$. By Corollary 3.3, all the simple paths between $\widetilde{V}$ and V_i in G' are given by $p \ge 1$ simple cycles $\rho_1', \rho_2', \dots, \rho_p'$ in G' such that $\widetilde{V} \in \rho_1'$, $V_i \in \rho_p'$. Consequently, if e_1 and e_2 are the two edges of ρ_1' incident to $\widetilde{V}$, $\{e_1, e_2\}$ separates $\widetilde{V}$ and V_i in G'. Therefore, the removal of $\{e_1, e_2\}$ breaks G' into two pieces: G_1' and G_2' such that G_1' contains $\widetilde{V}$ and G_2' contains V_i. We claim that the removal of $\{e_1, e_2\}$ must leave $C(x)$ and $C(y)$ on the same side (i.e., either both in G_1' or both in G_2'). Indeed assume by contradiction that $C(x)$ and $C(y)$ are left in different sides by the removal of $\{e_1, e_2\}$. Without loss of generality, let $C(x)$ be in G_1' and $C(y)$ be in G_2'. This implies that $\{e_1, e_2\}$ is an edge-cut for $C(x)$ and $C(y)$ in G'. Since e_1 and e_2 are both incident to $\widetilde{V}$ in G', also the removal of $\widetilde{V}$ together with its incident edges disconnects $C(x)$ and $C(y)$ in G'. This implies that $\widetilde{V}$ is an articulation point whose removal disconnects $C(x)$ and $C(y)$ in G'. But by Corollary 3.3, this gives $\widetilde{V} = V_i'$, for some i, $1 \le i \le q$, contradicting our assumption. Since $C(x)$ and $C(y)$ must be either both in G_1' or both in G_2', $\{e_1, e_2\}$ still separates $\widetilde{V}$ and V_i in $\varphi(G^{new}) = G' \cup \{C(x), C(y)\}$. Consequently, $\widetilde{V} \not\equiv_3 V_i$ in $\varphi(G^{new})$.

By Theorem 1.2, all and only the vertices $V_1', V_2', \dots, V_q'$ become 3-edge-connected in $\varphi(G^{new})$, while all the remaining vertices of $\varphi(G^{new})$ stay 2-edge-connected. By Lemma 2.2 all and only the vertices of G contained in $V_1', V_2', \dots, V_q'$ become 3-edge-connected. □

Lemma 3.4 gives a way to compute $(G^{new})'$ from G'. As shown in the lemma, all the paths between V_1' and V_q' in G' are given by $q-1$ edge-disjoint simple cycles $\rho_1, \rho_2, \dots, \rho_{q-1}$ in G' such that $V_1' \in \rho_1$, $V_q' \in \rho_{q-1}$, and $\rho_{i-1} \cap \rho_i = \{V_i'\}$, $2 \le i \le q-1$. Each ρ_i is composed of two edge-disjoint paths between V_i' and V_{i+1}', say, $\rho_i^{(1)}$ and $\rho_i^{(2)}$. Furthermore, the insertion of (x, y) in G implies that all and only the vertices in V_i', $1 \le i \le p$, become 3-edge-connected in G. Therefore, $(G^{new})'$ can be computed from G' by simply shrinking all the vertices $V_1', V_2', \dots, V_q'$ into one vertex V'. This destroys the previous simple cycles ρ_i, $1 \le i \le q-1$, and creates at most $2(q-1)$ new simple cycles $\rho_i^{(1)}$ and $\rho_i^{(2)}$, $1 \le i \le q-1$, that are all incident to V' in $(G^{new})'$.

The changes in G' induce updates in $\mathcal{T}$, which is the block tree of G'. We denote by λ the least common ancestor of $C(x)$ and $C(y)$ in $\mathcal{T}$. The update causes the following transformations on the path $\pi_{x,y}$ of $\mathcal{T}$, to obtain $\mathcal{T}^{new}$, the block tree of $(G^{new})'$.

(i) Merge together all the square nodes $V_1', V_2', \dots, V_q'$ in $\pi_{x,y}$ into a new square node V'.

(ii) For each round node $\rho_i \ne \lambda$ in $\pi_{x,y}$, denote by V_i' and V_{i+1}' the two square nodes in $\pi_{x,y}$ adjacent to ρ_i. Without loss of generality, let V_{i+1}' be the parent of ρ_i in $\pi_{x,y}$. Let $a_1^{(i)}, \dots, a_s^{(i)}, V_i', b_1^{(i)}, \dots, b_t^{(i)}$, $s, t \ge 0$, be the children of ρ_i sorted from left to right. Split ρ_i into two round nodes $\rho_i^{(1)}$ and $\rho_i^{(2)}$: make $a_1^{(i)}, \dots, a_s^{(i)}$ children of $\rho_i^{(1)}$ and $b_1^{(i)}, \dots, b_t^{(i)}$ children of $\rho_i^{(2)}$. Make $\rho_i^{(1)}$ and $\rho_i^{(2)}$ children of V'. If $s = 0$ [$t = 0$], delete $\rho_i^{(1)}$ [$\rho_i^{(2)}$].

(iii) If λ is a round node, again denote by V_i' and V_{i+1}' the two square nodes in $\pi_{x,y}$ adjacent to λ. Without loss of generality, let $a_1^{(i)}, \dots, a_s^{(i)}$, V_i', $b_1^{(i)}, \dots, b_t^{(i)}$, V_{i+1}' $c_1^{(i)}, \dots, c_u^{(i)}$, $s, t, u \ge 0$, be the children of λ sorted from left to right. Split λ into two round nodes $\lambda^{(1)}$ and $\lambda^{(2)}$: make $a_1^{(i)}, \dots, a_s^{(i)}$, V', $c_1^{(i)}, \dots, c_u^{(i)}$ children of $\lambda^{(1)}$ and

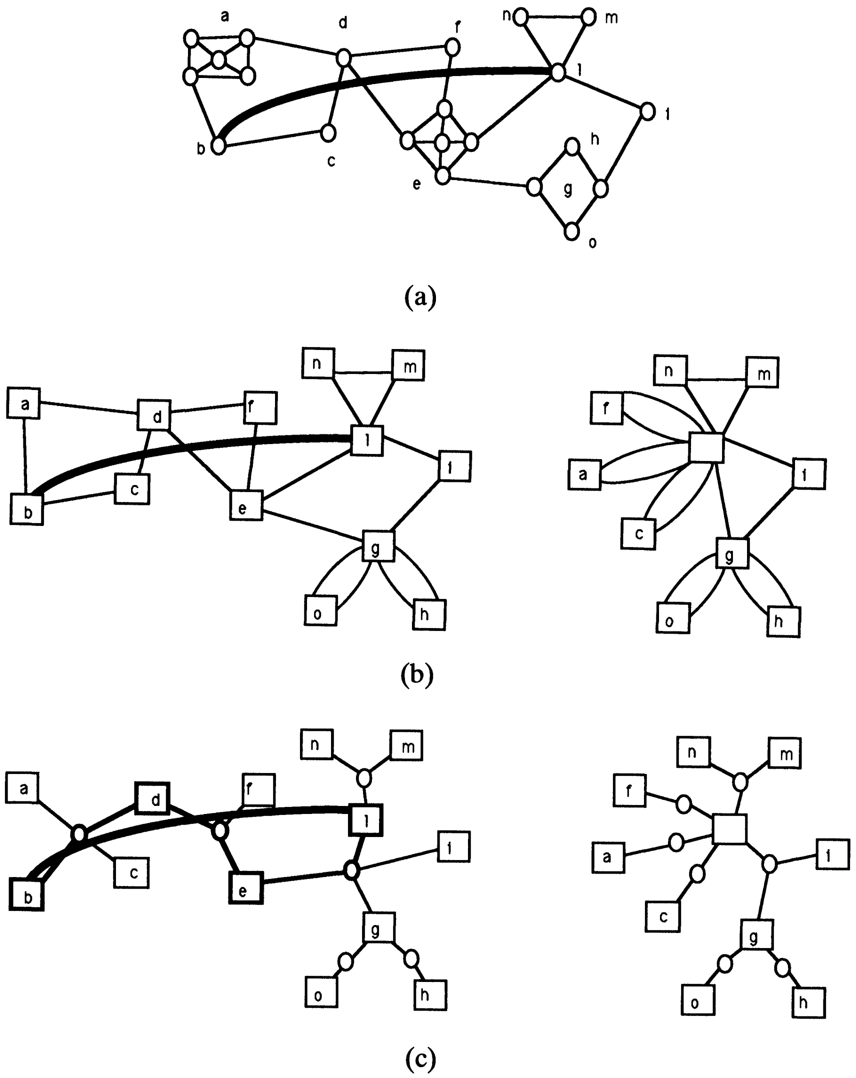


FIG. 2. (a) *G after the insertion of a new edge*; (b) *updates in the graph G' because of the new edge*; *the new G' is shown to the right*; (c) *updates in T because of the new edge*; *the path involved in the update is shown in bold*; *the new T is shown to the right.*

$b_1^{(i)}, \ldots, b_t^{(i)}$ children of $\lambda^{(2)}$. Make $\lambda^{(1)}$ child of the previous parent of λ and $\lambda^{(2)}$ child of V'. If $t = 0$, delete $\lambda^{(2)}$. Notice that since $\mathcal{T}$ is rooted at a square node, λ cannot be the root of $\mathcal{T}$ and thus must have a parent.

The sequence of above transformations, plus the actual merging of the 3-edge-connected components, is referred to in the sequel as *path compression*. Figure 2 shows the updates needed in the graph of Fig. 1, as a result of the insertion of a new edge.

To implement the above rules, we need to support the following primitives on the tree $\mathcal{T}$ during an *InsertEdge* operation:

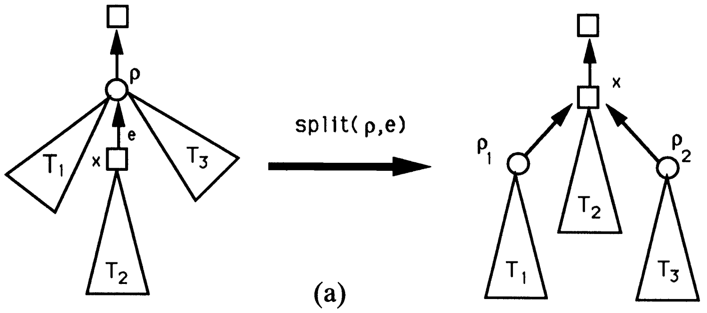
split(ρ, e): Given a round node ρ and an edge $e = (x, \rho)$, where x is a square node child of ρ, split ρ into two round nodes ρ_1 and ρ_2: the edges previously to the left [right] of

e are now incident to ρ_1 [ρ_2]. Make ρ_1 and ρ_2 children of x, and x child of the former parent of ρ (see Fig. 3(a)). If either ρ_1 or ρ_2 has no children, then delete it. Since ρ is a round node, it cannot be the root; therefore, the parent of ρ is always defined.

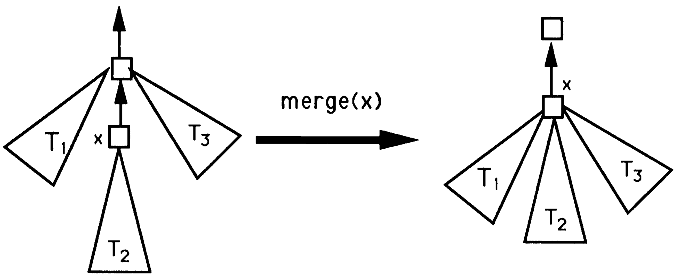
merge(v): Given a square node v, whose parent is a square node itself, contract the edge $(v, \mathit{parent}(v))$ (see Fig. 3(b)), and union the corresponding 3-edge-connected components.

parent(v): Given any (round or square) node v, return its parent.

Notice that the primitive *merge*(v) is not applicable to the tree $\mathcal{T}$ itself since in $\mathcal{T}$ each square node has a round parent. However, a *split*(ρ, e) produces two adjacent square nodes as shown in Fig. 3 and, therefore, allows a primitive *merge* to be applied in the resulting tree.



(a)



(b)

FIG. 3

We are now able to describe our implementation of an *InsertEdge*(x, y) operation. First we perform a *Same3EdgeBlock*(x, y) operation: if it returns true, then x and y are in the same 3-edge-connected component and by Lemma 3.4 nothing need be done. Otherwise, x and y are in different 3-edge-connected components. We proceed as follows. Given $\mathcal{T}$, we first locate the path $\pi_{x,y}$. Then we perform path compression on $\pi_{x,y}$ by doing *merge* operations on the square nodes of $\pi_{x,y}$ and *split* operations on the round nodes of $\pi_{x,y}$. We do this as follows. Denote by λ the least common ancestor of $C(x)$ and $C(y)$ in $\mathcal{T}$. Set $\beta = C(x)$ and $\gamma = \mathit{parent}(\beta)$. While $\gamma \neq \lambda$ repeat the following step:

If γ is a round node, then set $e = (\beta, \gamma)$ and perform *split*(γ, e). If γ is a square node, then perform *merge*(β). In both cases set $\gamma = parent(\beta)$.

Denote by β_1 the square node child of λ at which the preceding step stops. Now set $\beta = C(y)$ and repeat the same splits and merges in the path from $C(y)$ to λ. Similarly, denote by β_2 the square node child of λ in this path. If λ is a round node, then split it as shown in Fig. 4(a). Otherwise, λ is a square node; merge β_1, β_2, and λ as shown in Fig. 4(b). Notice that those two operations can be implemented by using a constant number of *merge* and *split* primitives. In the following, we refer to the update of $\mathcal{T}$ after an *InsertEdge*(u, v) operation as *CompressPath*$(u, v, \mathcal{T})$.

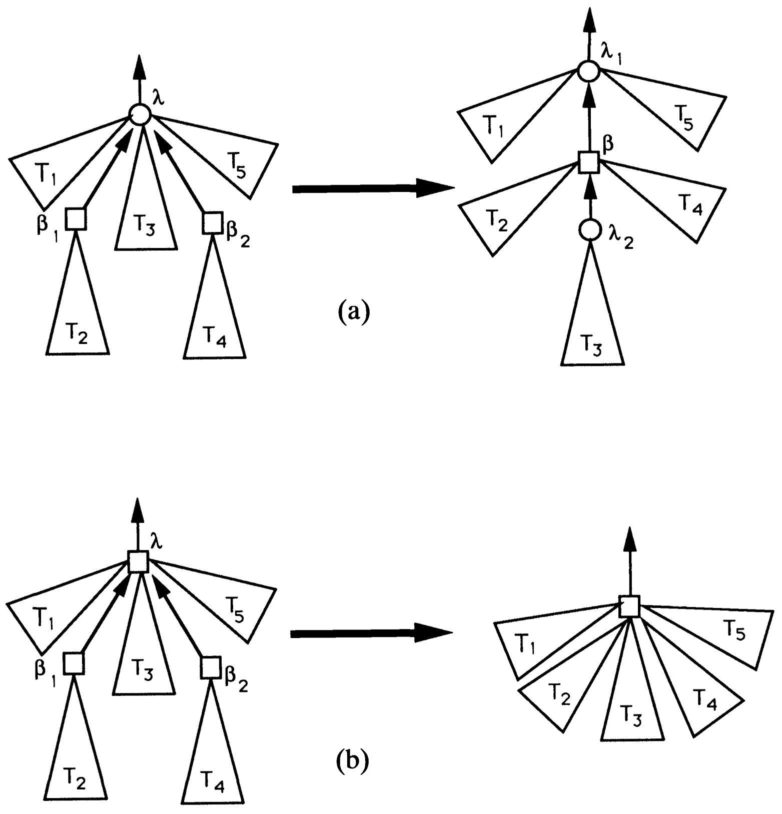


FIG. 4

The following lemma shows how to perform efficiently any sequence of *merge*, *split*, and *parent* operations.

LEMMA 3.5. *Any sequence of p merge, split, and parent primitives on a tree with n nodes requires a total of $O(p\alpha(p, n))$ worst-case time.*

Proof. To support efficiently these primitives, we use different data structures for each node of $\mathcal{T}$. We represent square nodes as *condensible nodes* [36], which enable us to merge two square nodes in $O(\alpha(p, n))$ amortized time at the price of spending $O(\alpha(p, n))$ amortized time to find the parent of a round node. The idea behind a condensible square node is to group all the sibling round nodes into disjoint sets and to maintain those disjoint sets under *union* and *find* operations. We do this as follows. By definition, the children of each square node in $\mathcal{T}$ are all round nodes. Let v be any square node in $\mathcal{T}$, with children $\rho_1, \rho_2, \ldots, \rho_\ell$, $\ell \geq 1$. We maintain a set $\mathcal{UF}(v)$ such that the name of $\mathcal{UF}(v)$ returns a pointer to v. Let ρ_i be a child of v: instead of having an edge from ρ_i to v, we have an edge from ρ_i to an item in the set $\mathcal{UF}(v)$.

We refer to [36] for the details of this method. It suffices to say that a *merge*(v) operation corresponds to a *union* between disjoint sets $\mathcal{UF}(v)$ and $\mathcal{UF}(parent(v))$, and to a *union* of the 3-edge-connected components corresponding to v and $parent(v)$. We remark here that unioning disjoint sets $\mathcal{UF}(v)$ and $\mathcal{UF}(parent(v))$ implies that the space used by the item corresponding to v in $\mathcal{UF}(parent(v))$ is left unused. This implies an $O(1)$ extra space per *merge* operation.

Finding the parent of a round node corresponds to a *find* operation. If for each square node $v \in \mathcal{T}$ we implement $\mathcal{UF}(v)$ by means of a fast set-union data structure [33], each *merge* and *parent* primitive can be supported in $O(\alpha(p, n))$ amortized time.

For splitting round nodes efficiently we use a similar data structure, which we call *expandable node*, and which allows us to split a round node and to find the parent of a square node in constant amortized time. The expandable node data structure has much the same flavor as condensible nodes, but it is based upon set-splitting data structures. We recall here that set-splitting data structures are able to perform *find* and *split* operations on disjoint sets in constant amortized time [12]. Now we group all the sibling square nodes into disjoint sets, and we maintain those disjoint sets under sequences of *split*, and *find* operations. Let ρ be any round node in $\mathcal{T}$, with children $v_1, v_2, \ldots, v_\ell$, $\ell \geq 1$. We maintain a set $\mathcal{SF}(\rho)$ containing ℓ items and such that the name of $\mathcal{SF}(\rho)$ returns a pointer to ρ. Let v_i be any child of ρ: instead of having an edge from v_i to ρ, we have an edge from v_i to the corresponding item in the set $\mathcal{SF}(\rho)$. We implement *split*(ρ, e) as follows. Let v be the square node child of ρ such that $e = (v, \rho)$, and let u be the square node parent of ρ. We perform the corresponding *split* operation in $\mathcal{SF}(\rho)$: this creates two new round nodes ρ_1 and ρ_2 and destroys ρ. To make ρ_1 and ρ_2 children of v we union $\mathcal{UF}(v)$ with two singleton sets. Finally we make v child of u by inserting an edge from u to the item of $\mathcal{UF}(u)$ that round node ρ was pointing to. Therefore, splitting a round node ρ corresponds to performing a *split* operation in $\mathcal{SF}(\rho)$ and at most two union operations involving $\mathcal{UF}(v)$. Finding the parent of a square node corresponds to a *find* operation in a set-splitting data structure.

Notice that due to the structure of $\mathcal{T}$, in which after each update every path alternates between square and round nodes, there is no interaction between condensible and expandable nodes, since only round nodes are subject to splits, and only square nodes are subject to merges. Each *merge* requires unioning two condensible nodes, while each *split* can be performed by splitting an expandable node and by performing a constant number of unions on condensible nodes. Each *parent* requires a *find* operation in either a set-union or a set-splitting data structure. Therefore, by using condensible and expandable nodes we are able to perform each sequence of p *merge*, *split*, and *parent* primitives on a tree with n nodes in a total of $O(p\alpha(p, n))$ worst-case time. $\square$

Lemma 3.5 allows us to prove the following theorem.

THEOREM 3.6. *The data structure supports any sequence of q Same3EdgeBlock and InsertEdge operations on an initially 2-edge-connected graph with n vertices in a total of $O((q + n)\alpha(q, n))$ time. The space required is $O(n)$.*

Proof. A *Same3EdgeBlock* operation consists of performing two *find* operations in disjoint sets and, therefore, can be performed in $O(\alpha(q, n))$ amortized time. As for an *InsertEdge*(x, y) operation, it requires (i) finding the path $\pi_{x,y}$ in $\mathcal{T}$, (ii) performing the merging and splitting of square and round nodes in $\pi_{x,y}$, and (iii) unioning the disjoint sets corresponding to the square nodes (3-edge-connected components) in $\pi_{x,y}$. Given $\mathcal{T}$, we can locate the path $\pi_{x,y}$ by first performing *find*(x) and *find*(y). This returns $C(x)$ and $C(y)$, the two square nodes corresponding to the 3-edge-connected components containing x and y, respectively. Then we trace the paths from $C(x)$ to the root of $\mathcal{T}$ and from $C(y)$ to the root of $\mathcal{T}$, alternating among them one edge at the time. We stop when we reach a node already visited. This requires $2|\pi_{x,y}|$ *parent* primitives in the worst case. Then we have to perform at most $|\pi_{x,y}|$ merges and splits along this path. We recall that *merge* operations union all the 3-edge-connected components in $\pi_{x,y}$. Henceforth, the whole operation can be implemented by using no more than $3|\pi_{x,y}|$ *merge*, *split*, and *parent* primitives. As a consequence of Lemma 3.5, it can be done in a total of $O(|\pi_{x,y}|\alpha(q,n))$ time.

The total time spent during a sequence of at most q *Same3EdgeBlock* and *InsertEdge* operations is, therefore, $O((q+T(n))\alpha(q,n))$, where

$$T(n) \le \sum_{(x,y)} 3|\pi_{x,y}|$$

is the total number of *merge*, *split*, and *parent* primitives performed during *InsertEdge* operations.

We prove the claimed time bound by showing that $T(n) = O(n)$. Notice that the path $\pi_{x,y}$ in $\mathcal{T}$ is a path between square node $C(x)$ and square node $C(y)$. Since any path in $\mathcal{T}$ alternates between square and round nodes, $\pi_{x,y}$ contains at least $\lceil |\pi_{x,y}|/2 \rceil + 1$ square nodes (i.e., 3-edge-connected components of G). After we perform path compression on $\pi_{x,y}$, we are left with only one square node. Consequently, the number of 3-edge-connected components of G decreases by at least $\lceil |\pi_{x,y}|/2 \rceil$. Since at the beginning G can have at most n 3-edge-connected components, and each time we perform *InsertEdge*(x,y) we decrease this number by at least $\lceil |\pi_{x,y}|/2 \rceil$,

$$T(n) \le \sum_{(x,y)} 3|\pi_{x,y}| = O(n).$$

As for the space complexity, we have to bound the size of the set-union data structures used to represent the 3-edge-connected components, and the size of the tree $\mathcal{T}$. Since each 3-edge-connected component of G consists of at least one vertex, and any two 3-edge-connected components are disjoint, the total space required by the set-union data structures used to store the 3-edge-connected components is $O(n)$. As for the size of the tree $\mathcal{T}$, we bound the number of square and round nodes in $\mathcal{T}$. Each square node of $\mathcal{T}$ uniquely corresponds to a vertex of G' (i.e., to a 3-edge-connected component of G), while each round node of $\mathcal{T}$ corresponds to a biconnected component of G'. Since we have at most $O(n)$ 3-edge-connected components in G, there can be at most $O(n)$ vertices and $O(n)$ biconnected components in G'. Therefore, there can be at most $O(n)$ square and round nodes in $\mathcal{T}$. Since $\mathcal{T}$ has at most $O(n)$ nodes, and each *merge* primitive implies an $O(1)$ extra space, $\mathcal{T}$ can be implemented using condensible and expandable nodes in $O(n)$ space as shown in [36]. □

4. Maintaining 3-edge-connected components of connected graphs. We now extend the previous result to the case of connected (but not necessarily 2-edge-connected) graphs. We start with a singleton vertex, and we would like to perform any intermixed sequence of *Same3EdgeBlock*, *InsertEdge*, and *AddVertex* operations.

In this case, we maintain the bridge-block tree $\mathcal{T}_1$ of G. We recall that each node B of $\mathcal{T}_1$ corresponds to a 2-edge-connected component of G, referred to as $graph(B)$, and each edge of $\mathcal{T}_1$ corresponds to a bridge of G. Since $graph(B)$ is a 2-edge-connected component of G, it is 2-edge-connected. We maintain for each node B of $\mathcal{T}_1$ the tree $\mathcal{T}_2(B)$ of 3-edge-connected components of $graph(B)$ as defined in the previous section.

Thus, we maintain information about the graph G by using a tree of trees. Each of these trees is rooted as follows. $\mathcal{T}_1$ is rooted arbitrarily at any node C, and $\mathcal{T}_2(C)$ is rooted arbitrarily at any square node. For each nonroot node B of $\mathcal{T}_1$, let (x_B, y_B) be the bridge in G corresponding to the edge $(B, parent(B))$ in $\mathcal{T}_1$, $x_B \in B$, $y_B \notin B$. We root $\mathcal{T}_2(B)$ at $C(x_B)$ (i.e., the square node corresponding to the 3-edge-connected component containing vertex x_B). Besides this data structure, we maintain also the 2-edge-connected components of G by using the algorithm of Westbrook and Tarjan [36].

As in the previous section, let G' be the graph obtained by shrinking each 3-edge-connected component of G into a super-vertex. Notice that this time G is not necessarily 2-edge-connected and, therefore, G' is not necessarily 2-edge-connected. However, by Lemma 2.2 there is a one-to-one correspondence between bridges of G' and bridges of G, and between 2-edge-connected components of G' and 2-edge-connected components of G. Consequently, $\mathcal{T}_1$ is the bridge-block tree of G' as well, and given any 2-edge-connected component B of G', $\mathcal{T}_2(B)$ is the block tree of B. We notice that combining $\mathcal{T}_1$ and all the trees $\mathcal{T}_2(B)$ yields a tree $\mathcal{T}$ that resembles the block tree of G'. Indeed, denote by $\mathcal{T}$ the tree obtained by plugging each tree $\mathcal{T}_2(B)$ in place of the corresponding node in $\mathcal{T}_1$: the only difference between $\mathcal{T}$ and the block tree of G' is that each bridge (u, v) of G' is represented by a simple edge (u, v) in $\mathcal{T}$, while there is a square node σ and edges (u, σ) and (σ, v) in the block tree of G'. Finally, we remark that Lemma 3.1 still holds for G'. As far as Lemma 3.2 is concerned, we have now that each biconnected component of G' consists of either a unique edge or a simple cycle.

We show how to perform the three operations. A *Same3EdgeBlock*(u, v) is carried out as follows. Find the 2-edge-connected components of G containing u and v, say, B_u and B_v by using the algorithm of Westbrook and Tarjan [36]. If $B_u \neq B_v$, then u and v are not even in a same 2-edge-connected component of G, and, therefore, they cannot be in the same 3-edge-connected component of G, and we return *false*. Otherwise, u and v are in a same 2-edge-connected component $B = B_u = B_v$ of G. We now perform *Same3EdgeBlock*(u, v) on a 2-edge-connected graph as explained in the previous section.

To support an *InsertEdge*(u, v), we first find the 2-edge-connected components of G containing vertices u and v, say, B_u and B_v. If $B_u = B_v = B$, then we perform *CompressPath*$(u, v, \mathcal{T}_2(B))$ as shown in the previous section.

If $B_u \neq B_v$, let $B_u = B_0, B_1, \ldots, B_\ell = B_v$ be the nodes in the path $\pi_{u,v}$ in $\mathcal{T}_1$ between B_u and B_v as they are met while going from B_u to B_v. Each B_i corresponds to the 2-edge-connected component $graph(B_i)$ of G. For each B_i let (x_i, y_i) be the edge in G between $graph(B_i)$ and $graph(B_{i+1})$, $0 \leq i \leq \ell - 1$. For $1 \leq i \leq \ell - 1$ define $start(B_i) = y_{i-1}$ and $end(B_i) = x_i$. Notice that for each 2-edge-connected component $graph(B_i)$, $1 \leq i \leq \ell - 1$, $start(B_i)$ $[end(B_i)]$ is the vertex to which the edge in $\pi_{u,v}$ between $graph(B_{i-1})$ and $graph(B_i)$ [between $graph(B_i)$ and $graph(B_{i+1})$] is incident to. For sake of completeness, define $start(B_0) = u$, $end(B_0) = x_0$, $start(B_\ell) = y_{\ell-1}$, and $end(B_\ell) = v$. Denote by B_λ the least common ancestor of B_u and B_v in $\mathcal{T}_1$ (see, for example, Fig. 5).

The following lemma explains the updates needed in our data structure.

LEMMA 4.1. *Let π_i be the path in $\mathcal{T}_2(B_i)$ between $C(start(B_i))$ and $C(end(B_i))$, $0 \leq i \leq \ell$. After the insertion of edge (u, v) in G, we have the following changes*:

(i) *All and only the vertices in $\cup_{0 \leq i \leq \ell} \{graph(B_i)\}$ become 2-edge-connected in G*;

(ii) *Fix i, $0 \leq i \leq \ell$. Let $u_1^{(i)}, u_2^{(i)}, \ldots, u_{q_i}^{(i)}$ be the square nodes in π_i. All and*

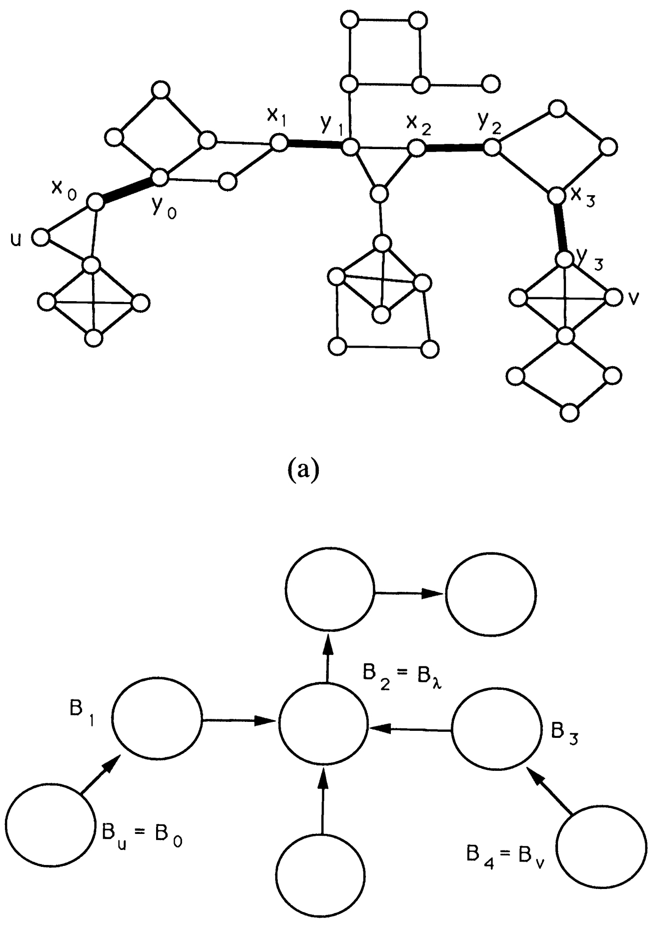


FIG. 5. (a) *A graph $G = (V, E)$ bridges between vertices u and v are in bold*; (b) *the tree of the 2-edge-connected components of G.*

only the vertices in the 3-edge-connected components of $graph(B_i)$ corresponding to $u_1^{(i)}$, $u_2^{(i)}, \ldots, u_{q_i}^{(i)}$ become 3-edge-connected in G.

Proof. Edge (u, v) introduces cycles in G that contain at least one vertex in $graph(B_0)$, $graph(B_1), \ldots, graph(B_\ell)$. Since these are the only new cycles introduced, all and only the vertices in those 2-edge-connected components become 2-edge-connected as stated in proposition (i).

The proof of proposition (ii) can be carried out along the same lines as the proof of Lemma 3.4. Let $V_{i,1}, V_{i,2}, \ldots, V_{i,p}$ be the 3-edge-connected components of $graph(B_i)$ before the insertion of edge (u, v). Let $V'_{i,1}, V'_{i,2}, \ldots, V'_{i,q}$, $1 \leq q \leq p$ be the 3-edge-connected of $graph(B_i)$ corresponding to the square nodes of $\mathcal{T}_2(B_i)$ in π_i, as they are met while going from $C(start(B_i))$ to $C(end(B_i))$ in $\mathcal{T}_2(B_i)$. Notice that $C(start(B_i)) = V'_{i,1}$ and $C(end(B_i)) = V'_{i,q}$. The insertion of edge (u, v) causes a new path outside $graph(B_i)$ between $V'_{i,1}$ and $V'_{i,q}$. By repeating the same argument given in the proof of Lemma 3.4, it can be shown that all and only the vertices of $graph(B_i)$ contained in $V'_{i,1}, V'_{i,2}, \ldots, V'_{i,q}$ become 3-edge-connected. □

Let $G^{new} = G \cup \{(u,v)\}$, and let $(G^{new})'$ be the graph obtained by shrinking the 3-edge-connected components of G^{new} into one vertex. Once again, Lemma 4.1 gives a way to compute $(G^{new})'$ from G'. Consider the 2-edge-connected components of G' obtained by shrinking the 3-edge-connected components of $graph(B_i)$, $0 \le i \le \ell$. Let us refer to them as B'_i, $0 \le i \le \ell$. For each one of them we can repeat the same argument given after Lemma 3.4. Fix i, $0 \le i \le \ell$. By definition of G' all the paths between $V'_{i,1}$ and $V'_{i,q}$ in G' are given by $q-1$ edge-disjoint simple cycles $\rho_{i,1}, \rho_{i,2}, \ldots, \rho_{i,q-1}$ in B'_i such that $V'_{i,1} \in \rho_{i,1}$, $V'_{i,q} \in \rho_{i,q-1}$, and $\rho_{i,j-1} \cap \rho_{i,j} = \{V'_{i,j}\}$, $2 \le j \le q-1$. Each $\rho_{i,j}$ is composed of two disjoint paths between $V'_{i,j}$ and $V'_{i,j+1}$, say, $\rho^{(1)}_{i,j}$ and $\rho^{(2)}_{i,j}$. Furthermore, the insertion of (u,v) in G implies that all and only the vertices in $V'_{i,s}$, $1 \le s \le p$, become 3-edge-connected in G. Therefore, in $(G^{new})'$ the vertices $V'_{i,1}, V'_{i,2}, \ldots, V'_{i,q}$ of G' have to be shrunk into one vertex V'_i. This destroys the previous simple cycles $\rho_{i,j}$, $1 \le j \le q-1$, and creates at most $2(q-1)$ new simple cycles $\rho^{(1)}_{i,j}$ and $\rho^{(2)}_{i,j}$, $1 \le j \le q-1$, that are all incident to V'_i. Furthermore, there is a new simple cycle in G' consisting of all the newly created vertices $V'_1, V'_2, \ldots, V'_\ell$ and edges $(x_0,y_0), (x_1,y_1), \ldots, (x_{\ell-1}, y_{\ell-1}), (u,v)$.

Because of Lemma 4.1 we have to perform the following updates in our data structure. First, all the nodes B_i, $0 \le i \le \ell$, have to be merged into one node B in $\mathcal{T}_1$ because all the vertices in $graph(B_0), graph(B_1), \ldots, graph(B_\ell)$ become now 2-edge-connected. This can be done by using the algorithm of Westbrook and Tarjan [36]. Second, we have to compute the tree $\mathcal{T}_2(B)$ of 3-edge-connected components of the new node $B = B_0 \cup B_1 \cup \ldots \cup B_\ell$. We compute $\mathcal{T}_2(B)$ starting from the trees $\mathcal{T}_2(B_0), \mathcal{T}_2(B_1), \ldots, \mathcal{T}_2(B_\ell)$ as follows. For $0 \le i \le \ell$, we perform *CompressPath*$(start(B_i), end(B_i), \mathcal{T}_2(B_i))$. This is correct because of condition (ii) of Lemma 4.1, and produces the new trees $\mathcal{T}_2^{new}(B_i)$, $0 \le i \le \ell$. Denote by σ_i the square node of $\mathcal{T}_2^{new}(B_i)$ which is the result of the path compression that took place in $\mathcal{T}_2(B_i)$. Notice that σ_i, $0 \le i \le \ell$, corresponds to the new vertex V'_i in $(G^{new})'$. We recall that B_λ is the least common ancestor of B_u and B_v in $\mathcal{T}_1$. As said before, all the new vertices V'_i are now in a new simple cycle of $(G^{new})'$: we create a new round node ρ, make ρ a child of σ_λ, and make $\sigma_{\lambda+1}, \ldots, \sigma_\ell, \sigma_0, \ldots, \sigma_{\lambda-1}$ children of ρ in $\mathcal{T}_2(B)$ in this order. This preserves the same order as in the new simple cycle created in $(G^{new})'$. Since σ_i, $i \ne \lambda$, is the root of $\mathcal{T}_2^{new}(B_i)$, we do not need to reroot any of these trees.

An *AddVertex*(u,v) operation inserts a new vertex u and connects it to v. We update our data structure as follows. Let B_v be the 2-edge-connected component containing v. We create a new 2-edge-connected component B_u containing only vertex u and make it child of B_v in $\mathcal{T}_1$. We initialize $\mathcal{T}_2(B_u)$ to be a tree with a singleton square root node, and initialize a new 3-edge-connected component containing only vertex u.

THEOREM 4.2. *There exists a data structure that supports any sequence of q Same3EdgeBlock and InsertEdge and n AddVertex operations in a total of $O((q+n)\alpha(q,n))$ time. The space required is $O(n)$.*

Proof. We first analyze the space complexity of our data structure. Notice that we start with a graph containing a single vertex. At the end of the sequence of operations, we end up with a graph G with at most $n+1$ vertices. Since G can have at most $O(n)$ 2-edge-connected components and bridges, $\mathcal{T}_1$ has size $O(n)$. For each 2-edge-connected component B_i of G, denote by n_i the number of vertices in B_i. Since each vertex of G is an at most one 2-edge-connected component, we have that $\sum_i n_i = n+1$. By Theorem 3.6, the tree of 3-edge-connected components of B_i has at most $O(n_i)$ nodes and edges.

As a result, all the trees in our data structure require

$$O\left(\sum_i n_i\right) = O(n)$$

space.

Each *Same3EdgeBlock* operation can be performed in $O(\alpha(q,n))$ amortized time as the following argument shows. Finding B_u and B_v, the 2-edge-connected components of G containing u and v, can be done in $O(\alpha(q,n))$ amortized time [36]. If $B_u \neq B_v$, then *Same3EdgeBlock* returns false and no further computation is required. If $B_u = B_v = B$, performing *Same3EdgeBlock* on $\mathcal{T}_2(B)$ requires $O(\alpha(q,n))$ amortized time by Theorem 3.6.

As for an *InsertEdge*(u,v) operation, the bridge-block tree can be updated in $O(\alpha(q,n))$ amortized time [36]. The update of the trees of 3-edge-connected components $\mathcal{T}_2(B_i)$, $0 \le i \le \ell$, requires

$$O\left(\ell + \sum_i |\pi_i|\right)$$

merge, *split*, and *parent* primitives. By applying exactly the same argument used in Theorem 3.6, we obtain that the total number of such primitives during any sequence of *InsertEdge* operations is $O(n)$. As a result, the total time required to perform any sequence of *InsertEdge* operations is $O((q+n)\alpha(q,n))$. Finally, each *AddVertex* operation can be performed in $O(1)$ time. □

Our data structure can be used even if we start with a nonempty graph $G_0 = (V_0, E_0)$ and allow $O(|V_0| + |E_0|)$ preprocessing.

LEMMA 4.3. *Given a graph $G_0 = (V_0, E_0)$ our data structure can be initialized in $O(|V_0| + |E_0|)$ time.*

Proof. Compute in $O(|V_0| + |E_0|)$ the 2-edge-connected components of G_0 by using the algorithm of Tarjan [32]. This gives also $\mathcal{T}_1$, the bridge-block tree of G_0, which can be initialized in $O(|\mathcal{T}_1|) = O(|V_0|)$ as a condensible and expandable nodes tree. For each 2-edge-connected component B of G_0, compute the 3-edge-connected components of B and initialize the tree of 3-edge-connected components $\mathcal{T}_2(B)$. This requires $O(|B|)$ time[16], and sums up to a total time of $O(|V_0| + |E_0|)$. □

5. Concluding remarks. In this paper we have studied the on-line maintenance of the 3-edge-connected components of an undirected graph during edge and vertex insertions. We have proposed algorithms that support any sequence of q *InsertEdge*, *AddVertex*, and *Same3EdgeBlock* operations on an initially connected graph with n vertices in a total of $O((n+q)\alpha(q,n))$ time. Recently La Poutré [23] extended this bound to unconnected graphs.

We remark that the same $O(\alpha(q,n))$ amortized bound holds also if we wish to maintain the names of the 3-edge-connected components of a graph subject to the following operations.

3EdgeBlock(x): Return the name of the 3-edge-connected component containing vertex x.

InsertEdge(x,y,A): Insert a new edge between vertices x and y and call A the new (if any) 3-edge-connected component created by (x,y).

We have been able to achieve an $O(m^{2/3})$ bound per operation for the fully dynamic maintenance of the 3-edge-connected components of a graph [13] by using techniques similar to the ones used for the fully dynamic maintenance of the 2-edge-connected components [14].

One might ask whether there is any efficient partially dynamic algorithm for maintaining on-line the k-edge-connected components of a graph, $k \geq 4$.

Acknowledgment. We are grateful to Dany Breslauer for many useful discussions.

REFERENCES

[1] G. AUSIELLO, G. F. ITALIANO, A. MARCHETTI-SPACCAMELA, AND U. NANNI, *Incremental algorithms for minimal length paths*, J. Algorithms, 12 (1991), pp. 615–638.

[2] G. AUSIELLO, A. MARCHETTI-SPACCAMELA, AND U. NANNI, *Dynamic maintenance of paths and path expressions in graphs*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, Lecture Notes in Computer Science 358, Springer-Verlag, Berlin, 1989, pp. 1–12.

[3] F. CHIN AND D. HOUK, *Algorithms for updating minimum spanning trees*, J. Comput. System Sci., 16 (1978), pp. 333–344.

[4] R. F. COHEN AND R. TAMASSIA, *Dynamic expression trees and their applications*, in Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991, pp. 52–61.

[5] G. DIBATTISTA AND R. TAMASSIA, *Incremental planarity testing*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 436–441.

[6] ———, *On-line graph algorithms with* SPQR-*trees*, in Proceedings of the 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 443, Springer-Verlag, Berlin, 1990, pp. 598–611.

[7] E. A. DINIC, A. V. KARZANOV, AND M. V. LOMONOSOV, *On the structure of the system of minimal edge cuts in a graph*, in Studies in Discrete Optimization, A. A. Fridman, ed., Nauka, Moskow, 1976, pp. 290–306. (In Russian.)

[8] D. EPPSTEIN, G. F. ITALIANO, R. TAMASSIA, R. E. TARJAN, J. WESTBROOK, AND M. YUNG, *Maintenance of a minimum spanning forest in a dynamic planar graph*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 1–11; J. Algorithms, to appear.

[9] S. EVEN AND H. GAZIT, *Updating distances in dynamic graphs*, Methods Oper. Res., 49 (1985), pp. 371–387.

[10] S. EVEN AND Y. SHILOACH, *An on-line edge deletion problem*, J. Assoc. Comput. Mach., 28 (1981), pp. 1–4.

[11] G. N. FREDERICKSON, *Data structures for on-line updating of minimum spanning trees*, SIAM J. Comput., 14 (1985), pp. 781–798.

[12] H. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci., 30 (1985), pp. 209–221.

[13] Z. GALIL AND G. F. ITALIANO, *Fully dynamic algorithms for* 3-*edge-connectivity*, in preparation.

[14] ———, *Fully dynamic algorithms for edge connectivity problems*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991, pp. 317–327.

[15] ———, *Maintaining biconnected components of dynamic planar graphs*, in Proceedings of the 18th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 510, Springer-Verlag, Berlin, 1991, pp. 339–350.

[16] ———, *Reducing edge connectivity to vertex connectivity*, Sigact News, 22 (1991), pp. 57–61.

[17] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[18] J. HOPCROFT AND R. E. TARJAN, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.

[19] T. IBARAKI AND N. KATOH, *On-line computation of transitive closure for graphs*, Inform. Process. Lett., 16 (1983), pp. 95–97.

[20] G. F. ITALIANO, *Amortized efficiency of a path retrieval data structure*, Theoret. Comput. Sci., 48 (1986), pp. 273–281.

[21] ———, *Finding paths and deleting edges in directed acyclic graphs*, Inform. Process. Lett., 28 (1988), pp. 5–11.

[22] A. V. KARZANOV AND E. A. TIMOFEEV, *Efficient algorithm for finding all minimal edge cuts of a nonoriented graph*, Cybernetics, (1986), pp. 156–162. Translated from Kybernetika, 2(1968), pp. 8–12.

[23] J. A. LA POUTRÉ, *personal communication*, 1992.

[24] J. A. LA POUTRÉ AND J. VAN LEEUWEN, *Maintenance of transitive closure and transitive reduction of graphs*, in Proceedings of the Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 314, Springer-Verlag, Berlin, 1988, pp. 106–120.

[25] C. C. LIN AND R. C. CHANG, *On the dynamic shortest path problem*, in Proceedings of the International Workshop on Discrete Algorithms and Complexity, 1989, pp. 203–212.

[26] J. H. REIF, *A topological approach to dynamic graph connectivity*, Inform. Process. Lett., 25 (1987), pp. 65–70.

[27] H. ROHNERT, *A dynamization of the all pairs least cost path problem*, in Proceedings of the 2nd Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 182, Springer-Verlag, Berlin, 1985, pp. 279–286.

[28] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comput. System Sci., 24 (1983), pp. 362–381.

[29] P. M. SPIRA AND A. PAN, *On finding and updating spanning trees and shortest paths*, SIAM J. Comput., 4 (1975), pp. 375–380.

[30] R. TAMASSIA, *A dynamic data structure for planar graph embedding*, in Proceedings of the 15th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 317, Springer-Verlag, Berlin, 1988, pp. 576–590.

[31] R. TAMASSIA AND F. P. PREPARATA, *Dynamic maintenance of planar digraphs, with applications*, Algorithmica, 5 (1990), pp. 509–527.

[32] R. E. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[33] R. E. TARJAN AND J. VAN LEEUWEN, *Worst-case analysis of set union algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 245–281.

[34] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–864.

[35] J. WESTBROOK, *Algorithms and data structures for dynamic graph problems*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, New Jersey, October 1989, Tech. Report CS-TR-229-89.

[36] J. WESTBROOK AND R. E. TARJAN, *Maintaining bridge-connected and biconnected components on-line*, Tech. Report CS-TR-228-89, Department of Computer Science, Princeton University, Princeton, New Jersey, August 1989; Algorithmica, to appear.

[37] D. M. YELLIN, *A dynamic transitive closure algorithm*, Tech. Report 13535, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 1988.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 29–45, February 1993

ON THE BOUNDEDNESS OF CONSTANT-TIME-MAINTAINABLE DATABASE SCHEMES*

HÉCTOR J. HERNÁNDEZ† AND KE WANG‡

Abstract. Constant-time-maintainable database schemes are highly desirable with respect to constraint enforcement, since it is possible to determine whether any of their consistent states plus an inserted tuple is consistent in time independent of the state size. Several proper subclasses of constant-time-maintainable database schemes are known to be bounded with respect to dependencies and hence very desirable with respect to query answering. However, whether the whole class of constant-time-maintainable database schemes is bounded is not known for sure.

In this paper, it is proven that the entire class of constant-time-maintainable database schemes is bounded with respect to dependencies and thus very desirable with respect to query answering in the following cases: (1) only cover-embedded functional dependencies appear as constraints; (2) only equality-generating dependencies appear as constraints and the database scheme has a lossless join. In particular, it is shown that total projections of representative instances can be computed via unions of projections of simple chase join expressions. Since it is known how to optimize these expressions, it is possible to compute total projections optimally. These results show that the class of constant-time-maintainable database schemes is the largest class of database schemes, which are highly desirable with respect to both constraint enforcement and query answering. This class of schemes can be effectively recognized by known algorithms. The previously known largest class of database schemes with these desirable properties is the class of independent database schemes, which is a proper subclass of constant-time-maintainable schemes.

Key words. database, dependencies, query processing, constraint enforcement, representative instance, boundedness

AMS(MOS) subject classifications. 68P15, 68Q

1. Introduction.

Within the *weak instance model* [H2], [Vas], the *maintenance problem* [GY], [GW], [W] of a database scheme is the following decision problem: Given a consistent state of the database scheme and a tuple to be inserted into the state, is the modified state consistent with respect to the constraints imposed on the database scheme? Database schemes for which this problem has "very fast" solutions are highly desirable in practice.

The notion of constant-time-maintainability was proposed by Graham and Wang [GW] to capture the intuition on "very fast" solutions to the maintenance problem. Informally, a database scheme is constant-time-maintainable with respect to the constraints imposed on the scheme if its maintenance problem can be solved in time independent of the state size. Therefore constant-time-maintainable database schemes are particularly desirable in a large and highly dynamic database environment. Also in such an environment, we will consider only constant-time-maintainable schemes to have fast solutions to constraint enforcement, because all other schemes are linear time lower bounded [GW], [WG]. As mentioned in [GW], constant-time-maintainability generalized the notion of independence [GY], [S1], [S2], and the class of constant-time-maintainable schemes properly contains the class of independent schemes when (1) only functional dependencies appear, (2) functional dependencies and the join dependency

*Received by the editors May 15, 1990; accepted for publication June 5, 1991.

†Texas A&M University, College Station, Texas 77843–3112. Present address, Department of Computer Science, New Mexico State University, Las Cruces, New Mexico 88003–0001.

‡Department of Computer Science, Chonqing University, Chonqing, Sichuan 63004, People's Republic of China. Present address, Department of Information Systems and Computer Science, National University of Singapore, 10 Kent Ridge Crescent, Singapore 0511. The work of this author was supported by the Nature Science Foundation of China for Young Scientists.

of the database scheme appear, and (3) functional dependencies and inclusion dependencies appear [AC2].

Recently, the recognition problem of constant-time-maintainable database schemes was solved in several useful cases. An exponential time recognition algorithm for constant-time-maintainable database schemes was given in [GW], [W], [WG] for each of the following cases: (1) The database scheme cover embeds functional dependencies; (2) functional dependencies plus the join dependency of the database scheme appear as constraints; and (3) equality-generating dependencies appear as constraints and the database scheme has a lossless join. An efficient recognition algorithm for constant-time-maintainable database schemes was presented in [HC] when cover-embedding Boyce–Codd Normal Form is assumed. Previous work on fast constraint enforcement can be found in [BrV], [CH1], [CH2], [CH4].

In view of the importance and generality of constant-time-maintainable database schemes, this paper investigates query processing in that class of database schemes. Under the weak instance model, the *X-total projection of the representative instance* [M], [S1], [S2], [Y] of a database can be used to answer a query defined on a set of attributes X; intuitively, the representative instance of a database is an adequate and correct representation of all the information that can be logically inferred from the database using certain rules derived from the dependencies that the database must satisfy; the X-total projection of the representative instance is the set of tuples in the representative instance that do not contain missing information on X. Under this approach, it is highly desirable for query processing to have a database scheme that would allow us to compute the X-total projection of the representative instance via a predetermined relational expression that is independent of the databases. This is possible exactly when the database scheme is bounded with respect to the dependencies given [GM], [MUV].

Unfortunately, the problem of deciding whether a database scheme is bounded with respect to dependencies is conjectured to be undecidable even for the case of functional dependencies [MUV]. It has been shown that proving boundedness of database schemes with respect to dependencies is difficult even for restricted cases [C], [CH1], [CH2], [IIK], [HC], [MRW], [S2], [S3]. Therefore, defining some classes of database schemes that are general enough and obviously bounded, like the one in [CM1], or proving the boundedness of some meaningful classes of database schemes seems to be a reasonable thing to do. The two largest classes of database schemes that can be effectively recognized and have an effective construction of relational expressions for computing total projections with respect to functional dependencies are the class of independent schemes [C], [IIK], [S3] and the class of independence-reducible database schemes [CH2]. A general, sufficient condition for unboundedness of database schemes when functional dependencies are considered was presented in [CH3]. A methodology for incrementally generating bounded database schemes can be found in [CH4]. When only total projections on the universe of attributes are considered, Sagiv [S4] gave a necessary and sufficient condition for computing total projections for lossless database schemes with only tuple-generating dependencies.

The notion of boundedness has also been investigated in the context of optimization of Datalog (the language of function-free Horn-clause) programs. A Datalog program is *bounded* if it is possible to eliminate recursion from it [CGKV], [GMSV], [I], [NS]. Boundedness of Datalog programs has been shown to be undecidable in [GMSV]. Some other decidable and undecidable results on boundedness of Datalog programs are presented in [CGKV], [S4], [Var].

The class of constant-time-maintainable database schemes seems to be a subclass of bounded database schemes. Several subclasses of constant-time-maintainable database schemes are known to be bounded: Boyce–Codd Normal Form (BCNF) independent database schemes [S2], independent database schemes [C], [IIK], [MRW], [S3], γ-acyclic BCNF database schemes [CH1], split-free independence-reducible schemes [CH2], and BCNF constant-time-maintainable database schemes [HC]. However, whether the whole class of constant-time-maintainable database schemes is bounded with respect to dependencies is not known.

In this paper we prove in §4 that the entire class of constant-time-maintainable database schemes is bounded when only cover-embedded functional dependencies appear as constraints. In §5 we describe a method to construct the relational expressions that compute total projections of representative instances. In particular, we prove that unions of projections of simple chase join expressions [AC3], [C], a kind of extension joins [H1], compute the total projections of the representative instances. The construction, however, takes exponential time in the size of the database scheme. Since it is known from [AC3], [C] how to optimize these expressions to a minimal number of union and join operations, we can compute total projections optimally (in that sense). Then in §6 we show that the boundedness and construction of relational expressions obtained in §§4 and 5 also hold when only equality-generating dependencies appear as constraints and the database scheme has a lossless join. In §7 we prove that we can compute X-total projections via unions of projections of simple chase join expressions, for any X included in any relation scheme in the database when a set of functional dependencies and the join dependency of the database scheme are the constraints. In §8 we give our conclusions.

2. Definitions and notation. In this section, we give most of the notation required for the rest of this paper.

2.1. Basic definitions. We shall follow standard notation [Ma], [U] and only give some nonstandard definitions here.

We fix a finite set U to be the *universe* of attributes (or columns) and fix $\mathbf{R}$, the *database scheme*, to be a collection of relation schemes $\{R_1, \ldots, R_n\}$ whose union is U. A *database state* (or *state*) $\mathbf{r}$ on $\mathbf{R}$ is an assignment of finite relations to relation schemes of $\mathbf{R}$; we shall denote it as $\mathbf{r} =< r_1 = \mathbf{r}(R_1), \ldots, r_n = \mathbf{r}(R_n) >$. A *tableau* is a set of tuples defined on U [ASU]. For each attribute $A_j \in U$, the domain of a tableau on A_j consists of countable many variables, and constants taken from $dom(A_j)$, the *domain* of A_j. We assume that all the tableaux and states are *typed*, that is, the domains of a tableau or state on different attributes are disjoint. A *symbol* is either a constant or a variable, and we say that a symbol is *unique* if it is distinct from any other symbol appearing anywhere else. We say that a tuple $\mu[X]$, the restriction of tuple μ onto attributes X, is *total* if $\mu[A_i]$ is a constant for all $A_i \in X$, where $X \subseteq U$.

Assume T_1 and T_2 are tableaux. A *valuation function* $\theta : T_1 \rightarrow T_2$ is a function from symbols in T_1 to symbols in T_2, which is the identity on constants. A *containment mapping* $\theta : T_1 \rightarrow T_2$ is a valuation function such that $t \in T_1$, implies $\theta(t) \in T_2$.

Let $\mathbf{r} =< r_1 = \mathbf{r}(R_1), \ldots, r_n = \mathbf{r}(R_n) >$ be a state on $\mathbf{R}$. We define $T_\mathbf{r}$, *the tableau for state* $\mathbf{r}$, as follows: For each relation $r_i \in \mathbf{r}$ and for each tuple $t \in r_i$, there is a tuple s in $T_\mathbf{r}$ corresponding to it; the tuple s is defined as follows: $s[R_i] = t$, and for all $A_j \in U - R_i$, $s[A_j]$ is a unique variable in $T_\mathbf{r}$. Now, let r'_j be a relation on $R_j \in \mathbf{R}$. Then $\mathbf{r} \cup r'_j$ shall denote the state $< r_1, \ldots, r_{j-1}, r_j \cup r'_j, r_{j+1}, \ldots, r_n >$.

2.2. Dependencies and chasing. The kinds of constraints considered here are (*typed*) *equality-generating dependencies* (*egd's*), *functional dependencies* (*fd's*), and the *join dependency* (*jd*) $\bowtie \mathbf{R}$ [ABU], [BV], [F], [YP]. We shall use the term *dependencies* to refer to the above-mentioned dependencies.

Associated with each dependency there is a *dependency rule* [ABU], [BV], [F], [YP]. Given a tableau T and a set of dependencies D, we can apply their associated rules to T to infer additional information. These dependency rules are defined in [ABU], [BV], [MMS]. $CHASE_D(T)$ denotes the final tableau obtained from applying exhaustively to T the rules for the dependencies in D; $CHASE_D(T)$ is also known as the *chase of* T (with respect to D) [MMS].

Without loss of generality, we will assume in this paper that every fd has a single attribute as its right-hand side. We define the *closure* of a set of fd's F, denoted as F^+, to be the set of fd's that logically follow from F [Ma], [U]. An fd $X \rightarrow A$ is *embedded* in a relation scheme R_i if $XA \subseteq R_i$. A set of fd's F is *embedded* in $\mathbf{R}$ if each fd $X \rightarrow A \in F$ is embedded in some $R_i \in \mathbf{R}$. For a set of fd's F and a relation scheme $R_i \in \mathbf{R}$, $F|R_i$ denotes all fd's from F that are embedded in R_i. $\mathbf{R}$ is said to be *cover embedding* (or to *cover embed*) a set of fd's F if there exists a *cover* G of F, that is, $G^+ = F^+$, such that G is embedded in $\mathbf{R}$; G is said to be an *embedded cover* of F. $\mathbf{R}$ is said to *preserve* a set of fd's F if for any relation I defined on U, I satisfies F implies $\bowtie \pi_{\mathbf{R}}(I)$ satisfies F, where $\pi_{\mathbf{R}}(I)$ denotes the state $< \pi_{R_1}(I), \ldots, \pi_{R_n}(I) >$ and $\bowtie \pi_R(I) = \pi_{R_1}(I) \bowtie \cdots \bowtie \pi_{R_n}(I)$. $\mathbf{R}$ is said to have a *lossless join* with respect to a set of dependencies D if the jd $\bowtie \mathbf{R}$ logically follows from D [ABU].

2.3. Simple chase join expressions and derivation sequences. Borrowing from [C], in this subsection we define derivation sequences and simple chase join expressions, a generalization of extension joins [H1].

Given a set of fd's F, a *derivation sequence* (*ds*) of some relation scheme R_i (with respect to F) is a finite sequence of fd's $< Y_1 \rightarrow A_1, \ldots, Y_m \rightarrow A_m >, m \geq 0$, that satisfies the following conditions: For all $1 \leq j \leq m$,

- $Y_j \rightarrow A_j \in F$;
- $Y_j \subseteq R_i A_1 \cdots A_{j-1}$, and $A_j \notin R_i A_1 \cdots A_{j-1}$.

The ds is said to *cover* X if $R_i A_1 \cdots A_m \supseteq X$. A ds $< Y_1 \rightarrow A_1, \ldots, Y_m \rightarrow A_m >$ that covers X is said to be *nonredundant* if for every $Y_j \rightarrow A_j$ with $1 \leq j \leq m$, either $A_j \in X$ or $A_j \in Y_k$, for some $k > j$ (otherwise, any fd that fails to satisfy this condition is said to be *redundant*). Two ds's of R_i are said to be *equivalent* if they are identical up to permutation of the fd's in the sequences. Given a ds $< Y_1 \rightarrow A_1, \ldots, Y_m \rightarrow A_m >$ of R_i that covers X, if each fd $Y_j \rightarrow A_j$ is embedded in some relation scheme R_j, then we define the *simple chase join expression* (*simple cje*) E *for the ds* as $E = \pi_X(R_i \bowtie \pi_{Y_1 A_1}(R_1) \bowtie \cdots \bowtie \pi_{Y_m A_m}(R_m))$. Given a simple cje E and a state $\mathbf{r}$ on $\mathbf{R}$, $E(\mathbf{r})$ denotes the evaluation of E after substituting every relation scheme R_j in E with the relation $r_j \in \mathbf{r}$. It should be obvious that for every ds π_1 of R_i covering X, there is a nonredundant ds π_2 of R_i covering X (obtained by removing redundant fd's from π_1) such that the simple cje E_1 for π_1 and the simple cje E_2 for π_2 satisfy $E_2(\mathbf{r}) \supseteq E_1(\mathbf{r})$ for every state $\mathbf{r}$.

2.4. Weak instance and boundedness. Let $\mathbf{r}$ be a state on $\mathbf{R}$, let I be a relation defined on U, and let D be a set of dependencies. Then I is a *weak instance* for $\mathbf{r}$ with respect to D if $\pi_{R_i}(I) \supseteq r_i$ for each $\mathbf{R}_i \in \mathbf{R}$ and I satisfies D. $\mathbf{r}$ is said to be *consistent* with respect to D if a weak instance exists for the state with respect to D [GMV], [H2], [Vas]. The set of all consistent states for $\mathbf{R}$ with respect to D is denoted by $CONS(\mathbf{R}, D)$. $CHASE_D(T_{\mathbf{r}})$ is called *the representative instance for state* $\mathbf{r}$ (with respect to D) [M], [S1],

[S2], [Y]. The *X-total projection* of the representative instance (with respect to D) for $\mathbf{r}$, denoted $[X]_{\mathbf{r}}^{D}$, is $\{t[X] | t \in CHASE_D(T_{\mathbf{r}})$ and $t[X]$ is total$\}$.

A database scheme $\mathbf{R}$ is *bounded* with respect to a set of dependencies D if for every $X \subseteq U$, every X-total tuple in the representative instance of any consistent state $\mathbf{r}$ of $\mathbf{R}$ with respect to D can be obtained in at most k applications of rules for dependencies in D to tableau $T_{\mathbf{r}}$ for some constant $k \geq 0$ [GM], [MUV]. It has been shown in [GM], [MUV] that a database scheme is bounded with respect to D if and only if for any $X \subseteq U$, the X-total projection of the representative instance for any consistent state can be computed by a predetermined relational expression that is independent of the state.

2.5. Constant-time-maintainable and independent schemes. The *maintenance problem* (for database states) of $\mathbf{R}$ with respect to a set of dependencies D is the following decision problem: Let $\mathbf{r}$ be a consistent state of a database scheme $\mathbf{R}$ with respect to D and assume we insert a tuple t into $r_p \in \mathbf{r}$. Is $\mathbf{r} \cup \{t\}$ a consistent state of $\mathbf{R}$ with respect to D? We say that $< \mathbf{r}, t >$ is a *yes-instance* of the maintenance problem of $\mathbf{R}$ with respect to D if $\mathbf{r} \cup \{t\}$ is consistent with respect to D; otherwise, $< \mathbf{r}, t >$ is a *no-instance* of the maintenance problem of $\mathbf{R}$ with respect to D. An algorithm *solves* the maintenance problem of $\mathbf{R}$ with respect to F if for every instance $< \mathbf{r}, t >$, the algorithm returns a *yes* answer exactly when $< \mathbf{r}, t >$ is a yes-instance of the maintenance problem of $\mathbf{R}$ with respect to D. We call such an algorithm a *maintenance algorithm*.

Following [GW], we define constant-time-maintainable database schemes as follows. Suppose there is a maintenance algorithm $\mathbf{A}$ that solves the maintenance problem of $\mathbf{R}$ with respect to a set of dependencies D. Let $< \mathbf{r}, t >$ be an instance of the maintenance problem of $\mathbf{R}$ with respect to D. Assume that $\mathbf{r}$ is stored on a device that responds to requests of the form $< R_i, \Psi >$, where $R_i \in \mathbf{R}$ and Ψ is a Boolean combination of formulas of the form $A =$ 'a,' where $A \in R_i$, a is an element of the domain of A that appears in either the inserted tuple t or the tuples previously returned, as defined below. The device responds to the request $< R_i, \Psi >$ by returning, if it exists, an arbitrary tuple from $r_i \in \mathbf{r}$ that satisfies Ψ. Now, we define $\#\mathbf{A}(\mathbf{r}, t)$ to be the number of requests of the above form made by $\mathbf{A}$ on $< \mathbf{r}, t >$. We say that $\mathbf{A}$ solves the maintenance problem of $\mathbf{R}$ with respect to D in *constant time* if there is a constant integer $k \geq 0$ such that $k \geq \#\mathbf{A}$ $(\mathbf{r}, t)$ for all instances $< \mathbf{r}, t >$ of the maintenance problem of $\mathbf{R}$ with respect to D. A database scheme $\mathbf{R}$ is said to be *constant-time-maintainable* (*ctm*) with respect to D if there is a maintenance algorithm that solves the maintenance problem of $\mathbf{R}$ with respect to D in constant time [GW]. The reader should note that the definition suggests that constant-time-maintainability is cover insensitive; that is, for any two equivalent sets of dependencies, say, D_1 and D_2, $\mathbf{R}$ is ctm with respect to D_1 if and only if $\mathbf{R}$ is ctm with respect to D_2.

A database scheme is *independent* with respect to a set of dependencies D if each relation in a state satisfies its projected dependencies implies that the state is consistent with respect to D [GY], [IIK], [S2].

The following example illustrates these definitions.

Example 1. Let $\mathbf{R} = \{CAZ, CZ\}$ and $F = \{CA \to Z, Z \to C\}$. This is the classic City, Address, Zip database scheme. $\mathbf{R}$ is not independent with respect to F because $Z \to C$ is embedded in both CAZ and CZ. We now show that $\mathbf{R}$ is ctm with respect to F. Suppose that we have a consistent state $\mathbf{r}$ with respect to F and that we insert a tuple t into either the CZ relation or the CAZ relation. We now prove that we need to issue at most three requests of the above form to verify whether $\mathbf{r} \cup \{t\} \in CONS(\mathbf{R}, F)$.

Case 1. Assume $t =< c, z >$ is to be inserted into the CZ relation. To verify whether $Z \to C$ is satisfied by the updated state, we need to issue the requests $< CZ, Z =$'z'$>$

and $< CAZ, Z =$'z'$>$ to retrieve from $\mathbf{r}$ any tuple of the form $< c_1, z >$ or $< c_2, a, z >$. Then the consistency of $\mathbf{r}$ implies that the updated state is consistent if and only if none of these tuples is returned (i.e., they do not exist in $\mathbf{r}$) or the returned tuples have the constant c on column C (i.e., if one tuple is returned, then $c_1 = c$ or $c_2 = c$; if two tuples are returned, then $c_1 = c$ and $c_2 = c$).

Case 2. Assume $t =< c, a, z >$ is to be inserted into the CAZ relation. To verify if $CA \rightarrow Z$ is satisfied by the updated state, we need to issue the request $< CAZ, C =$ 'c' $\wedge A =$'a'$>$ to retrieve from $\mathbf{r}$ any tuple of the form $< c, a, z' >$; if a tuple is returned with $z' \neq z$, the updated state is not consistent with respect to F. Otherwise, we still have to check for a possible violation of $Z \rightarrow C$. To verify this, we need to issue at most two requests, as it is shown in Case 1 above.

The above discussion shows that at most three requests are required to solve the maintenance problem of $\mathbf{R}$ with respect to F. Therefore $\mathbf{R}$ is ctm with respect to F. $\square$

3. An equivalent definition of ctm schemes. It is difficult to work directly with the notion of constant-time-maintainability as it was defined above, because the notion is concerned with the existence of some kind of algorithm. To alleviate that problem, an equivalent definition of constant-time-maintainability, which is based on a specific computation, was given in [W]. We now present this alternative definition.

Assume that F is a set of fd's embedded in $\mathbf{R}$ such that $F|R_i$ is a cover of $F^+|R_i$ for every $R_i \in \mathbf{R}$. Let $\mathbf{r}$ be a state on $\mathbf{R}$ and let u be a tuple on U. We say that the pair $\tau =< v_i, X \rightarrow A >$ can *expand* u if $X \rightarrow A \in F|R_i$, v_i is a tuple in $r_i \in \mathbf{r}$ for some $R_i \in \mathbf{R}$, and $u[X] = v_i[X]$. Sometimes, we just say that v_i (or $X \rightarrow A$) can expand u if we are not concerned with the other element. Assume that $\tau =< v_i, X \rightarrow A >$ can expand u. Then $\tau(u)$, the result of expanding u by τ, is the tuple defined on U as follows: $\tau(u)$ is u, except that $\tau(u)[A]$ is $v_i[A]$ if $u[A] \neq v_i[A]$. If $\tau(u) \neq u$, we say that τ can *strictly expand* u. Let $\chi =< \tau_1, \ldots, \tau_m >$, where for $1 \leq j \leq m$, $\tau_j =< p_j, X_j \rightarrow A_j >$ and $X_j \rightarrow A_j \in F|R_l$ for some $R_l \in \mathbf{R}$, and p_j is a tuple from r_l in $\mathbf{r}$. We say that χ can (*strictly*) *expand* u if τ_1 can (strictly) expand u, and for $2 \leq j \leq m$, τ_j can (strictly) expand $\tau_{j-1}(\tau_{j-2}(\cdots\tau_1(u)\cdots))$. We also say that the sequence of fd's $< X_1 \rightarrow A_1, \ldots, X_m \rightarrow A_m >$ can (strictly) expand u in the above case.

Given a state $\mathbf{r} \in CONS(\mathbf{R}, F)$ and a total tuple v on $V \subseteq U$, let $aug_U(v)$ be the tuple on U defined as follows: $aug_U(v)[V] = v$ and for each $B \in U - V$, $aug_U(v)[B]$ is a unique variable. The *expansion computation of* v *in* $\mathbf{r}$ (with respect to F) is defined as follows [W]:

```
Let $\bar{v}$ be $aug_U(v)$.
Repeatedly expand $\bar{v}$ by a tuple in r and an fd in $F$ until either
        (i)  no more changes can be made to $\bar{v}$ or
        (ii) some constant is replaced by a different constant.
```

It was proven in [W], [WG] that the expansion computation has finite Church–Rosser-like properties, provided that for every $R_i \in \mathbf{R}$, $F|R_i$ is a cover of $F^+|R_i$, which has been assumed at the beginning of this section. In particular, the computation stops after at most $|U|$ strict expansions, and the condition (i) or (ii) on which the computation stops and the final tuple $\bar{v}$ if it stops at (i) are unique for every input $\mathbf{r}$ and v. If the expansion computation of v in $\mathbf{r}$ stops at condition (i), v is said to be *expansible in* $\mathbf{r}$ (with respect to F), and the final universal tuple $\bar{v}$ is called *the expansion of* v *in* $\mathbf{r}$ (with respect to F); otherwise, v is said to be *not expansible in* $\mathbf{r}$ (with respect to F), and we say that the expansion computation of v in $\mathbf{r}$ *reveals a contradiction*. Obviously, for any state $\mathbf{r} \in CONS(\mathbf{R}, F)$, every tuple in $\mathbf{r}$ is expansible in $\mathbf{r}$.

The following theorem gives another equivalent definition of ctm schemes, which we shall use for the rest of this paper.

THEOREM 1. *Let* $\mathbf{R}$ *be a database scheme, and let* F *be a set of* fd*'s embedded in* $\mathbf{R}$ *such that* $F|R_i$ *is a cover of* $F^+|R_i$*, for every* $R_i \in \mathbf{R}$*. Then* $\mathbf{R}$ *is* ctm *with respect to* F *if and only if for every instance* $< \mathbf{r}, u >$ *of the maintenance problem of* $\mathbf{R}$ *with respect to* $F, \mathbf{r} \cup \{u\} \notin CONS(\mathbf{R}, F)$ *implies* u *is not expansible in* $\mathbf{r}$ *with respect to* F.

Proof. See the proof of Theorem 3.3 and Definition 3.1 in [W]. (For a full proof see Thm. 4.2.3 in [WG]. The assumption that for any $R_i \in \mathbf{R}$, $F|R_i$ is a cover of $F^+|R_i$ is essential in that proof.) □

4. Ctm schemes are bounded with respect to cover-embedded fd's. Throughout this section we assume that (1) F is a set of fd's embedded in $\mathbf{R}$; (2) $F|R_i$ is a cover of $F^+|R_i$ for every $R_i \in \mathbf{R}$; and (3) every fd in F has a single attribute on the right-hand side. Any set G of cover-embedded fd's can be transformed into a set satisfying (1) and (3) above in polynomial time [BH], [GY]. The transformation into a set satisfying (2) takes exponential time in general. For any such cover F of G, $CHASE_F(T_\mathbf{r})$ and $CHASE_G(T_\mathbf{r})$ are identical up to the renaming of variables [MMS], and therefore they have identical total projections. Thus our results in this section, about boundedness, and in §5, about the computation of total projections, are actually those for cover-embedding ctm schemes.

In this section, we shall prove that the constant-time-maintainability of $\mathbf{R}$ with respect to F implies the boundedness of $\mathbf{R}$ with respect to F. We prove this fact by showing how to chase any consistent state of a ctm database scheme in a particular way. In §5, we show how to construct the relational expressions that compute the total projections of representative instances.

4.1. The expansion of r. Let $\mathbf{r} \in CONS(\mathbf{R}, F)$. We define

$$T_\mathbf{r}^* = \{\bar{u} | \bar{u} \text{ is the expansion of } u \text{ in } \mathbf{r} \text{ (with respect to } F), u \in r_i, r_i \in \mathbf{r}\},$$

where we assume that all the variables in $T_\mathbf{r}^*$ are unique. We say that $T_\mathbf{r}^*$ is the *expansion of* $\mathbf{r}$ (with respect to F). Observe that $T_\mathbf{r}^*$ is a tableau in a chase of $T_\mathbf{r}$ with respect to F. We shall prove that if $\mathbf{R}$ is ctm with respect to F, then we can obtain $CHASE_F(T_\mathbf{r})$ from $T_\mathbf{r}^*$ without equating any variable in $T_\mathbf{r}^*$ to a constant from $\mathbf{r}$. This shall imply that for any $X \subseteq U$ and for any $\mathbf{r} \in CONS(\mathbf{R}, F)$, $[X]_\mathbf{r}^F = \{t[X] | t \in T_\mathbf{r}^* \text{ and } t[X] \text{ is total}\}$. Then the boundedness of $\mathbf{R}$ with respect to F follows because every X-total tuple in $T_\mathbf{r}^*$ can be obtained by at most $|U|$ strict expansions, each being an application of an fd-rule for some fd in F. We are going to prove this claim in §4.3 by induction on the number of applications of fd-rules to the tableau. The following subsection shall constitute the basis of such proof.

4.2. Basis of the proof of the main lemma. Assume the following for the rest of this subsection:

- $\mathbf{R}$ is ctm with respect to F, where F is as assumed above,
- $\mathbf{r} \in CONS(\mathbf{R}, F)$, and

- There are two tuples $\bar{u}$ and $\bar{v}$ in $T^*_{\mathbf{r}}$, and a nontrivial fd $X \to A \in F^+|R_1$ for some $R_1 \in \mathbf{R}$ such that $\bar{u}[X] = \bar{v}[X]$ and $\bar{u}[A] \neq \bar{v}[A]$. Furthermore, we assume that $\bar{u}$ and $\bar{v}$ come from tuples u and v in $\mathbf{r}$, respectively, that is, $\bar{u}$ is the expansion of u in $\mathbf{r}$ (with respect to F) and $\bar{v}$ is the expansion of v in $\mathbf{r}$ (with respect to F).

We shall prove that both $\bar{u}[A]$ and $\bar{v}[A]$ are variables.

Let $V = \{B | B \in R_1$ and $\bar{u}[B] = \bar{v}[B]\}$; notice that $R_1 - V$ is nonempty since $A \in R_1 - V$; also observe that $X \subseteq V$ and $\bar{u}[V]$ are constants because by construction of $T^*_{\mathbf{r}}$ all the variables in $\bar{u}$ and $\bar{v}$ are distinct. Let z be a tuple on R_1 defined as follows: $z[V] = \bar{u}[V]$ and for all $B \in R_1 - V$, $z[B]$ is a unique constant. We are going to prove (1) z is expansible in $\mathbf{r}$, and hence, by Theorem 1, $\mathbf{r} \cup \{z\}$ is consistent with respect to F, and (2) $\bar{v}[A]$ and $\bar{u}[A]$ are variables.

LEMMA 1. *$z[V]$ is expansible in* $\mathbf{r}$.

Proof. This follows from $\bar{u}[V] = z[V]$ and $\bar{u}[V]$ is expansible in $\mathbf{r}$. □

We now prove that the expansion computation of $z[V]$ in $\mathbf{r}$ does not equate to a constant any unique variable on columns $R_1 - V$ of $aug_U(z[V])$. Therefore, if we replace these unique variables with the unique constants in $z[R_1 - V]$, the same expansion computation shows that z is expansible in $\mathbf{r}$.

LEMMA 2. *Let $\bar{zv}$ be the expansion of $z[V]$ in* $\mathbf{r}$. *Let $Y = \{B | B \in U$ and $\bar{zv}[B]$ is a constant$\}$. $Y \cap (R_1 - V) = \emptyset$.*

Proof. We first prove that $\bar{u}[Y] = \bar{zv}[Y] = \bar{v}[Y]$.

Let $\tau_1 = < y_1, Y_1 \to B_1 >, \ldots, \tau_l = < y_l, Y_l \to B_l >, l \geq 0$, be a sequence that can strictly expand $aug_U(z[V])$ in an expansion computation of $z[V]$ in $\mathbf{r}$. From Lemma 1, no contradiction is revealed in this expansion.

Let $z_j = \tau_j(\cdots(\tau_1(z_0))\cdots)$, for $1 \leq j \leq l$, where $z_0 = aug_U(z[V])$. It is not difficult to prove that for $0 \leq j \leq l$ and for any $C \in U$ if $z_j[C]$ is a constant, then $\bar{u}[C] = z_j[C]$. This implies $\bar{u}[Y] = \bar{zv}[Y]$. Similarly, we can prove that for $0 \leq j \leq l$ and for any $C \in U$ if $z_j[C]$ is a constant, then $\bar{v}[C] = z_j[C]$. This implies $\bar{v}[Y] = \bar{zv}[Y]$.

Now assume that there is $B \in Y \cap (R_1 - V)$. Then $B \notin V$. But since $\bar{u}[Y] = \bar{v}[Y]$, $\bar{u}[B] = \bar{v}[B]$ and thus, by definition of V, $B \in V$. This is a contradiction. □

Now we prove that z is expansible in $\mathbf{r}$, that is that $\mathbf{r} \cup \{z\} \in CONS(\mathbf{R}, F)$.

LEMMA 3. (a) *z is expansible in* $\mathbf{r}$, *and* (b) $\mathbf{r} \cup \{z\} \in CONS(\mathbf{R}, F)$.

Proof. From Lemma 2, the expansion computation of $z[V]$ in $\mathbf{r}$ does not equate to a constant any unique variable on columns $R_1 - V$ of $aug_U(z[V])$. Therefore, by replacing each unique variable of $aug_U(z[V])$ on column A, $A \in R_1 - V$, with the unique constant $z[A]$, the same sequence of expansions shows that z is expansible in $\mathbf{r}$.

Part (b) follows from part (a) above and Theorem 1. □

The following lemma says that if we can apply an fd-rule to two tuples in $T^*_{\mathbf{r}}$, then the fd-rule equates only variables.

LEMMA 4. *Let $\bar{u}$, $\bar{v}$, and A be as defined above. $\bar{u}[A]$ and $\bar{v}[A]$ are variables.*

Proof. Without loss of generality, we assume that $\bar{v}[A]$ is a constant. If $\bar{u}[A]$ is a constant, then by assumption that $\bar{u}[X] = \bar{v}[X]$ and $\bar{u}[A] \neq \bar{v}[A]$, $\mathbf{r} \notin CONS(\mathbf{R}, F)$, which is a contradiction to our assumption about $\mathbf{r}$. Hence $\bar{u}[A]$ must be a variable. Let z and V be as defined above. We now show that z and $\bar{v}$ violate the fd $X \to A$. Since $A \notin V$ (because $\bar{u}[A] \neq \bar{v}[A]$), $z[A] \neq \bar{v}[A]$ and both are constants by definition of z; thus $X \to A$ is violated by these two tuples, because $z[X] = \bar{v}[X]$ by definition of z. This implies $\mathbf{r} \cup \{z\} \notin CONS(\mathbf{R}, F)$. But this is a contradiction to Lemma 3. Therefore both $\bar{v}[A]$ and $\bar{u}[A]$ must be variables. □

4.3. Proof of main lemma. We now want to prove that in any chase of $T^*_{\mathbf{r}}$ no variable is replaced by a constant, provided the database scheme is ctm. Before proving this, we need to introduce the following notation. Let T be a tableau, and let τ be the triple $< u, v, X \rightarrow A >$, where u and v are tuples defined on U and $X \rightarrow A$ is a nontrivial fd in $F^+|R_i$, for some $R_i \in \mathbf{R}$. We say that τ is an *fd-rule for* T (with respect to F) if u and v are tuples in T and $u[X] = v[X]$. If $u[A] \neq v[A]$, we say that τ is a *strict fd-rule for* T. $\tau(T)$ denotes the tableau that results by equating $u[A]$ with $v[A]$. We say that the sequence $\tau_1, \ldots, \tau_m$ of triples of the above form is a *sequence of (strict) fd-rules for* T if τ_1 is a (strict) fd-rule for T, and for $2 \leq i \leq m$, τ_i is a (strict) fd-rule for $\tau_{i-1}(\cdots(\tau_1(T))\cdots)$.

LEMMA 5. *Let* $\mathbf{R}$ *be a* ctm *database scheme with respect to* F. *Then for all* $\mathbf{r} \in CONS(\mathbf{R}, F)$, *there is no finite sequence of* fd*-rules for* $T^*_{\mathbf{r}}$ *(the expansion of* $\mathbf{r}$*) that replaces a variable in* $T^*_{\mathbf{r}}$ *by a constant from* $\mathbf{r}$.

Proof. We prove it by induction on the length, k, of any sequence of fd-rules for $T^*_{\mathbf{r}}$.

Basis. $k = 1$. By Lemma 4, there is no sequence of fd-rules for $T^*_{\mathbf{r}}$ of length at most 1 that replaces a variable of $T^*_{\mathbf{r}}$ by a constant from $\mathbf{r}$.

Induction. $k > 1$. Assume that for all $\mathbf{r} \in CONS(\mathbf{R}, F)$, there is no sequence of fd-rules for $T^*_{\mathbf{r}}$ of length at most $k = m - 1 \geq 1$ that replaces a variable of $T^*_{\mathbf{r}}$ by a constant from $\mathbf{r}$. We show the lemma holds for $k = m$. In fact, if it does not hold for $k = m$, we can construct a consistent state $\mathbf{r}_1$ that will let us show that there is a sequence of fd-rules for $T^*_{\mathbf{r}_1}$, the expansion of $\mathbf{r}_1$, of length no more than $m - 1$ that replaces a variable in $T^*_{\mathbf{r}_1}$ by a constant from $\mathbf{r}_1$, contradicting the inductive hypothesis.

Let $\mathbf{r} \in CONS(\mathbf{R}, F)$ and let $T^*_{\mathbf{r}}$ be the expansion of $\mathbf{r}$. We assume there is a sequence of fd-rules $\tau_1 =< u_1, v_1, X_1 \rightarrow A_1 >, \tau_2 =< u_2, v_2, X_2 \rightarrow A_2 >, \ldots, \tau_m =< u_m, v_m, X_m \rightarrow A_m >$, for $T^*_{\mathbf{r}}$, in which τ_m is the first fd-rule that replaces a variable in $T^*_{\mathbf{r}}$ by a constant from $\mathbf{r}$. Without loss of generality, we further assume that each fd $X_j \rightarrow A_j$ is in F, for $1 \leq j \leq m$, and $u_m[A_m]$ is a constant and $v_m[A_m]$ is a variable. By the inductive hypothesis, all of the other fd-rules equate only variables.

We now construct a state $\mathbf{r}_1$ that shows that the sequence $\tau_1 \cdots \tau_m$ does not exist. Let us consider the first fd-rule, $\tau_1 =< u_1, v_1, X_1 \rightarrow A_1 >$, and assume without loss of generality that $X_1 A_1 \subseteq R_1$, for some $R_1 \in \mathbf{R}$. Then $u_1[X_1] = v_1[X_1]$, v_1 and u_1 are both in $T^*_{\mathbf{r}}$, and we may assume $u_1[A_1] \neq v_1[A_1]$; otherwise, $\tau_2 \cdots \tau_m$ will be a sequence of fd-rules for $T^*_{\mathbf{r}}$ of length at most $m - 1$ that replaces the variable $v_m[A_m]$ with the constant $u_m[A_m]$, contradicting the inductive hypothesis. From the inductive hypothesis, $u_1[A_1]$ and $v_1[A_1]$ are variables. Let $V = \{B | B \in R_1 \text{ and } u_1[B] = v_1[B]\}$ and let z be the (total) tuple on R_1 defined as follows: $z[V] = u_1[V]$ and $z[R_1 - V]$ are all unique constants (i.e., new constants that are not in $\mathbf{r}$). Then from Lemma 3, $\mathbf{r}_1 = \mathbf{r} \cup \{z\} \in CONS(\mathbf{R}, F)$.

We now expand every tuple in $T^*_{\mathbf{r}}$ by the tuple z (and fd's in $F^+|R_1$) as much as possible and let the final result be T'. (Note that we cannot assume that $T^*_{\mathbf{r}}$ and T' are the same, because z, which contains unique constants, is not in $\mathbf{r}$, and $T^*_{\mathbf{r}}$ is the expansion with respect to only the tuples in $\mathbf{r}$.) Furthermore, let z' be the expansion of z in the state $\mathbf{r}$, and let $T_1 = T' \cup \{z'\}$. In the following, we show that T_1 is identical to $T^*_{\mathbf{r}_1}$ up to renaming of variables and that $< \tau'_2, \ldots, \tau'_m >$ is a sequence of fd-rules for T_1 (hence for $T^*_{\mathbf{r}_1}$) that equates a variable with a constant, where $\tau'_j =< u'_j, v'_j, X_j \rightarrow A_j >, u'_j \in T_1$ originates from u_j and $v'_j \in T_1$ originates from v_j, $2 \leq j \leq m$. The proofs are stated as the following claims. (See the appendix for the proofs of these claims.)

CLAIM 1. T_1 *is the expansion of* $\mathbf{r}_1$. *That is,* T_1 *is identical to the* $T^*_{\mathbf{r}_1}$ *up to the renaming of variables.*

CLAIM 2. $u_1[A_1]$ *and* $v_1[A_1]$ *are both replaced by the constant* $z[A_1]$, *that is, the effect of the first fd-rule has been enforced when computing* T_1 *from* $T^*_{\mathbf{r}}$. *(Note that this is different from saying that* $u_1[A_1]$ *and* $v_1[A_1]$ *are equated directly in the computation of* T_1 *from* $T^*_{\mathbf{r}}$. *Thus, the expansion* T_1 *contains no repeated variables still holds.)*

In the following claim, we prove that there is a sequence of fd-rules $\tau'_1 =< u'_1, v'_1, X_1 \to A_1 >, \ldots, \tau'_m =< u'_m, v'_m, X_m \to A_m >$ for T' that is "parallel" to the sequence $\tau_1, \ldots, \tau_m$ for $T^*_{\mathbf{r}}$ in the sense that u'_j and u_j come from the same tuple in $T^*_{\mathbf{r}}$ and so do v'_j and v_j. This is intuitively correct because the computation of T' from $T^*_{\mathbf{r}}$ does not make distinct any repeated occurrences of symbols; therefore we can always apply the fd's in $\tau_1, \ldots, \tau_m$ to the corresponding tuples in T'. A formal argument is given in terms of containment mappings by the following claim.

CLAIM 3. *There are mappings $\theta_0, \ldots, \theta_{m-1}$ such that for all $0 \le j \le m-1$,*

(i) *θ_j is a containment mapping from $\tau_j(\cdots(\tau_0(T^*_{\mathbf{r}}))\cdots)$ to $\tau'_j(\cdots(\tau'_0(T'))\cdots)$;*

(ii) *θ_j satisfies the condition that any variable δ is mapped to either itself or to the unique constant from $z[R_1 - V]$ that replaced δ when we computed T', where for $1 \le l \le j$, $\tau'_l =< \theta_{l-1}(u_l), \theta_{l-1}(v_l), X_l \to A_l >$, and τ_0 and τ'_0 are defined to be such that $\tau_0(T^*_{\mathbf{r}}) = T^*_{\mathbf{r}}$ and $\tau'_0(T') = T'$.*

We now complete the proof of the lemma by showing that there is a sequence of fd-rules for T' (hence for $T_1 = T^*_{\mathbf{r}_1}$) of length no more than $m-1$ that replaces a variable by a constant. Let us consider $\tau_m =< u_m, v_m, X_m \to A_m >$ and θ_{m-1} defined as above. Recall that $u_m[A_m]$ is a constant from $\mathbf{r}$ and $v_m[A_m]$ is a variable. Since θ_{m-1} is a containment mapping, we have that $\theta_{m-1}(u_m)[X_m] = \theta_{m-1}(v_m)[X_m]$, because $u_m[X_m] = v_m[X_m]$. Then from part (ii) of Claim 3, $\theta_{m-1}(v_m)[A_m]$ must be the variable $v_m[A_m]$; otherwise $\theta_{m-1}(v_m)$ and $\theta_{m-1}(u_m)$ will violate $X_m \to A_m$, which is a contradiction to $\mathbf{r}_1 \in CONS(\mathbf{R}, F)$. Therefore, applying τ'_m to $\tau'_{m-1}(\cdots(\tau'_0(T'))\cdots)$ equates the variable $v_m[A_m]$ to the constant $u_m[A_m]$, where $\tau'_m =< \theta_{m-1}(u_m), \theta_{m-1}(v_m), X_m \to A_m >$. Then, Claims 1 and 2 and the above discussion show that $\mathbf{r}_1$ is a consistent state for which $< \tau'_2, \tau'_3, \ldots, \tau'_m >$ is a sequence of fd-rules for $T^*_{\mathbf{r}_1}$ of length at most $m-1$ that equates a variable with a constant. But this is a contradiction to the inductive hypothesis of the lemma.

This completes the induction and the proof of this lemma. □

4.4. Main theorem. The following theorem follows from the way we compute $T^*_{\mathbf{r}}$, Lemma 5, and the definition of bounded schemes.

THEOREM 2. *Let $\mathbf{R}$ be a* ctm *data base scheme with respect to F. Then $\mathbf{R}$ is bounded with respect to F.*

5. Computing $[X]^F_{\mathbf{r}}$ by simple chase join expressions. We now show that the unions of simple cje's that compute the total projections of representatives instances can be constructed in exponential time in the scheme size.

We first give a lemma stating a close relationship between the sequences of fd's used in the expansion computations and derivation sequences.

LEMMA 6. *Let $\mathbf{R}$ be a database scheme, and let F be a set of functional dependencies. Let $\mathbf{r}$ be a consistent state on $\mathbf{R}$ with respect to F and let t be a total tuple on some $R_i \in \mathbf{R}$ such that t is expansible in $\mathbf{r}$ with respect to F. If some sequence of* fd'*s $< X_1 \to A_1, \ldots, X_m \to A_m >, m \ge 0$, in F strictly expands $aug_U(t)$ in an expansion computation of t in $\mathbf{r}$ and the expanded tuple has constants on $X \subseteq U$, then $< X_1 \to A_1, \ldots, X_m \to A_m >$ is a* ds *of R_i covering X.*

Proof. The lemma follows immediately from the definitions. □

Unlike independent schemes, when ctm schemes are considered, one fd may be embedded in more than one relation scheme, as illustrated by Example 1. To deal with such a multiple embedding situation, we define for any set of attributes $W \subseteq U$

$$\downarrow W \downarrow = \cup\{\pi_W(R_j) \mid R_j \in \mathbf{R}, W \subseteq R_j\}.$$

That is, $\downarrow W \downarrow$ is the relational expression that is the union of projections on $W$ of all the relation schemes that embed $W$.

In fact, Lemma 5 and the way the expansion $T^*_{\mathbf{r}}$ was computed in the last section suggest a method to extract the relational expression that computes the total projections of representative instances. Given a consistent state $\mathbf{r}$ with respect to F and a subset $X \subseteq U$, let t be a tuple in $[X]^F_{\mathbf{r}}$. From the result of the last subsection, $t = \bar{v}[X]$ for some tuple $\bar{v}$ in $T^*_{\mathbf{r}}$, where $\bar{v}$ is the expansion of some tuple v in some $r_p \in \mathbf{r}$. From Lemma 6, let $< X_1 \to A_1, \ldots, X_m \to A_m >, m \geq 0$, be the ds of R_p covering X that strictly expands $aug_U(v)$ in the expansion computation of v in $\mathbf{r}$ with respect to F. We define the expression $E = \pi_X(R_p \bowtie\downarrow X_1A_1 \downarrow\bowtie \cdots \bowtie\downarrow X_mA_m \downarrow)$. Clearly, from the way $\bar{v}$ is computed, we have $t \in E(\mathbf{r})$. By Theorem 5 in [MUV], $E(\mathbf{r}) \subseteq [X]^F_{\mathbf{r}}$. It is not difficult to see that we can transform the expression E into a union of simple cje's for the ds $< X_1 \to A_1, \ldots, X_m \to A_m >$, in which each simple cje corresponds to a choice from the relation schemes embedding the fd's in the ds.

From the above discussion, the union of all simple cje's for all ds's of R_i's covering X produces exactly the X- total projection of representative instances. Hence we have an algorithm to compute the X-total projections for consistent states of any ctm database scheme $\mathbf{R}$ with respect to a set F of embedded fd's: For each $R_i \in \mathbf{R}$ such that $R_i^+ \supseteq X$, find all nonequivalent and nonredundant ds's of R_i covering X. The ds's are of the form $< X_1 \to A_1, \ldots, X_m \to A_m >$, $m \geq 0$, where $X_j \to A_j \in F$ for all $1 \leq j \leq m$. For each of these ds's, construct the union of simple cje's E for it as above. The union of all these E's is an expression for computing the X-total projection of representative instances. The following theorem is a consequence of the above discussion.

THEOREM 3. *Let* $\mathbf{R}$ *be a* ctm *database scheme with respect to F, where F is a set of* fd*'s embedded in* $\mathbf{R}$. *Then for any* $X \subseteq U$ *and for any consistent state* $\mathbf{r}$ *of* $\mathbf{R}$, $[X]^F_{\mathbf{r}}$, *the X-total projection of the representative instance of* $\mathbf{r}$, *can be computed with a union of projections onto X of simple* cje*'s that cover X.*

By an algorithm given in [AC3], [C], we can optimize the unions of simple cje's obtained above in polynomial time of size of these expressions. The returned expression is minimal both in the number of subexpressions and in the number of join operations [AC3], [C]. However, the algorithm given above does not suggest an efficient way of doing it, since in general, there may be an exponential number of simple cje's. As shown in [AC1], [IIK], when independent schemes are considered, for each $X \subseteq U$ and each $R_j \in \mathbf{R}$ such that $R_j^+ \supseteq X$, there is just one "maximum" simple cje of R_j covering X that has to be considered. This is not true of ctm schemes because there is no such maximum simple cje, as shown below. We usually examine all simple cje's for all the combinatorics of derivations for each $R_j \in \mathbf{R}$. Let us consider the following example.

Example 2. Let $\mathbf{R} = \{R_1(ID), R_2(IC), R_3(CD)\}$ and $F = \{I \to D, I \to C, C \to I, C \to D\}$. This is a Course-Instructor-Department database scheme. It is not difficult to see that $\mathbf{R}$ is not independent with respect to F. However $\mathbf{R}$ is ctm with respect to F; Example 1 in [HC] shows this fact. To compute the CD-total projection of representative instances, for example, we need to consider the following ds's that cover CD: the empty ds $<>$ of R_3, $< C \to D >$ and $< I \to D >$ of R_2, and $< I \to C >$ of R_1. Their corresponding simple cje's are $E_1 = R_3, E_2 = \pi_{CD}(R_2 \bowtie R_3), E_3 = \pi_{CD}(R_2 \bowtie R_1)$, and $E_4 = \pi_{CD}(R_1 \bowtie R_2)$. Observe that E_2 and E_3 are incomparable[1]. In particular, let $\mathbf{r}_1$ and $\mathbf{r}_2$ be the consistent states shown in Table 1. It is easy to verify that $E_2(\mathbf{r}_1) \supset E_3(\mathbf{r}_1)$ and $E_3(\mathbf{r}_2) \supset E_2(\mathbf{r}_2)$.

[1] E and E' are incomparable if neither $E \subseteq E'$ nor $E' \subseteq E$ holds, where $E \supseteq E'(E \supset E')$ if and only if $E(\mathbf{r}) \supseteq E'(\mathbf{r})(E(\mathbf{r}) \supset E'(\mathbf{r}))$ for every consistent state $\mathbf{r}$.

TABLE 1
States for Example 2.

$\mathbf{r}_1$				$\mathbf{r}_2$			
I	C	D	Tag	I	C	D	Tag
i	c		R_2	i	c		R_2
	c	d	R_3	i		d	R_1

The reader may also note that $< C \rightarrow D >$ and $< I \rightarrow D >$ both are minimal ds's of R_2 covering CD in the terms of [C], and both are minimal derivations of CD from R_2 in terms of [IIK], and neither is "embedded" in the other.

Then from the above we get the expression

$$R_3 \cup \pi_{CD}(R_2 \bowtie R_3) \cup \pi_{CD}(R_2 \bowtie R_1) \cup \pi_{CD}(R_1 \bowtie R_2),$$

which is equivalent to $R_3 \cup \pi_{CD}(R_2 \bowtie R_1)$.

It is interesting to note that the polynomial constructions of algebra computing X-total projections for independent schemes [AC1], [IIK], [S2] follow immediately from the uniqueness property of independent schemes and our results for ctm schemes. When independent schemes are considered, the problem of combinatorics of derivations disappears because at most one ds has to be considered for each relation scheme, and each term $\downarrow X_iA_i \downarrow$ becomes $\pi_{X_iA_i}(R_{i_j})$ for the unique relation scheme R_{i_j} that embeds the fd $X_i \rightarrow A_i$.

Alternatively, to compute the total projection $[X]^F_{\mathbf{r}}$ for ctm schemes, we may first compute the expansion $T^*_{\mathbf{r}}$ of $\mathbf{r}$ using the expansion computation as a subroutine. This way we can compute total projections in linear time (in the state size) without predetermining any relational algebra expression, assuming each strict expansion is charged one unit of time. One advantage of this alternative is that the expansion $T^*_{\mathbf{r}}$, once computed, can be used for answering all queries until state $\mathbf{r}$ is updated next time, and all we have to do for each additional query is a total projection operation on $T^*_{\mathbf{r}}$. Another advantage is the uniformity of enforcing constraints and processing queries. The expansion computation is all we need for both kinds of transactions. This method is particularly meaningful when queries are more often imposed than updates and the universe of attributes is not very large.

6. Computing $[X]_{\mathbf{r}}$ with respect to egd's. In this section, we prove that if the constraints considered is a set of egd's and the database scheme has a lossless join with respect to the constraints, then the database scheme is bounded if it is ctm. The following theorem is due to Wang and Graham [W], [WG].

THEOREM 4. *Let* $\mathbf{R}$ *be a database scheme, let D be a set of* egd'*s on U such that the scheme* $\mathbf{R}$ *has a lossless join with respect to D, and let G be the embedded* fd'*s implied by D. Then* $\mathbf{R}$ *is* ctm *with respect to D if and only if the following statements hold.*

1. *G is a cover of D;*
2. $\mathbf{R}$ *is* ctm *with respect to G.*

The following is a corollary to Theorem 4 above.

THEOREM 5. *Let* $\mathbf{R}$ *be a database scheme and let D be a set of* egd'*s on U such that the scheme* $\mathbf{R}$ *has a lossless join with respect to D. If* $\mathbf{R}$ *is* ctm *with respect to D, then*

1. $\mathbf{R}$ *is bounded with respect to D;*
2. *A relational algebra expression for computing total projections of representative instances can be constructed in exponential time in the number of attributes.*

Proof. Assume $\mathbf{R}$ is a ctm database scheme with respect to D. Let G be a cover of the embedded fd's implied by D. We can find G by enumerating all fd's and selecting those logically implied by D. From part (1) of Theorem 4 and the cover-insensitive property of total projections, for every consistent state $\mathbf{r}$ with respect to D, and for every $X \subseteq U$, $[X]_{\mathbf{r}}^{D} = [X]_{\mathbf{r}}^{G}$. Then from part (2) of Theorem 4, $\mathbf{R}$ is ctm with respect to G. Thus, the method in the last section for computing $[X]_{\mathbf{r}}^{G}$ can be used to compute $[X]_{\mathbf{r}}^{D}$ as well. Then the theorem follows from the results in that section. □

A useful case to which the above results can be applied is when D is a set of fd's and the scheme $\mathbf{R}$ has a lossless join with respect to D.

7. Computing $[X]_{\mathbf{r}}$ with respect to fd's and $\bowtie$ R. When $D = F \cup \{\bowtie \mathbf{R}\}$ is given as the constraints and X is taken from a single relation scheme, our method to compute the X-total projection with respect to a set of embedded fd's F can also be used.

The following lemma is Theorem 5 in [GW] (or Thm. 5.4.1 in [WG]).

LEMMA 7. *Let* $\mathbf{R}$ *be a* ctm *database scheme with respect to* $D = F \cup \{\bowtie \mathbf{R}\}$, *where* F *is a set of* fd*'s. Let* G *be the set of* fd*'s implied by* D. *Then* $\mathbf{R}$ *cover embeds* G.

From the above lemma, we have the following result.

THEOREM 6. *Let* $\mathbf{R}$ *be a* ctm *database scheme with respect to* $D = F \cup \{\bowtie \mathbf{R}\}$, *where* F *is a set of* fd*'s. Let* G *be the set of* fd*'s implied by* D *and let* H *be an embedded cover of* G. *Then for any* $\mathbf{r} \in CONS(\mathbf{R}, D)$, *and for any* $X \subseteq R_i, R_i \in \mathbf{R}$, $[X]_{\mathbf{r}}^{D} = [X]_{\mathbf{r}}^{H}$.

Proof. By Lemma 7, $H \cup \{\bowtie \mathbf{R}\}$ is equivalent to $F \cup \{\bowtie \mathbf{R}\}$. Since the chase process is cover insensitive, we have $[X]_{\mathbf{r}}^{D} = [X]_{\mathbf{r}}^{H \cup \{\bowtie \mathbf{R}\}}$. Then by Theorem 7 in [CM2], $[X]_{\mathbf{r}}^{H} = [X]_{\mathbf{r}}^{H \cup \{\bowtie \mathbf{R}\}}$, and the theorem follows. □

Given a set of fd's G, if $\mathbf{R}$ cover embeds G, an algorithm in [GY] can find an embedded cover H of G in polynomial time. Therefore the total projection of any set of attributes $X \subseteq R_i$, $R_i \in \mathbf{R}$, for ctm schemes with respect to fd's plus the jd $\bowtie \mathbf{R}$ can be computed by our methods in §5.

8. Conclusions. We have shown that constant-time-maintainable database schemes are bounded with respect to dependencies in the following cases: (1) only cover-embedded functional dependencies appear as constraints; (2) only equality-generating dependencies appear as constraints and the database scheme has a lossless join. Interestingly, we showed that total projections can be computed via unions of projections of simple chase join expressions. Therefore by previous results in [C], we can compute optimally the total projections of the representative instances of constant-time-maintainable database schemes. We also proved that by the same method we can compute X-total projections when a set of functional dependencies and the join dependency of the database scheme are the constraints, for any X included in some relation in the database scheme.

Within the above context, our results show that fast constraint enforcement is stronger than efficient query processing in the sense that every constant-time-maintainable scheme is bounded with respect to dependencies. Our results also showed that the class of constant-time-maintainable schemes is highly desirable with respect to query processing. Therefore the class of constant-time-maintainable database schemes is the largest class of database schemes, which is highly desirable with respect to both constraint enforcement and query processing. Importantly, this class of schemes can be effectively recognized by known methods [GW], [W], [WG]. The previously known largest class of database schemes with these desirable properties is the class of independent database schemes which is a proper subclass of constant-time-maintainable schemes.

The boundedness of constant-time-maintainable schemes was proven in the absence of the essential uniqueness property [GY], [S2] of independent schemes. As a consequence, the polynomial construction of the relational expressions that compute total projections for independent schemes with only functional dependencies [AC1], [C], [IIK], [S2], [S3] follows immediately from the uniqueness property and our construction.

Our work illustrates a more general technique to prove boundedness, and it provides more insight into characterizing boundedness.

Appendix.

A. Proofs of claims in main lemma. In this section, we present the proofs of Claims 1, 2, and 3 in Lemma 5. These proofs should be read in the context of that lemma, since we are assuming all the assumptions, definitions, and notation introduced there. First, we present a claim needed in Claims 1 and 3.

CLAIM 0. *No variables are replaced by constants from $z[V]$ when computing T' from $T^*_{\mathbf{r}}$.*

Proof. Let t_1 be a tuple in $T^*_{\mathbf{r}}$. Let $\lambda_1 =< z, W_1 \rightarrow B_1 >, \ldots, \lambda_m =< z, W_m \rightarrow B_m >$ be a sequence that can strictly expand t_1 when computing T' from $T^*_{\mathbf{r}}$; observe that the fd's are from $F^+|R_1$. Since $\lambda_1, \ldots, \lambda_m$ can strictly expand t_1 and $\mathbf{r}_1 \in CONS(\mathbf{R}, F)$, $t_1[B_l]$ is a variable (because $z[B_l]$ is a constant) for $1 \leq l \leq m$.

We claim that for $1 \leq l \leq m, B_l \in R_1 - V$. Assume otherwise. That is, assume there is B_q for some $1 \leq q \leq m$ such that $B_q \in V$. Let $Y_1 = \{B | t_1[B] = z[B]\}$. Then $Y_1 \subseteq V$ since $z[R_1 - V]$ are all unique constants. Thus $t_1[Y_1] = z[Y_1] = u_1[Y_1]$. Also, notice that $B_q \notin Y_1$, because $t_1[B_q]$ is a unique variable (and $z[B_q]$ is a constant). From a property of the chase process and the fact that t_1 and z have common values exactly on Y_1, we have $Y_1 \rightarrow B_q \in F^+|R_1$. Therefore one step of chasing $T^*_{\mathbf{r}}$ by applying the fd-rule $< u_1, t_1, Y_1 \rightarrow B_q >$ to u_1 and t_1 will equate the variable $t_1[B_q]$ to the constant $u_1[B_q]$, which is a contradiction to the inductive hypothesis. Thus the claim $B_l \in R_1 - V$ for $1 \leq l \leq m$ holds, and therefore Claim 0 is proven. □

CLAIM 1. *T_1 is the expansion of $\mathbf{r}_1$. That is, T_1 is identical to $T^*_{\mathbf{r}_1}$ up to the renaming of variables.*

Proof. Assume that T_1 is not the expansion of $\mathbf{r}_1$. Then there is a tuple $t \in T_1$ and a tuple $p \in r_l$ for some $r_l \in \mathbf{r}_1$ and an fd $Y_l \rightarrow B_l \in F|R_l$ such that $< p, Y_l \rightarrow B_l >$ can strictly expand t. Then $t[B_l]$ must be a variable; otherwise, t and p violate $Y_l \rightarrow B_l$, which is a contradiction to $\mathbf{r}_1 \in CONS(\mathbf{R}, F)$. We claim that t comes from a tuple in $T^*_{\mathbf{r}}$, and p is a tuple from $\mathbf{r}$. First, notice that t cannot be the tuple z' because z' is already the expansion of z in $\mathbf{r}_1$. Then t must come from a tuple in $T^*_{\mathbf{r}}$ (that is, $t \in T'$). Therefore, p cannot be z, because at this point we already expanded every tuple from $T^*_{\mathbf{r}}$ by z as much as possible. Therefore our claim holds. Since we are expanding $t \in T'$ by p from $\mathbf{r}$, and since by Claim 0 to obtain T' from $T^*_{\mathbf{r}}$ we have just replaced some variables with unique constants that do not appear in $\mathbf{r}$, these changes certainly will not create the possibility for new strict expansions by tuples from $\mathbf{r}$. Therefore this is a contradiction. Thus Claim 1 holds. □

CLAIM 2. *$u_1[A_1]$ and $v_1[A_1]$ are both replaced by the constant $z[A_1]$, that is, the effect of the first* fd-*rule has been enforced when computing T_1 from $T^*_{\mathbf{r}}$.*

Proof. It is not difficult to see that in the above computation of T_1, $u_1[A_1]$, and $v_1[A_1]$ are both equated to $z[A_1]$ because expanding u_1 by $< z, X_1 \rightarrow A_1 >$ will set $u_1[A_1]$ to $z[A_1]$, and expanding v_1 by $< z, X_1 \rightarrow A_1 >$ will set $v_1[A_1]$ to $z[A_1]$. Therefore Claim 2 holds. □

CLAIM 3. *There are mappings $\theta_0, \ldots, \theta_{m-1}$ such that for all $0 \leq j \leq m-1$,*

1. *θ_j is a containment mapping from $\tau_j(\cdots(\tau_0(T^*_{\mathbf{r}}))\cdots)$ to $\tau'_j(\cdots(\tau'_0(T'))\cdots)$;*
2. *θ_j satisfies the condition that any variable δ is mapped to either itself or to the unique constant from $z[R_1 - V]$ that replaced δ when we computed T',*

where for $1 \le l \le j, \tau'_l = < \theta_{l-1}(u_l), \theta_{l-1}(v_l), X_l \to A_l >$, *and* τ_0 *and* τ'_0 *are defined to be such that* $\tau_0(T^*_{\mathbf{r}}) = T^*_{\mathbf{r}}$ *and* $\tau'_0(T') = T'$.

Proof. The proof is by induction on j. □

Basis. $j = 0$. Let $t_1, t_2, \ldots, t_q$ be the tuples in $T^*_{\mathbf{r}}$; for each $t_l, 1 \le l \le q$, let t'_l be the tuple in T' that comes from t_l, that is, t'_l is t_l with possibly some of its variables replaced by the unique constants from $z[R_1 - V]$ from Claim 0. Thus it should be clear that there is a containment mapping θ_0 from $T^*_{\mathbf{r}}$ to T' given by $\theta_0(t_l) = t'_l$ for $1 \le l \le q$; and θ_0 is such that any variable δ is mapped to either itself or to the unique constant from $z[R_1 - V]$ that replaced δ when we computed T'.

Induction. We now assume that our claim is true for $j = k - 1, 1 \le k \le m - 1$. We prove it for $j = k$.

First we notice that the sequence $\tau'_1, \ldots, \tau'_k$ are fd-rules for T' from the inductive hypothesis; and therefore $\theta_{k-1}(u_k)[X_k] = \theta_{k-1}(v_k)[X_k]$. Let $T^*_{k-1} = \tau_{k-1}(\cdots(\tau_0(T^*_{\mathbf{r}}))\cdots)$ and let $T'_{k-1} = \tau'_{k-1}(\cdots(\tau'_0(T'))\cdots)$. We now derive a containment mapping θ_k from θ_{k-1} and τ_k. Since $u_k[A_k]$ and $v_k[A_k]$ must be some distinct variables δ_1 and δ_2 in T^*_{k-1} (because at the beginning of the proof of Lemma 5 we have assumed that all fd-rules except τ_m equate only variables), θ_{k-1} maps each of them to either itself or to the unique constant from $z[R_1 - V]$ that replaced it when we computed T'. There are two cases to analyze depending on whether both $\theta_{k-1}(\delta_1)$ and $\theta_{k-1}(\delta_2)$ are unique constants from $z[R_1 - V]$.

Case 1. $\theta_{k-1}(\delta_1)$ and $\theta_{k-1}(\delta_2)$ are both unique constants from $z[R_1 - V]$. Then $\theta_{k-1}(\delta_1) = \theta_{k-1}(\delta_2)$, because we only introduced at most one unique constant on each column in the construction of T'. (Then τ'_k is a trivial fd-rule for T'_{k-1} that makes no change to T'_{k-1}, i.e., $T'_k = T'_{k-1}$.) Without loss of generality, we assume the application of τ_k to T^*_{k-1} replaces δ_1 with δ_2. We define θ_k to be θ_{k-1} except that θ_k does not need to be defined for δ_1.

Case 2. Either $\theta_{k-1}(\delta_1)$ or $\theta_{k-1}(\delta_2)$ (or both) are variables. Assume without loss of generality that $\theta_{k-1}(\delta_1)$ is a variable and that we apply τ_k to T^*_{k-1} by replacing δ_1 with δ_2. Then we apply τ'_k to T'_{k-1} by replacing $\theta_{k-1}(\delta_1)$ with $\theta_{k-1}(\delta_2)$ and define θ_k to be θ_{k-1} except that θ_k does not need to be defined for δ_1.

The mapping θ_k defined above satisfies the conditions required by our claim. This completes the induction and our proof of Claim 3. □

Acknowledgments. We are grateful to the anonymous referees for their constructive comments and suggestions that made this paper more readable.

REFERENCES

[ABU] A. V. Aho, C. Beeri, and J. D. Ullman, *The theory of joins in relational databases*, ACM Trans. Database Systems, 4 (1979), pp. 297–314.

[AC1] P. Atzeni and E. P. F. Chan, *Efficient query answering in the representative instance approach*, in Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1985, pp. 181–188.

[AC2] ———, *Independent database schemes under functional and inclusion dependencies*, in Proceedings of the Thirteenth International Conference on Very Large Databases, Brighton, England, 1987, pp. 159–166.

[AC3] ———, *Efficient optimization of simple chase join expressions*, ACM Trans. Database Systems, 14 (1989), pp. 212–230.

[ASU] A. V. Aho, Y. Sagiv, and J. D. Ullman, *Equivalence of relational expressions*, SIAM J. Comput., 8 (1979), pp. 218–246.

[BH] C. BEERI AND P. HONEYMAN, *Preserving functional dependencies*, SIAM J. Comput., 10 (1981), pp. 647–656.

[BV] C. BEERI AND M. Y. VARDI, *A proof procedure for data dependencies*, J. Assoc. Comput. Mach., 31 (1984), pp. 718–741.

[BrV] V. BROSDA AND G. VOSSEN, *Update and retrieval in a relational database through a universal schema interface*, ACM Trans. Database Systems, 13 (1988), pp. 449–485.

[C] E. P. F. CHAN, *Optimal computation of total projections with unions of simple chase join expressions*, in Proceedings of ACM SIGMOD Annual Meeting, Boston, MA, June 1984, pp. 149–163.

[CM1] E. P. F. CHAN AND A. O. MENDELZON, *Answering queries on the embedded-complete database schemes*, J. Assoc. Comput. Mach., 34 (1987), pp. 349–375.

[CM2] ———, *Independent and separable database schemes*, SIAM J. Comput., 16 (1987), pp. 841–851.

[CGKV] S. S. COSMADAKIS, H. GAIFMAN, P. C. KANELLAKIS, AND M. Y. VARDI, *Decidable optimization problems for database logic programs*, in Proceedings of 20th ACM Symposium on Theory of Computing, 1988, pp. 477–490. (It also appears as IBM Research Report RJ 6145 (60855), Yorktown Heights, NY, March 22, 1988.)

[CH1] E. P. F. CHAN AND H. J. HERNÁNDEZ, *On the desirability of γ-acyclic BCNF database schemes*, Theoret. Comput. Sci., 62 (1988), pp. 67–104.

[CH2] ———, *Independence-reducible database schemes*, in Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Austin, TX, 1988, pp. 163–173.

[CH3] ———, *Testing unboundedness of database schemes and functional dependencies*, Inform. Process. Lett., 28 (1988), pp. 317–326.

[CH4] ———, *On generating database schemes bounded or constant-time-maintainable by extensibility*, Acta Inform., 25 (1988), pp. 475–496.

[F] R. FAGIN, *Horn clauses and database dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 952–983.

[GM] M. H. GRAHAM AND A. O. MENDELZON, *The power of canonical queries*, unpublished manuscript, 1983.

[GMSV] H. GAIFMAN, H. MAIRSON, Y. SAGIV, AND M. Y. VARDI, *Undecidable optimization problems for database logic programs*, in Proc. of 2nd IEEE Symposium on Logic in Computer Science, Ithaca, NY, 1987, pp. 106–115. (It appears also as IBM Research Report RJ 5583 (56702), Yorktown Heights, NY, April 3, 1987.)

[GMV] M. H. GRAHAM, A. O. MENDELZON, AND M. Y. VARDI, *Notions of dependency satisfaction*, J. Assoc. Comput. Mach., 33 (1986), pp. 105–129.

[GW] M. H. GRAHAM AND K. WANG, *Constant time maintenance or the triumph of the fd*, in Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Cambridge, MA, 1986, pp. 202–216.

[GY] M. H. GRAHAM AND M. YANNAKAKIS, *Independent database schemas*, J. Comput. System Sci., 28 (1984), pp. 121–141.

[H1] P. HONEYMAN, *Extension joins*, in Proceedings of International Conference on Very Large Databases, 1980, pp. 239–244.

[H2] ———, *Testing satisfaction of functional dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 668–677.

[HC] H. J. HERNÁNDEZ AND E. P. F. CHAN, *A characterization of constant-time maintainability for BCNF database schemes*, in Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, IL, June 1988, pp. 209–217.

[I] Y. E. IOANNIDIS, *A time bound on the materialization of some recursively defined views*, in Proceedings of the Eleventh International Conference on Very Large Databases, Stockholm, Sweden, 1985, pp. 219–226.

[IIK] M. ITO, M. IWASAKI, AND T. KASAMI, *Some results on the representative instance in relational databases*, SIAM J. Comput., 14 (1985), pp. 334–354.

[M] A. O. MENDELZON, *Database states and their tableaux*, ACM Trans. Database Systems, 9 (1984), pp. 264–282.

[Ma] D. MAIER, *The Theory of Relational Databases*, Computer Science Press, Rockville, MD, 1983.

[MMS] D. MAIER, A. O. MENDELZON, AND Y. SAGIV, *Testing implications of data dependencies*, ACM Trans. Database Systems, 4 (1979), pp. 455–469.

[MRW] D. MAIER, D. ROZENSHTEIN, AND D. S. WARREN, *Windows functions*, in Advances in Computing Research, JAI Press, Greenwich, CT, 1986, Vol. 3, pp. 213–246.

[MUV] D. Maier, J. D. Ullman, and M. Y. Vardi, *On the foundations of the universal relation model*, ACM Trans. Database Systems, 9 (1984), pp. 283–308.

[NS] J. F. Naughton and Y. Sagiv, *A decidable class of bounded recursions*, in Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA, 1987, pp. 227–236.

[S1] Y. Sagiv, *Can we use the universal instance assumption without using nulls?*, in Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, 1981, pp. 108–120.

[S2] ———, *A characterization of globally consistent databases and their correct access paths*, ACM Trans. Database Systems, 8 (1983), pp. 266–286.

[S3] ———, *Evaluation of queries in independent database schemes*, J. Assoc. Comp. Mach., to appear.

[S4] ———, *On bounded database schemes and bounded horn-clause programs*, SIAM J. Comput., 17 (1988), pp. 1–22.

[U] J. D. Ullman, *Principles of Database Systems*, Computer Science Press, Rockville, MD, 1982.

[Var] M. Y. Vardi, *Decidability and undecidability results for boundedness of linear recursive queries*, in Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Austin, TX, 1988, pp. 341–351.

[Vas] Y. Vassiliou, *A formal treatment of imperfect information in data management*, CSRG TR-123, University of Toronto, Toronto, Canada, 1980.

[W] K. Wang, *Can constant-time-maintainability be more practical?* in Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1989, pp. 120–127.

[WG] K. Wang and M. Graham, *Constant-time maintainability: A generalization of independence*, submitted for publication, 1988.

[Y] M. Yannakakis, *Algorithms for acyclic database schemes*, in Proceedings of the International Conference on Very Large Databases, 1981, pp. 82–94.

[YP] M. Yannakakis and C. H. Papadimitriou, *Algebraic dependencies*, J. Comput. Systems Sci., 25 (1982), pp. 2–41.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 46–56, February 1993

TIGHT WORST-CASE PERFORMANCE BOUNDS FOR NEXT-k-FIT BIN PACKING*

WEIZHEN MAO†

Abstract. The bin packing problem is to pack a list of reals in (0, 1] into unit-capacity bins using the minimum number of bins. Let $R[A]$ be the limiting worst value for the ratio $A(L)/L^*$ as L^* goes to ∞, where $A(L)$ denotes the number of bins used in the approximation algorithm A, and L^* denotes the minimum number of bins needed to pack L. Obviously, $R[A]$ reflects the worst-case behavior of A. For Next-k-Fit(NkF for short, $k \geq 2$), which is a linear time approximation algorithm for bin packing, it was known that $1.7 + \frac{3}{10(k-1)} \leq R[NkF] \leq 2$. In this paper, a tight bound $R[NkF] = 1.7 + \frac{3}{10(k-1)}$ is proved.

Key words. bin packing, approximation algorithm, worst-case performance

AMS(MOS) subject classifications. 68Q25, 68R05

1. Introduction. Given a finite list $L = (a_1, a_2, \ldots, a_m)$ of reals in (0, 1], and a sequence of unit-capacity bins, $B_1, B_2, \ldots$, the bin packing problem is to pack the numbers in the list into the bins such that no bin contains a total exceeding 1 and that the number of bins used is minimized.

Since the bin packing is NP-complete [9], no polynomial-time algorithm has ever been developed. A lot of effort has been made to find good approximation algorithms for the problem.

In order to evaluate and compare the quality of different approximation algorithms, we need to have a rigorous mathematical analysis of the worst-case behavior of these algorithms. Given an approximation algorithm A, and for any list L, let $A(L)$ be the number of bins used in the packing resulting when A is applied to L, and L^* be the minimum number of bins needed to pack L. The *worst-case performance bound* of the approximation algorithm A is defined to be $R[A] = \limsup \max\{A(L)/L^*\}$ as $L^* \to \infty$.

Besides those well-studied approximation algorithms such as First-Fit (FF), Best-Fit (BF), First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD), and Next-Fit (NF) [1], [5], [6], [7], [8], there is another important algorithm called Next-k-Fit (NkF), where k is an integer greater than 1. In NkF, we process the numbers in L in turn, starting from a_1, which is placed at the bottom of the first bin B_1. Suppose that a_i is now to be packed. We look at the last k nonempty bins. If a_i does not fit into any of them, a new bin is created; otherwise, a_i will go to the lowest indexed one of these k nonempty bins into which it fits. Earlier, Johnson [7] proved that $1.7 + \frac{3}{10k} \leq R[NkF] \leq 2$. In the recent paper written by Csirik and Imreh [2], a new lower bound of $R[NkF]$ was given. They showed that $R[N2F] = 2$ and $1.7 + \frac{3}{10(k-1)} \leq R[NkF] \leq 2$ for $k \geq 3$. In this paper, we study the tight worst-case performance bound for the Next-k-Fit algorithm. Our result is the following theorem.

MAIN THEOREM.

$$R[NkF] = 1.7 + \frac{3}{10(k-1)}, \quad k \geq 2.$$

In §2, we study the upper bound proving technique for NkF bin packing. In §3, we prove an important lemma. In §4, we show the proof of the main theorem.

*Received by the editors May 6, 1989; accepted for publication (in revised form) October 1, 1991. This research was supported by National Science Foundation grant CCR-88-13283, and was partially done at Princeton University.

†Department of Computer Science, The College of William and Mary, Williamsburg, Virginia 23187-8795.

2. The upper bound of $R[NkF]$. It is known that $R[N2F] = 2$, and $R[NkF] \geq 1.7 + \frac{3}{10(k-1)}$ for $k \geq 3$ [2]. To prove the main theorem, all we need to do is prove the upper bound result, i.e., $R[NkF] \leq 1.7 + \frac{3}{10(k-1)}$ for $k \geq 3$. We need some careful analyses and preliminary results.

In the NkF packing of any list L, there are $NkF(L)$ nonempty bins, $B_1, B_2, \ldots, B_{NkF(L)}$. For each bin B_i, its content can be divided into k areas, $A_{i,1}, A_{i,2}, \ldots, A_{i,k}$, where $A_{i,1}$ contains all the numbers coming to B_i when B_i is the rightmost, or, in other words, the most recently created nonempty bin in the current packing, and $A_{i,2}$ contains all the numbers coming to B_i when B_i becomes the second rightmost nonempty bin, etc. Finally, $A_{i,k}$ contains all the numbers coming to B_i when B_i becomes the oldest among the k active bins and is about to be thrown away. Figure 1 shows the division for $N3F$.

To prove the upper bound, we wish to show that $NkF(L) \leq (1.7 + \frac{3}{10(k-1)})L^* + c$ for all L, where c is a constant. With the help of the following weighting function $W : (0,\ 1] \rightarrow R^+$, also shown in Fig. 2, we will find the relation between $NkF(L)$ and L^*.

$$W(\alpha) = \begin{cases} \frac{6}{5}\alpha & \text{if } \alpha \in (0,\ \frac{1}{6}]; \\ \frac{9}{5}\alpha - \frac{1}{10} & \text{if } \alpha \in (\frac{1}{6},\ \frac{1}{3}]; \\ \frac{6}{5}\alpha + \frac{1}{10} & \text{if } \alpha \in (\frac{1}{3},\ \frac{1}{2}]; \\ \frac{6}{5}\alpha + \frac{2}{5} + \frac{3}{10(k-1)} & \text{if } \alpha \in (\frac{1}{2},\ 1]. \end{cases}$$

For any number a_i in L, $W(a_i)$ is called the weight of a_i. $W(B_i)$, the weight of the bin B_i, is defined to be the sum of the weight of all numbers in B_i, i.e., $W(B_i) = \sum_{\forall a_j \in B_i} W(a_j)$. And $W(L)$, the weight of the list L, is defined to be the sum of the weight of all numbers in L, i.e., $W(L) = \sum_{\forall a_j \in L} W(a_j)$. When there is no possibility of confusion, we also use B_i to denote the sum of the numbers in bin B_i, $A_{i,h}$ the sum of the numbers in area $A_{i,h}$, and b_i the bottommost item in bin B_i.

3. A lemma.

LEMMA. *In the NkF packing of L, for $j < NkF(L)$, if $B_j < \frac{5}{6}$, then there is $l > 0$ such that either* (1) $j + l \leq NkF(L)$, *and* $\frac{6}{5}B_j + W(B_{j+1}) + \cdots + W(B_{j+l}) \geq l + \frac{6}{5}B_{j+l}$, *or* (2) $j + l = NkF(L)$, *and* $\frac{6}{5}B_j + W(B_{j+1}) + \cdots + W(B_{NkF(L)}) + 2 \geq l + \frac{6}{5}B_{NkF(L)}$.

Proof. For notational simplicity, we assume $j = 1$. Because $B_1 < \frac{5}{6}$, items in $A_{2,1}$ and $A_{3,1}$ must be greater than $\frac{1}{6}$. Consider the following cases.

Case I. If $B_1 \leq \frac{1}{2}$, then B_1 must be followed by k bins with their bottommost items greater than $\frac{1}{2}$, i.e., $b_2, \ldots, b_{k+1} > \frac{1}{2}$ (Fig. 3).

$$\begin{aligned} &\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_k) + W(B_{k+1}) \\ &\geq \tfrac{6}{5}A_{1,1} + (\tfrac{6}{5}b_2 + \tfrac{2}{5} + \tfrac{3}{10(k-1)}) + \cdots + (\tfrac{6}{5}b_k + \tfrac{2}{5} + \tfrac{3}{10(k-1)}) + \tfrac{6}{5}B_{k+1} + \tfrac{2}{5} + \tfrac{3}{10(k-1)} \\ &\geq \tfrac{6}{5}(A_{1,1} + b_2) + \tfrac{6}{5}(b_3 + \cdots + b_k) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})k + \tfrac{6}{5}B_{k+1} \\ &\geq \tfrac{6}{5} \times 1 + \tfrac{6}{5} \times \tfrac{1}{2} \times (k-2) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})k + \tfrac{6}{5}B_{k+1} \\ &\geq k + \tfrac{6}{5}B_{k+1}. \end{aligned}$$

Case II. If $\frac{1}{2} < B_1 < \frac{5}{6}$, then we consider the cases in Fig. 4.

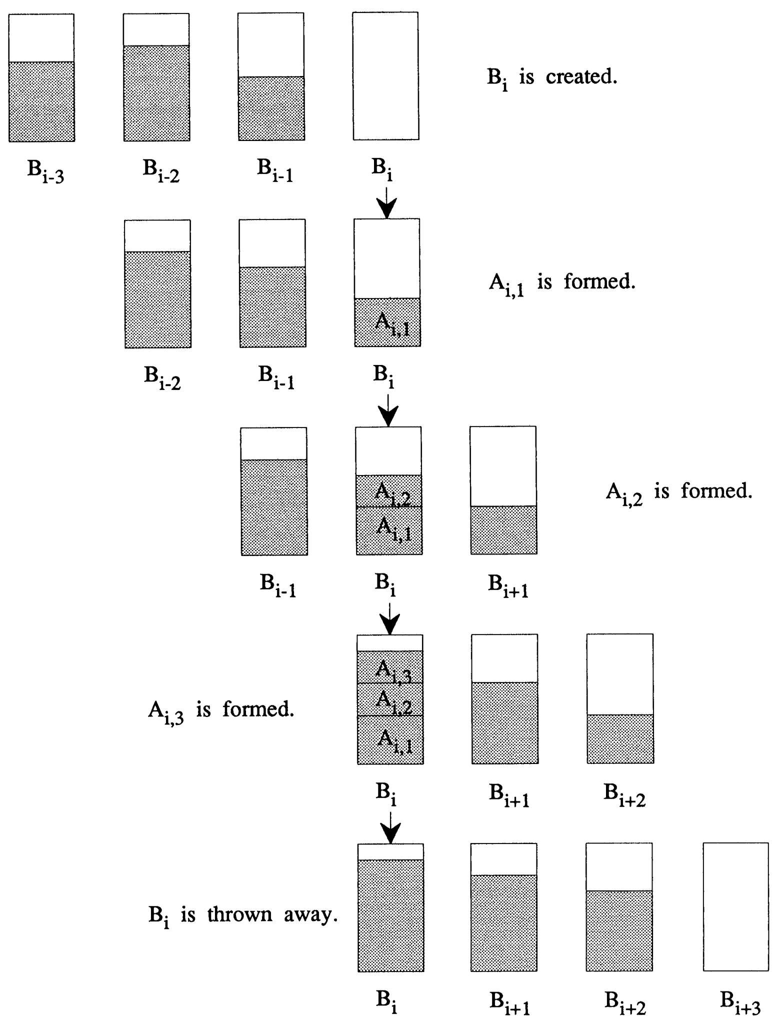


FIG. 1. *How the three areas of B_i in $N3F$ packing are formed.*

Case 1. B_2 has one item greater than $\frac{1}{2}$. We have

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2)\\
&\quad \geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{2}{5} + \tfrac{3}{10(k-1)}\\
&\quad \geq \tfrac{6}{5} \times \tfrac{1}{2} + \tfrac{2}{5} + \tfrac{6}{5}B_2\\
&\quad \geq 1 + \tfrac{6}{5}B_2.
\end{aligned}$$

Starting from now, we assume that all the items in B_2 are no greater than $\frac{1}{2}$.

Case 2. $A_{2,1} > \frac{1}{2}$. Since $A_{2,1}$ has at least two items, we assume at least one of its two bottommost items is in $(\frac{1}{6}, \frac{1}{3}]$. It is clear that B_1 is greater than $\frac{2}{3}$.

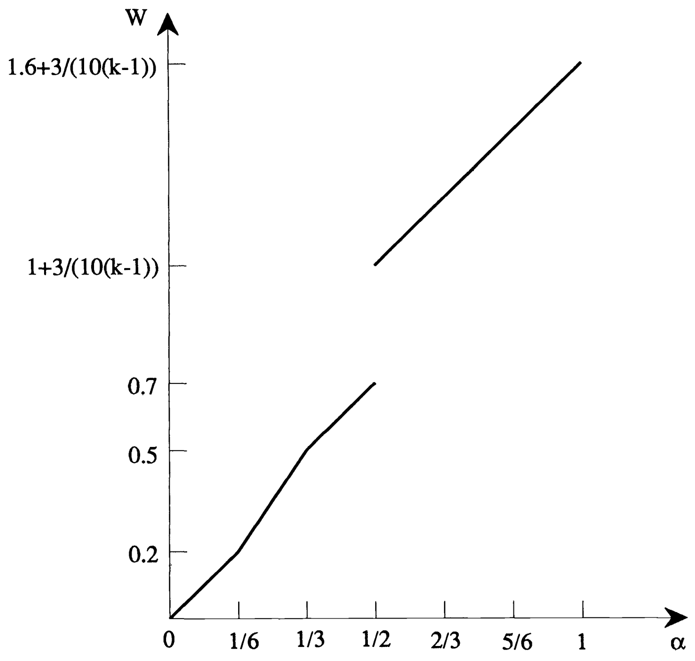


FIG. 2. *The weighting function $W(\alpha)$.*

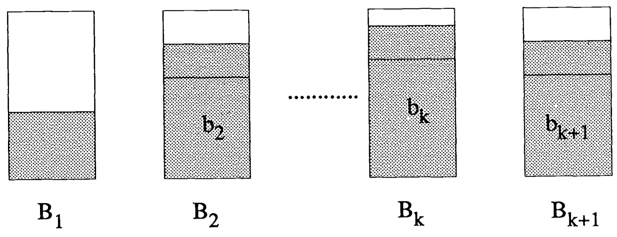


FIG. 3. *The possible packing when $B_1 \le \frac{1}{2}$.*

If the other item in $A_{2,1}$ is also in $(\frac{1}{6}, \frac{1}{3}]$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2)\\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{3}{5}(1 - B_1) - \tfrac{1}{10} + \tfrac{3}{5}(1 - B_1) - \tfrac{1}{10}\\
&\ge 1 + \tfrac{6}{5}B_2.
\end{aligned}$$

If the other item is in $(\frac{1}{3}, \frac{1}{2}]$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2)\\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{3}{5}(1 - B_1) - \tfrac{1}{10} + \tfrac{1}{10}\\
&\ge \tfrac{3}{5}B_1 + \tfrac{3}{5} + \tfrac{6}{5}B_2\\
&\ge \tfrac{3}{5} \times \tfrac{2}{3} + \tfrac{3}{5} + \tfrac{6}{5}B_2\\
&\ge 1 + \tfrac{6}{5}B_2.
\end{aligned}$$

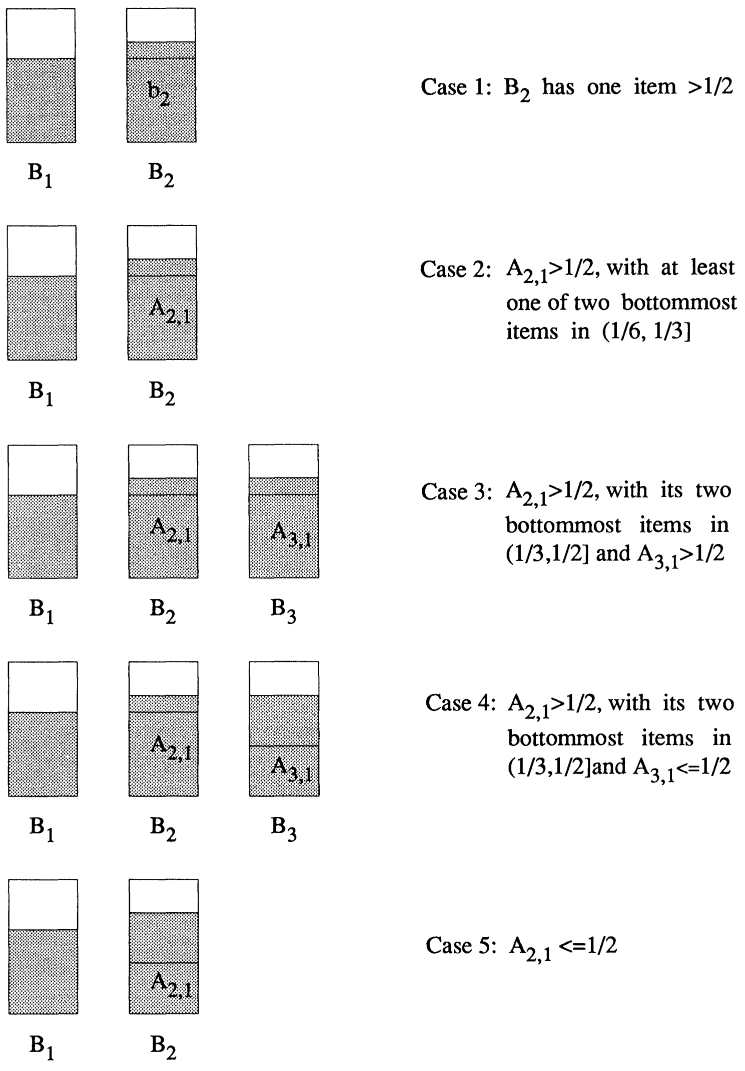

b2
B1
B2
Case 1: B2 has one item >1/2
A2,1
B1
B2
Case 2: A2,1>1/2, with at least one of two bottommost items in (1/6, 1/3]
A2,1
A3,1
B1
B2
B3
Case 3: A2,1>1/2, with its two bottommost items in (1/3,1/2] and A3,1>1/2
A2,1
A3,1
B1
B2
B3
Case 4: A2,1>1/2, with its two bottommost items in (1/3,1/2]and A3,1<=1/2
A2,1
B1
B2
Case 5: A2,1 <=1/2

FIG. 4. *The possible packings when* $\frac{1}{2} < B_1 < \frac{5}{6}$.

Case 3. $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} > \frac{1}{2}$. It is clear that $B_2 > \frac{2}{3}$.

If $A_{3,1}$ has one item greater than $\frac{1}{2}$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + W(B_3) \\
&\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{2}{5} + \tfrac{3}{10(k-1)} \\
&\geq \tfrac{6}{5} \times \tfrac{1}{2} + \tfrac{6}{5} \times \tfrac{2}{3} + \tfrac{3}{5} + \tfrac{6}{5}B_3 \\
&\geq 2 + \tfrac{6}{5}B_3.
\end{aligned}$$

If the two bottommost items of $A_{3,1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + W(B_3)\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{3}{5}(1 - B_1) - \tfrac{1}{10} + \tfrac{3}{5}(1 - B_2) - \tfrac{1}{10}\\
&\quad\geq \tfrac{3}{5}B_1 + \tfrac{3}{5}B_2 + \tfrac{6}{5} + \tfrac{6}{5}B_3\\
&\quad\geq \tfrac{3}{5}B_1 + \tfrac{3}{5}(1 - B_1 + \tfrac{1}{3}) + \tfrac{6}{5} + \tfrac{6}{5}B_3\\
&\quad\geq 2 + \tfrac{6}{5}B_3.
\end{aligned}$$

If one of the two bottommost items is in $(\frac{1}{6}, \frac{1}{3}]$, and the other is in $(\frac{1}{3}, \frac{1}{2}]$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + W(B_3)\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{3}{5}(1 - B_2) - \tfrac{1}{10} + \tfrac{1}{10}\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{3}{5}B_2 + \tfrac{4}{5} + \tfrac{6}{5}B_3\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{3}{5}(1 - B_1 + 1 - B_1) + \tfrac{4}{5} + \tfrac{6}{5}B_3\\
&\quad\geq 2 + \tfrac{6}{5}B_3.
\end{aligned}$$

If the two bottommost items are in $(\frac{1}{3}, \frac{1}{2}]$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + W(B_3)\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{1}{10} + \tfrac{1}{10}\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}(1 - B_1 + \tfrac{1}{3}) + \tfrac{2}{5} + \tfrac{6}{5}B_3\\
&\quad\geq 2 + \tfrac{6}{5}B_3.
\end{aligned}$$

Case 4. $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} \leq \frac{1}{2}$. In this case, we need to consider several possibilities according to the area distribution of B_3. In Fig. 5, on the right side of the vertical line are the three such possible packings that may follow the bins B_1 and B_2.

If $A_{3,1} + \cdots + A_{3,h} \leq \frac{1}{2}$, but $A_{3,1} + \cdots + A_{3,h+1} > \frac{1}{2}$, for $1 \leq h \leq k - 2$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{h+3})\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{6}{5}(b_4 + \cdots + b_{h+2}) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})h + \tfrac{6}{5}B_{h+3}\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{5} + \tfrac{6}{5}(A_{3,1} + A_{3,h+1}) + \tfrac{6}{5} \times \tfrac{1}{2} \times (h - 1) + \tfrac{2}{5}h + \tfrac{6}{5}B_{h+3}\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{6}{5}(1 - B_1 + 1 - B_2) + h - \tfrac{2}{5} + \tfrac{6}{5}B_{h+3}\\
&\quad\geq h + 2 + \tfrac{6}{5}B_{h+3}.
\end{aligned}$$

If $A_{3,1} + \cdots + A_{3,k-1} \leq \frac{1}{2}$, but $A_{3,1} + \cdots + A_{3,k} > \frac{1}{2}$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{k+2})\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{2}{10} + \tfrac{6}{5}B_3 + \tfrac{6}{5}(b_4 + \cdots + b_{k+1}) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})(k - 1) + \tfrac{6}{5}B_{k+2}\\
&\quad\geq \tfrac{6}{5}B_1 + \tfrac{6}{5}(1 - B_1 + \tfrac{1}{3}) + \tfrac{1}{5} + \tfrac{6}{5} \times \tfrac{1}{2} + \tfrac{6}{5} \times \tfrac{1}{2} \times (k - 2) + \tfrac{2}{5}(k - 1) + \tfrac{3}{10} + \tfrac{6}{5}B_{k+2}\\
&\quad\geq k + 1 + \tfrac{6}{5}B_{k+2}.
\end{aligned}$$

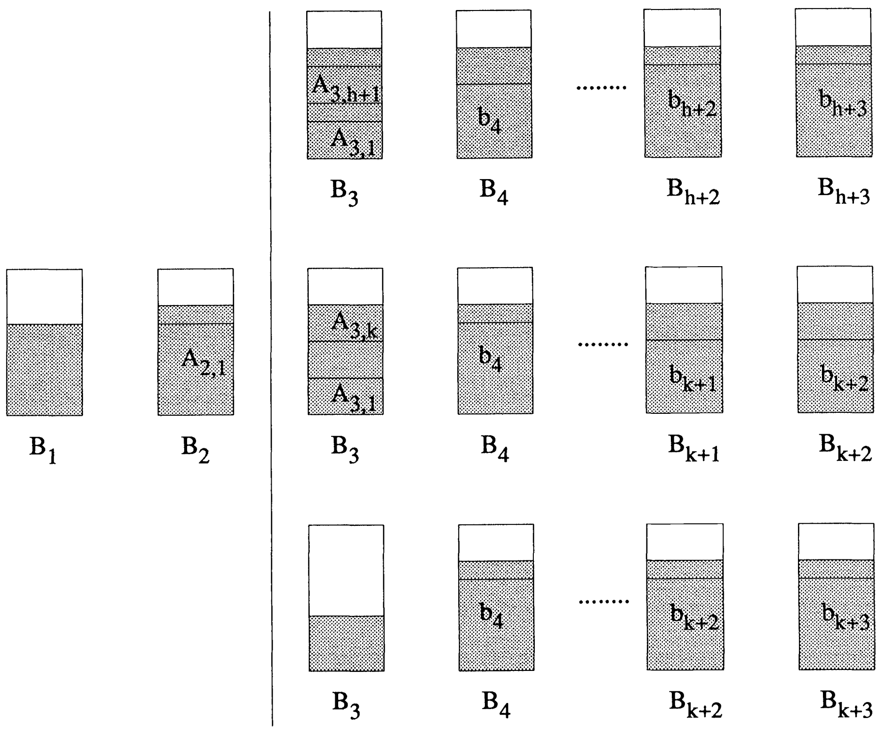


FIG. 5. *The possible packings when $A_{2,1} > \frac{1}{2}$, with its two bottommost items in $(\frac{1}{3}, \frac{1}{2}]$, and $A_{3,1} \le \frac{1}{2}$.*

If $A_{3,1} + \cdots + A_{3,k} \le \frac{1}{2}$, then

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{k+3}) \\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{1}{10} + \tfrac{1}{10} + \tfrac{6}{5}B_3 + \tfrac{6}{5}(b_4 + \cdots + b_{k+2}) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})k + \tfrac{6}{5}B_{k+3} \\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}(1 - B_1 + \tfrac{1}{3}) + \tfrac{1}{5} + \tfrac{6}{5}A_{3,1} + \tfrac{6}{5}b_4 + \tfrac{6}{5} \times \tfrac{1}{2} \times (k-2) + \tfrac{2}{5}k + \tfrac{3}{10} + \tfrac{6}{5}B_{k+3} \\
&\ge \tfrac{6}{5}(A_{3,1} + b_4) + k + \tfrac{9}{10} + \tfrac{6}{5}B_{k+3} \\
&\ge \tfrac{6}{5} \times 1 + k + \tfrac{9}{10} + \tfrac{6}{5}B_{k+3} \\
&\ge k + 2 + \tfrac{6}{5}B_{k+3}.
\end{aligned}$$

Case 5. $A_{2,1} \le \frac{1}{2}$. Let us consider the subcases in Fig. 6.

If $A_{2,1} + \cdots + A_{2,h} \le \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,h+1} > \frac{1}{2}$, where $1 \le h \le k-2$, then it is easy to prove that $W(B_2) \ge \frac{6}{5}a + \frac{2}{5}$, where a is the smallest item among $A_{2,1}, \ldots, A_{2,h+1}$. Because we know that $A_{2,1}$ and $A_{2,h+1}$ are both nonzero, so there are at least two items in these areas. Since $A_{2,1} + \cdots + A_{2,h+1} > \frac{1}{2}$, then $(A_{2,1} + \cdots + A_{2,h+1}) - a > \frac{1}{4}$. If there is one item in $A_{2,1}, \ldots, A_{2,h+1}$ in $(\frac{1}{3}, \frac{1}{2}]$, then $W(B_2) \ge \frac{6}{5}a + \frac{6}{5}((A_{2,1} + \cdots + A_{2,h+1}) - a) + \frac{1}{10} > \frac{6}{5}a + \frac{2}{5}$. Otherwise, all numbers in $A_{2,1}, \ldots, A_{2,h+1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, and there are at least two of them. If there are only two numbers in $(\frac{1}{6}, \frac{1}{3}]$, then $W(B_2) \ge \frac{6}{5}a + \frac{6}{5}((A_{2,1} + \cdots + A_{2,h+1}) - a) + \frac{3}{5}(A_{2,1} + \cdots + A_{2,h+1}) - \frac{2}{10} > \frac{6}{5}a + \frac{6}{5} \times \frac{1}{4} + \frac{3}{5} \times \frac{1}{2} - \frac{2}{10} = \frac{6}{5}a + \frac{2}{5}$. If $A_{2,1}, \ldots, A_{2,h+1}$ have at least three items in $(\frac{1}{6}, \frac{1}{3}]$, then $W(B_2) \ge \frac{6}{5}a + \frac{6}{5} \times (\frac{1}{6} + \frac{1}{6}) = \frac{6}{5}a + \frac{2}{5}$. Therefore,

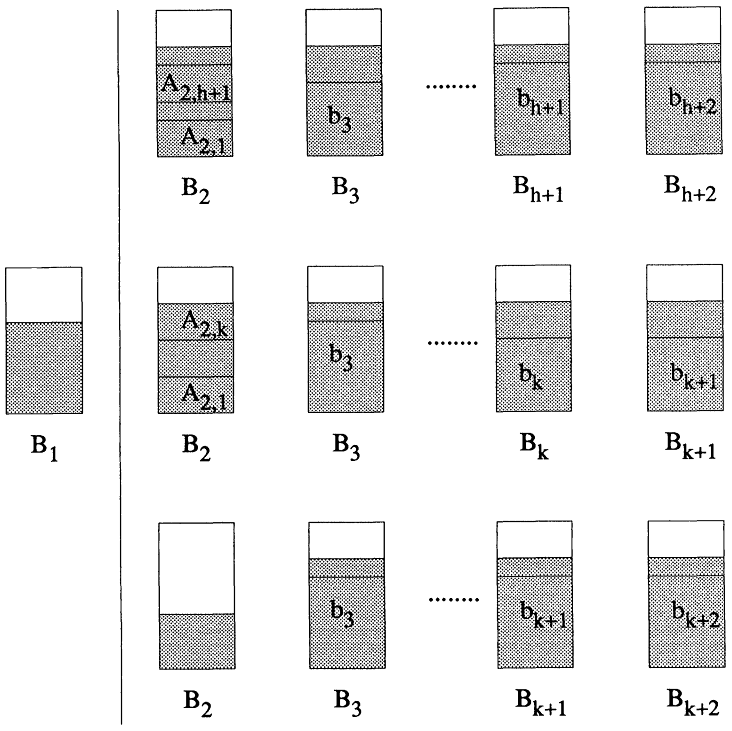


FIG. 6. *The possible packings when* $A_{2,1} \le \frac{1}{2}$.

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{h+2})\\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}a + \tfrac{2}{5} + \tfrac{6}{5}(b_3 + \cdots + b_{h+1}) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})h + \tfrac{6}{5}B_{h+2}\\
&\ge \tfrac{6}{5}(B_1 + a) + \tfrac{2}{5} + \tfrac{6}{5} \times \tfrac{1}{2} \times (h-1) + \tfrac{2}{5}h + \tfrac{6}{5}B_{h+2}\\
&\ge \tfrac{6}{5} \times 1 + \tfrac{2}{5} + \tfrac{3}{5}(h-1) + \tfrac{2}{5}h + \tfrac{6}{5}B_{h+2}\\
&\ge h + 1 + \tfrac{6}{5}B_{h+2}.
\end{aligned}$$

If $A_{2,1} + \cdots + A_{2,k-1} \le \frac{1}{2}$, but $A_{2,1} + \cdots + A_{2,k} > \frac{1}{2}$, then it is easy to prove that $W(B_2) \ge \frac{6}{5}A_{2,1} + \frac{1}{10}$. Because if $A_{2,1}$ has at least one item in $(\frac{1}{3}, \frac{1}{2}]$, then the inequality is obvious. If all the items in $A_{2,1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, then there are at most two such items in $A_{2,1}$ since $A_{2,1}$ is less than $\frac{1}{2}$. So $W(B_2) \ge \frac{9}{5}A_{2,1} - \frac{1}{10} \times 2 + \frac{6}{5}(A_{2,2} + \cdots + A_{2,k}) > \frac{6}{5}A_{2,1} - \frac{1}{5} + \frac{3}{5}B_2 > \frac{6}{5}A_{2,1} - \frac{1}{5} + \frac{3}{5} \times \frac{1}{2} = \frac{6}{5}A_{2,1} + \frac{1}{10}$. Therefore,

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{k+1})\\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}A_{2,1} + \tfrac{1}{10} + \tfrac{6}{5}(b_3 + \cdots + b_k) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})(k-1) + \tfrac{6}{5}B_{k+1}\\
&\ge \tfrac{6}{5} \times \tfrac{1}{2} + \tfrac{6}{5}(A_{2,1} + b_3) + \tfrac{1}{10} + \tfrac{6}{5} \times \tfrac{1}{2} \times (k-3) + \tfrac{2}{5}(k-1) + \tfrac{3}{10} + \tfrac{6}{5}B_{k+1}\\
&\ge \tfrac{3}{5} + \tfrac{6}{5} \times 1 + \tfrac{1}{10} + \tfrac{3}{5}(k-3) + \tfrac{2}{5}(k-1) + \tfrac{3}{10} + \tfrac{6}{5}B_{k+1}\\
&\ge k + \tfrac{6}{5}B_{k+1}.
\end{aligned}$$

If $A_{2,1} + \cdots + A_{2,k} \le \frac{1}{2}$, then it is easy to prove that $W(B_2) \ge \frac{6}{5}B_2 + \frac{3}{5}A_{2,1} - \frac{1}{5}$. Because if $A_{2,1}$ has at least one item in $(\frac{1}{3}, \frac{1}{2}]$, then $W(B_2) \ge \frac{6}{5}B_2 + \frac{1}{10} = \frac{6}{5}B_2 + \frac{3}{5} \times \frac{1}{2} - \frac{1}{5} \ge \frac{6}{5}B_2 + \frac{3}{5}A_{2,1} - \frac{1}{5}$. If all the numbers in $A_{2,1}$ are in $(\frac{1}{6}, \frac{1}{3}]$, then $W(B_2) \ge \frac{9}{5}A_{2,1} - \frac{1}{10} \times 2 + \frac{6}{5}(A_{2,2} + \cdots + A_{2,k}) = \frac{6}{5}B_2 + \frac{3}{5}A_{2,1} - \frac{1}{5}$. Therefore,

$$\begin{aligned}
&\tfrac{6}{5}B_1 + W(B_2) + \cdots + W(B_{k+2}) \\
&\ge \tfrac{6}{5}B_1 + \tfrac{6}{5}B_2 + \tfrac{3}{5}A_{2,1} - \tfrac{1}{5} + \tfrac{6}{5}(b_3 + \cdots + b_{k+1}) + (\tfrac{2}{5} + \tfrac{3}{10(k-1)})k + \tfrac{6}{5}B_{k+2} \\
&\ge \tfrac{3}{5}B_1 + \tfrac{3}{5}(B_1 + A_{2,1}) - \tfrac{1}{5} + \tfrac{6}{5}(B_2 + b_3) + \tfrac{6}{5} \times \tfrac{1}{2} \times (k-2) + \tfrac{2}{5}k + \tfrac{3}{10} + \tfrac{6}{5}B_{k+2} \\
&\ge \tfrac{3}{5} \times \tfrac{1}{2} + \tfrac{3}{5} \times 1 - \tfrac{1}{5} + \tfrac{6}{5} \times 1 + \tfrac{3}{5}(k-2) + \tfrac{2}{5}k + \tfrac{3}{10} + \tfrac{6}{5}B_{k+2} \\
&\ge k + 1 + \tfrac{6}{5}B_{k+2}.
\end{aligned}$$

This ends the case analysis. If beginning with B_j (B_1 in the case analysis) there is a portion of the NkF packing which matches one of the above cases, and if we let l be the index of the last bin in that portion minus j, then $j + l \le NkF(L)$, and $\frac{6}{5}B_j + W(B_{j+1}) + \cdots + W(B_{j+l}) \ge l + \frac{6}{5}B_{j+l}$, which satisfies (1) in the Lemma. However, if the NkF packing of the list L ends without completely matching any of the above cases, i.e., $B_j, \ldots, B_{NkF(L)}$ only matches the first part of one of the cases, then we can see that no matter where the packing ends B_j is followed by $h(\ge 0)$ bins with $\frac{6}{5}B_j + W(B_{j+1}) + \cdots + W(B_{j+h}) \ge h$, then followed by $g(\ge 0)$ bins with items greater than $\frac{1}{2}$, hence each having weight greater than 1. If we let l be the index of the last bin in the packing minus j, i.e., $NkF(L) - j$, then $j + l = NkF(L)$, and $\frac{6}{5}B_j + W(B_{j+1}) + \cdots + W(B_{NkF(L)}) + 2 \ge (NkF(L) - j) + 2 \ge l + \frac{6}{5}B_{NkF(L)}$, which satisfies (2) in the Lemma.

4. Proof of the main theorem.

CLAIM 1. *For any bin B_i of items of total size 1 or less*,

$$W(B_i) \le 1.7 + \frac{3}{10(k-1)}.$$

Proof. See the proof of Lemma 1 in the work of Garey, Graham, Johnson, and Yao [4]. We note that our weighting function differs from that in the reference only by the addition of $\frac{3}{10(k-1)}$ for the items of size exceeding $\frac{1}{2}$, and there can be only one such item in B_i. So the bound in the claim exceeds the bound 1.7 in the reference by precisely this amount. □

CLAIM 2. *For any list L,*

$$W(L) \le \left(1.7 + \frac{3}{10(k-1)}\right) L^*.$$

Proof. Apply the optimal algorithm to L. We get L^* nonempty bins.

$$\begin{aligned}
W(L) &= \textstyle\sum_{i=1}^{L^*} W(B_i) \\
&\le \textstyle\sum_{i=1}^{L^*} (1.7 + \frac{3}{10(k-1)}) \qquad \text{(by Claim 1)} \\
&= (1.7 + \tfrac{3}{10(k-1)})L^*.
\end{aligned}$$
□

CLAIM 3. *For any list L, there exists a constant c such that*

$$W(L) + c \ge NkF(L).$$

Proof. Let j be the largest index of the bins in the NkF packing such that $\sum_{i=1}^{j} W(B_i) \geq j - 1 + \frac{6}{5}B_j$. Such j always exists.

If $j = NkF(L)$, then $W(L) = \sum_{i=1}^{j} W(B_i) \geq j - 1 + \frac{6}{5}B_j \geq NkF(L) - 1$. So $W(L) + 1 \geq NkF(L)$. Now assume $j < NkF(L)$. Let us consider B_j.

If $B_j \geq \frac{5}{6}$, then $\sum_{i=1}^{j} W(B_i) + W(B_{j+1}) \geq j - 1 + \frac{6}{5}B_j + \frac{6}{5}B_{j+1} \geq j + \frac{6}{5}B_{j+1}$. There exists $j + 1$, such that $\sum_{i=1}^{j+1} W(B_i) \geq j + \frac{6}{5}B_{j+1}$. This is a contradiction to the assumption that j is the largest index having the property. So the case of $B_j \geq \frac{5}{6}$ can never happen.

If $B_j < \frac{5}{6}$, and (1) in Lemma happens, then $\sum_{i=1}^{j} W(B_i) + W(B_{j+1}) + \cdots + W(B_{j+l}) \geq j - 1 + \frac{6}{5}B_j + l + \frac{6}{5}B_{j+l} - \frac{6}{5}B_j$. Therefore, $\sum_{i=1}^{j+l} W(B_i) \geq j + l - 1 + \frac{6}{5}B_{j+l}$. This is again a contradiction to the assumption that j is the largest index. So (1) in Lemma can never happen.

If $B_j < \frac{5}{6}$, and (2) in Lemma happens, then we have $\sum_{i=1}^{j} W(B_i) + W(B_{j+1}) + \cdots + W(B_{NkF(L)}) + 2 \geq j - 1 + \frac{6}{5}B_j + NkF(L) - j + \frac{6}{5}B_{NkF(L)} - \frac{6}{5}B_j$. So $W(L) + 3 \geq NkF(L)$. $\square$

Now we are prepared to prove Main Theorem.

Proof of Main Theorem.

$$\begin{aligned} R[NkF] &= \limsup \max\{NkF(L)/L^*\} \\ &\leq \lim_{L^* \to \infty}(W(L) + c)/L^* \quad \text{(by Claim 3)} \\ &\leq \lim_{L^* \to \infty}((1.7 + \tfrac{3}{10(k-1)})L^* + c)/L^* \quad \text{(by Claim 2)} \\ &= 1.7 + \tfrac{3}{10(k-1)}. \end{aligned}$$

Combining with the previous results $R[N2F] = 2$ and $R[NkF] \geq 1.7 + \frac{3}{10(k-1)}$, we have $R[NkF] = 1.7 + \frac{3}{10(k-1)}$ for $k \geq 2$.

Acknowledgment. The author wishes to thank Professor Andrew C. Yao, without whose valuable advice, constructive comments, and enthusiastic encouragement the work of this paper could never have been accomplished. And also, many thanks go to Dr. David S. Johnson, who read the first version of the paper carefully, pointed out one mistake in the proof of the lemma, and suggested the current proving style, which significantly simplifies the entire proof.

REFERENCES

[1] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing—an updated survey*, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini, and P. Serafini, eds., Springer-Verlag, Berlin, New York, 1984, pp. 49–106.

[2] J. Csirik and B. Imreh, *On the worst-case performance of the NkF bin-packing heuristic*, Acta Cybernet., 9 (1989), pp. 89–105.

[3] J. Csirik and D. S. Johnson, *Bounded space on-line bin packing: Best is better than first*, Proc. 2nd Ann. ACM-SIAM Symposium on Discrete Algorithms, 1991, pp. 309–319.

[4] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, *Resource constrained scheduling as generalized bin packing*, J. Combin. Theory, 21 (1976), pp. 257–298.

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

[6] D. S. Johnson, *Near-optimal bin packing algorithms*, Ph.D. thesis, Tech. Report Mac TR-109, Massachusetts Institute of Technology, Cambridge, MA, 1973.

[7] ———, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 274–314.

[8] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp. 229–325.

[9] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[10] W. Mao, *Scheduling and Bin Packing: A Study of the Worst-Case Performance Bounds*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1990.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 57–61, February 1993

A NOTE ON THE COMPLEXITY OF A SIMPLE TRANSPORTATION PROBLEM*

GREG N. FREDERICKSON†

Abstract. Consider the problem of using a vehicle to transport k objects one at a time between s stations on a circular track. Let the cost of the transportation be the total distance traveled by the vehicle on the track. An $O(k+M(s,q))$ time algorithm is presented to find a minimum cost transportation, where $M(m,n)$ is the time to solve a minimum spanning tree problem on a graph with m edges and n vertices, and $q \leq \min\{k,s\}$ is the number of strongly connected components in an associated balanced problem. Also, the minimum spanning tree problem on a graph with m edges and n vertices is reduced to a transportation problem on a linear track with $O(m)$ stations, $O(m)$ objects, and $O(n)$ strongly connected components in $O(m)$ time.

Key words. transportation problems, robot arm motion, circular track, graph augmentation

AMS(MOS) subject classification. 68Q25

1. Introduction. Consider an undirected weighted graph with objects located at various vertices. Associated with each object is a destination vertex, to which that object is to be moved by a vehicle that traverses the edges of the graph. A fundamental problem in motion planning is to determine a minimum cost tour of the vehicle that transports all objects from their initial positions to their destinations. In the case of general graphs, the problem is NP-hard, even if the vehicle can transport only one object at a time [FHK]. Recently, attention has focused on solving this motion planning problem on very simple classes of graphs with unit capacity vehicles. Such examples have potential applications in robotics. Atallah and Kosaraju consider graphs that are simple paths and simple cycles [AK]. Frederickson and Guan consider graphs that are trees [FG1], [FG2], [FG3]. These papers distinguish between two cases, based on whether or not drops are allowed in the transportation. A *drop* is an unloading of an object at a vertex that is not its destination. If an object is dropped, its move is not immediately completed, and the object must be picked up and transported farther at some later time in the transportation.

In this note we tighten the bound of [AK] for the case of simple cycles with no drops. Whereas all possible solutions are considered in a divide-and-conquer search that is used in [AK], we quickly prune away all but a constant number of possible solutions. We also show that the asymptotic complexity of the transportation problem with no drops for either a simple path or a simple cycle is essentially the same as that of the minimum spanning tree problem (on a general graph).

For simplicity, we shall use much of the notation in [AK]. In particular, we shall refer to the underlying graph as a track, and the vertices in the graph as stations. Let k be the number of objects; s, the number of stations; and q, the number of strongly connected components once additional moves are added to yield a certain "balanced" problem. Let $M(m,n)$ be the time to solve the minimum spanning tree problem on a graph with n vertices and m edges. (Currently, the fastest known algorithm in a standard comparison-based model for finding the minimum spanning tree problem is given in [GGST], providing an upper bound on $M(m,n)$ of $O(m \log \beta(m,n))$, where $\beta(\cdot,\cdot)$ is a slowly growing function similar to the $\log^*(\cdot)$ function. In the trans-dichotomous model of Fredman and Willard, an upper bound of $O(m)$ is achieved for $M(m,n)$ in [FW].) For the case in which drops are allowed, an $O(k+s)$ time algorithm is given in [AK] for the

*Received by the editors September 30, 1988; accepted for publication (in revised form) December 2, 1991. This research was supported in part by the National Science Foundation under grant CCR-8620271 and by the Office of Naval Research under contract N00014-86-K-0689.

†Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907.

circular track and hence the linear track. For the case in which no drops are allowed, an $O(k + s \log s)$ time algorithm is given in [AK] for a circular track and an $O(k + M(s, q))$ time algorithm for a linear track (where the time is as indicated in the note added in proof in [AK]).

We make several observations about the structure of the transportation problem on a circular track with no drops that allows us to generate an $O(k + M(s, q))$ time algorithm. We also provide a simple argument that solving a transportation problem on a linear track where $k = s$ is in general no easier than solving a minimum spanning tree problem on q vertices and s edges. Thus in the sense of asymptotic complexity, the circular track problem with no drops is no harder than the linear track problem with no drops. Thus for both the case of drops and the case of no drops, restricting the graph from a circular track to a linear track will make the problem no easier.

2. A faster algorithm. In this section we derive several observations that lead to a faster algorithm. We first recall some definitions from [AK]. Let the stations be indexed from 1 to s, and the edge between stations i and $i + 1$ be denoted as interval $(i, i + 1)$. Assume that each object moves in the shorter of the two directions for it, either clockwise or counterclockwise around the cycle. Let $\phi(i)$ be the *input flux* across interval $(i, i + 1)$, defined as the number of clockwise moves in the input minus the number of counterclockwise moves in the input across interval $(i, i + 1)$.

Recall from [AK] that in any transportation, the number of clockwise moves across an interval minus the number of counterclockwise moves across an interval will be the same for all intervals. More generally, for any set of moves in which the difference across every interval is the same, this common difference is called the *flux*. By adding moves in which the vehicle carries no object, any particular value of flux can be achieved. Let ψ be the value of the flux for some set of moves. Let l_i be the length of interval $(i, i + 1)$. Let $db(\psi)$ be the cost of a minimum cost set of augmenting moves that yield a flux of ψ. The function $db(\cdot)$ represents the cost to achieve degree balance between incoming and outgoing moves at each station. Then $db(\psi) = \sum_{i=1}^{n} |(\psi - \phi(i))|\, l_i$. Note that adding moves with the "empty object" to achieve degree balance may result in more than one strongly connected component, where each component is Eulerian, and each is isolated from the others. Among all minimum-cost sets of augmenting moves that yield a flux of ψ, let q_ψ be the minimum number of strongly connected components that result from any of these augmentations. There is a minimum-cost set of augmenting moves that achieves flux ψ, creates q_ψ strongly connected components, and is of cardinality $O(k + s)$, and such a set can be found in $O(k + s)$ time. Besides the augmenting moves that achieve a particular flux, additional moves with no object are in general necessary to achieve connectivity among the components. With these additional moves a transportation can then be constructed.

We introduce some additional notation. Let c_ψ be the total length of intervals with $\phi(i) = \psi$. Then $c_\psi = \sum_{\phi(i)=\psi} l_i$. Note that $c_\psi \geq 0$ for all ψ. Let $tc(\psi)$ be the cost of a minimum cost set of augmenting moves that yield a transportation with flux ψ. The function $tc(\cdot)$ represents the *total cost* to achieve both degree balance and connectivity. Let γ be the largest value of flux for which a set of augmenting moves of overall minimum cost achieves degree balance. We note some simple relationships among these quantities in the following lemmas.

LEMMA 1. *For all values ψ of flux, $tc(\psi) \leq db(\psi) + 2c_\psi$, which holds with strict inequality if $db(\psi) < tc(\psi)$.*

Proof. The addition of the augmenting moves that achieve degree balance will leave one or more strongly connected components. If there is more than one such component, then the components are separated by intervals $(i, i+1)$ with $\phi(i) = \psi$. Adding a move in each direction across all but one such interval will yield one strongly connected component. □

LEMMA 2. *For all values ψ of flux, $db(\psi) = db(\psi - 1) - \sum_{j>\psi-1} c_j + \sum_{j\leq\psi-1} c_j$.*

Proof. Given a set of moves that achieve degree balance for flux $\psi - 1$, one can generate a set of moves that achieve degree balance for flux ψ by removing an augmenting move from each interval i for which $\phi(i) > \psi - 1$ and by adding an augmenting move for each interval i for which $\phi(i) \leq \psi - 1$. □

LEMMA 3. *The value γ satisfies the following*:

(i) $\sum_{j>\gamma-1} c_j \geq \sum_{j\leq\gamma-1} c_j$;

(ii) $\sum_{j\leq\gamma} c_j > \sum_{j>\gamma} c_j$.

Proof. Part (i) follows directly from Lemma 2 and the definition of γ. Part (ii) follows by combining the definition of γ with

$$db(\gamma + 1) = db(\gamma) - \sum_{j>\gamma} c_j + \sum_{j\leq\gamma} c_j,$$

which is obtained from Lemma 2. □

Next we note that $db(\psi)$ is a concave (upwards) function of ψ.

LEMMA 4. *The function $db(\cdot)$ is concave.*

Proof. A simple proof by induction on n establishes that functions of the form $\sum_{i=1}^{n} |a_i x - b_i|$ are concave. Note that $db(\cdot)$ is of this form. □

Let δ be the largest value of flux for which a minimum cost transportation is achieved. Then $tc(\delta) = \min_\psi\{tc(\psi)\}$. We next show that at most three values of ψ need to be considered.

LEMMA 5. *A value of ψ in the range $\gamma - 1 \leq \psi \leq \gamma + 1$ achieves the minimum value of $tc(\cdot)$.*

Proof. Suppose $\delta \geq \gamma + 2$. Then

$$\begin{aligned} db(\delta) &\leq tc(\delta - 1), \\ &\leq db(\delta - 1) + 2c_{\delta-1} \text{ (by Lemma 1)} \\ &= db(\delta) + \sum_{j>\delta-1} c_j - \sum_{j\leq\delta-1} c_j + 2c_{\delta-1} \\ &= db(\delta) + \left(\sum_{j>\gamma} c_j - \sum_{j\leq\gamma} c_j\right) - 2\sum_{j=\gamma+1}^{\delta-2} c_j. \end{aligned}$$

By the nonnegativity of c_ψ, $c_j \geq 0$ for $\gamma+1 \leq j \leq \delta-2$. But then $\sum_{j\leq\gamma} c_j - \sum_{j>\gamma} c_j \leq 0$, which contradicts Lemma 3(ii). Thus $\delta < \gamma + 2$.

Suppose $\delta \leq \gamma - 2$ and $db(\delta + 1) = tc(\delta + 1)$. Then by Lemma 4

$$db(\delta) \geq db(\delta + 1) = tc(\delta + 1).$$

Thus the cost of a solution with flux $\delta + 1$ is always at least as good as a solution with flux δ, a contradiction to the choice of δ.

Suppose $\delta \leq \gamma - 2$ and $db(\delta+1) < tc(\delta+1)$. Then

$$\begin{aligned} db(\delta) &\leq tc(\delta+1), \\ &< db(\delta+1) + 2c_{\delta+1} \text{ (by Lemma 1)} \\ &= db(\delta) - \sum_{j>\delta} c_j + \sum_{j\leq\delta} c_j + 2c_{\delta+1} \\ &= db(\delta) + \left(\sum_{j\leq\gamma-1} c_j - \sum_{j>\gamma-1} c_j\right) - 2\sum_{j=\delta+2}^{\gamma-1} c_j. \end{aligned}$$

By the nonnegativity of c_ψ, $c_j \geq 0$ for $\delta+2 \leq j \leq \gamma-1$. But then $\sum_{j\leq\gamma-1} c_j - \sum_{j>\gamma-1} c_j > 0$, which contradicts Lemma 3(i). From this and the preceding case, we may conclude that $\delta > \gamma - 2$.

Thus the above cases rule out all values ψ of flux except those in the range $\gamma - 1 \leq \psi \leq \gamma + 1$. □

The algorithm to solve the transportation problem is the following. First, compute the values of $db(\psi)$ for values of ψ from $-k$ to k, as discussed in [AK]. Next perform a scan of the values $db(\psi)$ to identify γ. Then for each of the values $\gamma-1$, γ, $\gamma+1$ of flux, solve the associated minimum spanning tree problem, using the fastest currently known algorithm. Choose from among the three transportations the solution that is of smallest total cost. Let q be $\max\{q_{\gamma-1}, q_\gamma, q_{\gamma+1}\}$.

THEOREM 1. *The time to solve a transportation problem with no drops on a graph that is a simple cycle is $O(k + M(s,q))$, where k is the number of objects, s is the number of stations, q is the maximum of number of strongly connected components in three related balanced problems, and $M(m,n)$ is the time to solve the minimum spanning tree problem on a graph with n vertices and m edges.*

Proof. By the discussion in [AK], the time to find γ is $O(k)$. The three minimum spanning tree problems can each be set up in $O(s)$ time. Since $M(s,\cdot)$ is $\Omega(s)$, the result follows. □

3. A reduction from the minimum spanning tree problem. We show how to reduce the minimum spanning tree problem to a transportation problem with no drops on a linear track. Recall from [AK] that the degree balanced version of the problem is one in which a minimum cost set of balancing moves has been added so that for any interval $(i, i+1)$ on the track, the number of moves across the interval in the clockwise direction equals the number of moves across the interval in the counterclockwise direction. Let k be the number of moves and s the number of stations. Among all such minimum-cost sets of augmenting moves, let q be the minimum number of strongly connected components that result from any of these augmentations. There is a minimum-cost set of augmenting moves that creates q strongly connected components and is of cardinality $O(k+s)$, and such a set can be found in $O(k+s)$ time.

THEOREM 2. *Let $R(k,s,q)$ be the time to solve a transportation problem with no drops on a linear track of s stations, with k moves and q components in the balanced problem. The time to find a minimum spanning tree in a graph of m edges and n vertices is $O(R(m,m,n))$.*

Proof. Let $G = (V,E)$ be a connected weighted undirected graph with m edges and $n > 3$ vertices. Without loss of generality, assume all edge weights are positive. Let W

be the largest of the edge weights. Compute the degree of every vertex. For any vertex of degree less than 3, add edges of cost $W+1$ to G to make every vertex be of degree at least 3. Once these edges have been added, if not all vertices are of even degree, introduce a new vertex with edges of cost $W+1$ to each vertex of odd degree. The resulting graph G' will have n' vertices, $n \leq n' \leq n+1$, and m' edges, $m \leq m' \leq m+3n$, and the degree of each vertex will be even and at least 4. By the choice of the cost of new edges, a minimum spanning tree of G' will be a minimum spanning tree of G, plus some edge to the new vertex if a new vertex was introduced. Given graph G', find an Euler tour of G', starting at any vertex. We denote the tour by the sequence of vertices and edges v_0, $e_1, v_1, e_2, v_2, \ldots, v'_m, e'_m, v_0$.

Given the Euler tour, we generate an instance P of a transportation problem on a linear track as follows. There will be $m'+1$ stations, one for each visit of a vertex in the Euler tour. The edge from the jth to the $(j+1)$st station will correspond to the $(j+1)$st edge in the Euler tour and thus be of cost $c(e_{j+1})$. There will be one object originating at each station. The destinations of the objects are determined as follows. Consider the rth station, and suppose it corresponds to a visit to vertex v in the Euler tour. Let the r'th station correspond to the next visit to vertex v in the Euler tour. (If there is no next visit to v, let the r'th station correspond to the first visit to v.) Then the destination of the object at station r is station r'. Since the degree of each vertex in G' is at least 4, every object will have a destination different from its originating station.

It is clear from the construction that for any vertex in G' the set of arcs in P form a cycle. It follows that there are n' strongly connected components in P. For each edge (v, w) in G' there is an edge of the same cost in the track from a station in the cycle of stations corresponding to v to a station in the cycle of stations corresponding to w.

Consider an optimal transportation Q for P. Consider the set of edges traversed by Q when no object is being carried. Any edge traversed in one direction is traversed in the other direction. The number of such edges is $n'-1$, and these edges correspond to a minimum spanning tree of G'. Thus we can solve a minimum spanning tree problem by generating an instance P, finding an optimal transportation Q for P, and extracting the edges of the minimum spanning tree from Q. Clearly, all steps other than that of finding the transportation will take $O(m')$ time. Thus the minimum spanning tree problem in G' can be solved in time $O(m' + R(m'+1, m'+1, n'))$. Assuming the monotonicity of $R(\cdot,\cdot,\cdot)$, and noting that $R(m,\cdot,\cdot)$ is $\Omega(m)$, this is $O(R(m,m,n))$. □

Acknowledgment. I would like to thank Mike Atallah for some helpful comments.

REFERENCES

[AK] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with application to minimizing robot arm travel*, SIAM J. Comput., 17 (1988), pp. 849–869.

[FG1] G. N. Frederickson and D.-J. Guan, *Ensemble motion planning in trees*, in Proc. 30th IEEE Symp. on Foundations of Computer Science, Research Triangle Park, NC, October 1989, pp. 66–71.

[FG2] ———, *Preemptive ensemble motion planning on a tree*, SIAM J. Comput., 21 (1992), pp. 1130–1152.

[FG3] ———, *Nonpreemptive ensemble motion planning on a tree*, J. Algorithms, to appear.

[FHK] G. N. Frederickson, M. S. Hecht, and C. E. Kim, *Approximation algorithms for some routing problems*, SIAM J. Comput., 7 (1978), pp. 178–193.

[FW] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, in Proc. 31st IEEE Symp. on Foundations of Computer Science, St. Louis, MO, October 1990, pp. 719–725.

[GGST] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan, *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*, Combinatorica, 6 (1986), pp. 109–122.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 62–71, February 1993

A LOWER BOUND ON THE SIZE OF SHELLSORT SORTING NETWORKS*

ROBERT CYPHER†

Abstract. Shellsort is a sorting algorithm that is based on a set of parameters called *increments*. Shellsort has been used both as a sequential sorting algorithm and as a sorting network. The central result of this paper is that all Shellsort sorting networks based on monotonically decreasing increments require $\Omega(N \log^2 N/ \log \log N)$ comparators. Previously, only the trivial $\Omega(N \log N)$ bound was known for this class of networks. The lower bound obtained in this paper nearly matches the upper bound of $O(N \log^2 N)$ that was proven by Pratt.

Key words. Shellsort, sorting networks, parallel sorting, lower bounds

AMS(MOS) subject classifications. 68P10, 68Q20, 68Q25

1. Introduction. Shellsort is a sorting algorithm that was first proposed by D. L. Shell in 1959 [11]. Shellsort is based on a sequence of integers $Z = z_k, z_{k-1}, \cdots, z_1$ that is called an *increment sequence*. Different increment sequences yield sorting algorithms with different complexities, so a great deal of effort has been devoted to finding the best increment sequence possible.

Given an array of items $A = A[1], A[2], \cdots, A[N]$ and an increment sequence $Z = z_k, z_{k-1}, \cdots, z_1$, Shellsort sorts the array A by performing z_j-sorts for $j = k, k-1, \cdots, 1$. A z_j-sort consists of partitioning the locations in the array A into equivalence classes modulo z_j and sorting the data in each equivalence class. Therefore, following the z_j-sort, for all i where $z_j < i \leq N$, $A[i] \geq A[i - z_j]$. It is required that $z_1 = 1$, so the z_1-sort completely sorts the file. In general, the increment sequence used for sorting N items can depend on N. However, one common technique is to define a single infinite sequence and to use, in decreasing order, those elements in this sequence that are less than N. Increment sequences that are created in this manner are said to be *uniform*.

The z_j-sorts are implemented by using *insertion sort* [5]. Insertion sort sorts a set of items by processing the items from left to right. Each item w is processed by comparing it to the items to its left until an item v is found that is smaller than w (or the list of items to the left of w has been exhausted). Item w is then inserted immediately to the right of v, and the items that were between v and w are moved one position to the right. Thus for any item w, if there are c items to the left of w that are larger than w, then $c + 1$ comparisons are required. Although insertion sort is an inefficient sorting algorithm (it requires $\Theta(N^2)$ time to sort N items in the worst case), it is used because it performs well when the data are nearly sorted. It is hoped that the z_j-sorts will be efficient because the sorts performed by the earlier increments will have left each item near its correct, sorted position.

Shellsort has been studied both as a sequential sorting algorithm and as a technique for creating sorting networks. A *sorting network* is a collection of comparators that are wired together in such a way that when a set of items is placed on the input wires, the items appear in sorted order on the output wires. A *comparator* is a device that takes two inputs and produces two outputs; the smaller input is placed on the first output and the larger input is placed on the second output. Shellsort can be used to create a sorting network by implementing each of the insertion sorts with a sorting network [5]. An example of a sorting network that performs an insertion sort is given in Fig. 1.

*Received by the editors May 14, 1990; accepted for publication (in revised form) December 3, 1991. This research was supported in part by a National Science Foundation Graduate Fellowship.

†Department K54/802, IBM Research Division, Almaden Research Center, 650 Harry Rd., San Jose, California 95120.

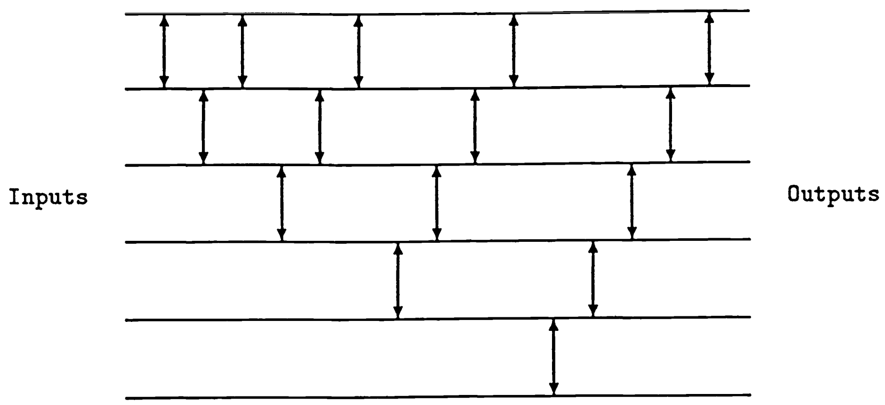


FIG. 1. *Insertion sort network. Horizontal lines represent wires, and vertical lines represent comparators. Each comparator places the smaller of its inputs on its upper output wire and the larger of its inputs on the lower output wire.*

The *size* of a sorting network is the number of comparators that it contains, and the *depth* of a sorting network is the maximum number of comparators through which an item may pass. Sorting networks are important because their nonadaptive nature allows them to be implemented directly in hardware and also because of a result due to Leighton [7]. Leighton has shown that any small-depth sorting network can be used to create a fast sorting algorithm for a bounded-degree parallel computer.

An $O(\log N)$-depth sorting network was created by Ajtai, Komlós, and Szemerédi [1]. Their sorting network is very complex, and it has an extremely large constant of proportionality associated with the $O(\log N)$ depth. A number of researchers have suggested that Shellsort might provide a simple $O(\log N)$-depth sorting network that would be practical for realistic values of N [4], [10]. This paper proves that a fundamentally new type of increment sequence will be needed if this goal is to be attained.

Two well-known results are helpful in understanding the behavior of Shellsort. First, Knuth proved that if an array is u-sorted and then it is v-sorted, it will remain u-sorted [2]. Second, from transitivity it follows that if an array is both u-sorted and v-sorted, then it is also $(u+v)$-sorted [9]. These two results suggest the use of a data structure that Pratt calls a *template* [9]. A template is a set of natural numbers that contains 0 and is closed under addition. Let $Z = z_k, z_{k-1}, \cdots, z_1$ be a sequence of increments, and let Y be the smallest template containing Z. If an array has been z_j-sorted for all j, $k \geq j \geq 1$, then it follows from the two results given above that the array is also y-sorted for each $y \in Y$. Thus templates provide a means of keeping track of all of the properties implied by the above results. Templates have been used to obtain upper bounds on the time requirements of Shellsort algorithms and on the size and depth of Shellsort sorting networks. This paper shows that templates can also be used to obtain lower bounds on the size of Shellsort sorting networks.

Shell used the increments $\lfloor N/2 \rfloor, \lfloor N/4 \rfloor, \lfloor N/8 \rfloor, \cdots, 1$. However, when N is a power of 2 this sequence requires $\Theta(N^2)$ comparisons. Modifications to Shell's increment sequence were suggested by Lazarus and Frank [6], Hibbard [3], and Knuth [5]. Papernov and Stasevich showed that Hibbard's increment sequence yields a sequential algorithm that runs in $O(N^{3/2})$ time [8]. All of the above modifications to Shell's increment sequence have the property that they are within an additive constant of a geometric sequence. Pratt proved that a large class of such nearly geometric sequences result in $\Theta(N^{3/2})$-time sequential algorithms [9]. Pratt also proposed using the uniform increment sequence consisting of all numbers of the form 2^i3^j, where i and j are integers, and he showed that these increments yield a sorting network with size $O(N \log^2 N)$ and depth $O(\log^2 N)$ [9].

Sedgewick then proposed a uniform increment sequence that yields an $O(N^{4/3})$-time sequential algorithm [10]. Although this is greater than the $O(N \log^2 N)$ sequential time required by Pratt's sequence, Sedgewick's sequence has $O(\log N)$ increments, whereas Pratt's sequence has $\Theta(\log^2 N)$ increments. The restriction to using $O(\log N)$ increments is required if an $O(N \log N)$-time algorithm is to be obtained. Incerpi then developed a uniform increment sequence with $O(\log N)$ increments that required $O\left(N^{1+\epsilon/\sqrt{\log N}}\right)$ comparisons for any $\epsilon > 0$ [4]. Finally, Weiss and Sedgewick proved that Sedgewick's sequence does require $\Omega(N^{4/3})$ comparisons in the worst case [12], [13]. In addition, Weiss and Sedgewick made a conjecture that, if true, would imply that $\Omega\left(N^{1+\epsilon/\sqrt{\log N}}\right)$ time is required by Incerpi's sequence and by any sequence $Z = z_k, z_{k-1}, \cdots, z_1$ for which $z_j = \Theta(\alpha^j)$ for some $\alpha > 1$ [12], [13].

All of the increment sequences mentioned above are monotonically decreasing. In fact, Knuth has called Shellsort the "diminishing-increment sort" [5]. Monotonically decreasing increments are natural because they sort larger sets of data as the data become increasingly ordered. Monotonically decreasing increments thus provide a smooth transition from sorting small, very unordered sets to sorting large, very ordered sets. However, this paper shows that all Shellsort sorting networks with monotonically decreasing increments are of size $\Omega(N \log^2 N / \log\log N)$. Previously, only the trivial $\Omega(N \log N)$ lower bound was known for this class of increment sequences.

Also, although the results presented here are for Shellsort sorting networks, they do provide some insight into the sequential Shellsort algorithm. In particular, many proofs of upper bounds on the sequential running time of Shellsort use templates to bound the time requirements of the insertion sorts. Such proofs ignore the adaptive nature of the sequential algorithm, so they actually imply an identical upper bound on the number of comparators in the corresponding Shellsort sorting network. Given the results of this paper, it is clear that such a proof will yield an upper bound that is $\Omega(N \log^2 N / \log\log N)$ if monotonically decreasing increments are chosen. It should be noted, however, that the $\Omega(N \log^2 N / \log\log N)$ bound applies only to the upper bound created by such a proof technique and not to the actual running time of the sequential algorithm.

This paper is organized as follows. Section 2 contains definitions and notation that are used throughout the paper, and §3 proves a connection between template costs (defined in §2) and the number of comparators that a sorting network must have. In §4, the relationship between the cost of an increment and the number of items that it adds to a template is established. Section 5 proves a lower bound on the number of elements missing from a template, §6 proves a lower bound on the total template costs of any Shellsort algorithm with monotonically decreasing increments, and §7 gives a lower bound on the number of comparators required when monotonically decreasing increments are used. Some conclusions and open problems are presented in §8.

2. Definitions and notation. The expression $\log N$ will be used to indicate $\log_2 N$. The set of nonnegative integers will be denoted by I. An *I-set* is any subset of I. Uppercase letters will be used to name I-sets, and lowercase letters will be used for arbitrary members of I-sets. An uppercase letter with the subscript $[i]$, where $i \geq 1$, will be used to indicate the ith smallest member of an I-set. For example, if Z is an I-set, then $Z_{[1]}$ is the smallest member of Z. Given a nonempty I-set Z, $\text{Min}(Z) = Z_{[1]}$, and if Z is finite, $\text{Max}(Z) = Z_{[|Z|]}$. Given an I-set Z, the *complement* of Z, written $\overline{Z}$, is $I - Z$. Given an I-set Z, a positive integer u, and an integer v, $\text{Slice}(Z, u, v) = \{z \in Z \mid z \equiv v \pmod{u}\}$.

An I-set Y is a *template* if it contains 0, it is closed under addition, and $\overline{Y}$ is finite. Given a template Y, a positive integer u, and an integer v, $\text{Lack}(Y, u, v) = |\text{Slice}(\overline{Y}, u, v)|$. The *cost of increment u with respect to template* Y, written $\text{Cost}(Y, u)$, is $\text{Lack}(Y, u, 0)$.

Let $D = d_1, d_2, \cdots, d_k$ be a finite sequence of nonnegative integers, and let Z be an I-set where $|Z| \geq k$. The *product* of D and Z, written $D \odot Z$, is $\sum_{i=1}^{k} d_i Z_{[i]}$. An integer x is said to be *representable* by the I-set Z if there exists a finite sequence of nonnegative integers $D = d_1, d_2, \cdots, d_k$ where $D \odot Z = x$. The *span* of an I-set Z, written $\text{Span}(Z)$, is the I-set consisting of the integers that are representable by Z. The *weight* of a finite sequence $D = d_1, d_2, \cdots, d_k$, written $\text{Weight}(D)$, is $\sum_{i=1}^{k} d_i$. If u is representable by Z, then $\text{Size}(Z, u)$ is the minimum of $\text{Weight}(D)$ taken over all sequences D such that $D \odot Z = u$.

We will consider the input to a Shellsort sorting network to be an array of registers $A = A[1], A[2], \cdots, A[N]$ that holds N items. Each comparator in the sorting network will be viewed as performing a compare–exchange operation on a fixed pair of registers. The compare–exchange operation compares the contents of the two registers and exchanges the contents if they are out of order. We will divide the comparators into a sequential set of *stages*, each of which performs the z-sort for a given increment z. For any I-set Z, the array A is said to be *sorted by* Z if for all $z \in Z$, A is z-sorted. For any I-set Z, Y is the *template generated by* Z if $Y = \text{Span}(Z \cup \{i \in I \mid i \geq N\})$ (it is easily verified that Y is, in fact, a template). Recall that if the array A is z-sorted for each $z \in Z$, then it must also be y-sorted for each $y \in Y$ (the integers greater than or equal to N are included in Y because the array A contains only N data items, so it must be y-sorted for all $y \geq N$).

3. Template costs and comparators.

Throughout this section let Z be an I-set, let Y be the template generated by Z, and let x be a positive integer. Also, let c be the cost of x with respect to template Y, and let L be any integer, where $1 \leq L \leq N$. We will first show that the cost of an increment with respect to a template can be used to put an upper bound on the size of the sorting network stage that performs the sort for that increment. It is easy to see that if the array A is sorted by Z, there are at most c array locations i such that $i < L$, $A[i] > A[L]$, and $i \equiv L \pmod{x}$. Therefore, when we perform the x-sort each location L will have to be compared to at most c other locations. As a result, the stage of the network that performs the x-sort requires only $O(Nc)$ comparators. Pratt used a similar argument to prove the correctness of his sorting network [9].

Of course, we are trying to prove a lower bound on the size of Shellsort sorting networks. In the remainder of this section we will show that the stage of the network that performs the x-sort does, in fact, require $\Omega(Nc)$ comparators.

LEMMA 3.1. *For any integer L, $1 \leq L \leq N$, there exists an assignment of values to the array A such that A is sorted by Z and for all i, $1 \leq i < L$, where $L - i \notin Y$, $A[i] > A[L]$.*

Proof. Let $E = \{i \mid 1 \leq i \leq L \text{ and } L - i \in Y\}$, let $F = \{i \mid 1 \leq i \leq L \text{ and } L - i \notin Y\}$, and let $G = \{i \mid L < i \leq N\}$. Assign values to A as follows. For each $E_{[i]} \in E$ let $A[E_{[i]}] = i$, for each $F_{[i]} \in F$ let $A[F_{[i]}] = i + |E|$, and for each $G_{[i]} \in G$ let $A[G_{[i]}] = i + L$ (see Fig. 2). Note that for all i, $1 \leq i < L$, where $L - i \notin Y$, $A[i] > |E| = A[L]$. Therefore, all that remains to be shown is that A is sorted by Z.

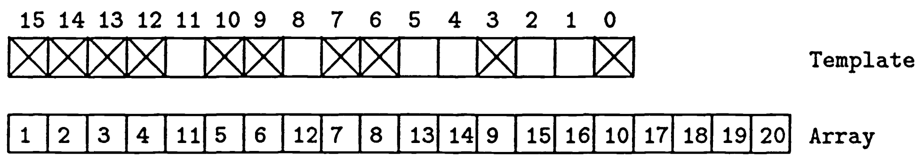


FIG. 2. *Worst-case array for Shellsort.*

Assume for the sake of contradiction that A is not sorted by Z. Then there exist i and j, where $i < j$, $A[i] > A[j]$, and $j - i \in Z$, which implies that $j - i \in Y$. Note that because $i < j$ and $A[i] > A[j]$, i and j cannot both be in E, they cannot both be in F, and they cannot both be in G. Also, $j \notin G$ because this would require that $A[j] > L$ and $A[i] \leq L$. Furthermore, $i \notin E$ because this would require that $A[i] \leq |E|$ and $A[j] > |E|$. Finally, $i \notin G$ because this would require that $i > L$ and $j \leq L$. Therefore, $i \in F$ and $j \in E$. However, $j \in E$ implies that $L - j \in Y$, and it was shown that $j - i \in Y$, so $(L - j) + (j - i) = L - i \in Y$ (because Y is a template and is closed under addition), which is a contradiction because $i \in F$ implies that $L - i \notin Y$. □

Recall that insertion sort works by performing a sequence of comparisons that insert each of the items into the list. Also, recall that insertion sort performs all of the comparisons that insert a given item into the list before performing any of the comparisons that insert the next item into the list. As a result, it is impossible for a single comparator in an insertion sort sorting network to participate in the insertion of two different items. We will use this observation to obtain a lower bound on the number of comparators required by a single stage of a Shellsort network.

THEOREM 3.2. *If A has been sorted by* Z, $x \leq N/2$, *and* $c =$ Cost(Y, x), *then at least* $Nc/24$ *comparators are needed to implement an* x*-sort of A.*

Proof. If $c = 0$, the proof is trivial. Otherwise, let $m =$ Max(Slice$(\overline{Y}, x, 0)$) (that is, m is the largest multiple of x that is not in Y). There are two cases based on the value of m.

Case 1: $m \leq 3N/4$. Let L be any integer where $3N/4 < L \leq N$. From Lemma 3.1 all c registers i, where $1 \leq i < L$ and $L - i \in$ Slice$(\overline{Y}, x, 0)$, can be such that $A[i] > A[L]$. Thus immediately before $A[L]$ is inserted it is possible that $A[L - jx] > A[L]$ for all j, where $1 \leq j \leq c$. As a result, at least c comparators are needed to insert $A[L]$ by using an insertion sort. Because there are at least $N/4$ such integers L, because c comparators are required for each L, and because no comparator can be shared by different values of L, at least $Nc/4$ comparators are needed.

Case 2: $m > 3N/4$. Let $U =$ Slice$(I, x, 0)$ (that is, U is the set of all nonnegative multiples of x), and let $V = \{i \in U \mid m/4 \leq i \leq 3m/4\}$. Let $k = m/x$, and note that $|V|$ is a function of k. Specifically, there are four cases. If $k \equiv 0 \pmod 4$, then $|V| = k/2 + 1$. If $k \equiv 1 \pmod 4$, then $|V| = \lfloor k/2 \rfloor$. If $k \equiv 2 \pmod 4$, then $|V| = k/2$. If $k \equiv 3 \pmod 4$, then $|V| = \lceil k/2 \rceil$. In any case, $|V| \geq \lfloor k/2 \rfloor$. Note that $c \leq m/x = k$, so $|V| \geq \lfloor c/2 \rfloor$. Also, note that $x \notin Y$ (because $x \in Y$ would imply that $c = 0$), $m \notin Y$, and $x \neq m$ (because $x \leq N/2$ and $m > 3N/4$), so $c \geq 2$ and $\lfloor c/2 \rfloor \geq c/3$. Therefore, $|V| \geq c/3$.

Now let $W = V \setminus Y$, and note that Max$(W) \leq 3N/4$. Because $m \notin Y$ and Y is closed under addition, for any $i \in Y$, where $i \leq m$, $m - i \notin Y$. Also, for any $i \in V$, $m - i \in V$. Therefore, for any $i \in Y \cap V$, $m - i \in \overline{Y} \cap V$. As a result, $|W| \geq |V|/2 \geq c/6$. Let L be any integer where $3N/4 < L \leq N$. From Lemma 3.1 all $|W| \geq c/6$ registers

i, where $1 \leq i < L$ and $L - i \in W$, can be such that $A[i] > A[L]$. Thus immediately before $A[L]$ is inserted it is possible that $A[L - jx] > A[L]$ for all j, where $1 \leq j \leq c/6$. As a result, at least $c/6$ comparators are needed to insert $A[L]$ by using an insertion sort. Because there are at least $N/4$ such integers L, because $c/6$ comparators are required for each L, and because no comparator can be shared by different values of L, at least $Nc/24$ comparators are needed. □

4. Efficiency of increments. At any time during the Shellsort algorithm let the *current template* be the template generated by the increments for which sorts have been performed. Before the sorts for the first increment are performed, the current template is missing $N - 1$ natural numbers, namely, $1, 2, \cdots, N - 1$. After the sort for the final increment (which must be 1) is performed, the current template contains all of the natural numbers. In this section we examine the relationship between the cost of an increment and the number of items that it adds to the current template.

For the remainder of this section let Z be an I-set, let x be a positive integer, let Y be the template generated by Z, and let c be the cost of x with respect to template Y. Also, let $Z' = Z \cup \{x\}$, and let Y' be the template generated by Z'.

THEOREM 4.1. *Given the above definitions*, $|Y' \setminus Y| \leq xc$.

Proof. Let $E = Y' \setminus Y$. Assume for the sake of contradiction that the claim is false, in which case $|E| > xc$. Therefore, there must exist a u for which $0 \leq u < x$ and $|\text{Slice}(E, x, u)| > c$. Let $F = \text{Slice}(E, x, u)$.

It will first be shown that for each $f \in F$ there exists a $t \in \text{Slice}(Y, x, u)$ where $t < f$. Let f be an arbitrary member of F, and note that $f \in E$. Therefore, f is not representable by Z, but f is representable by Z'. Let the sequence $D = d_1, d_2, \cdots, d_j$ be such that $D \odot Z' = f$. Note that if $Z'_{[i]} = x$, then $d_i > 0$ because f is not representable by Z. Let the sequence $H = d_1, \cdots, d_{i-1}, 0, d_{i+1}, \cdots, d_j$, and let $t = H \odot Z'$. Then $t < f$, t is representable by Z, and $t \equiv f \pmod{x}$. Therefore, $t \in \text{Slice}(Y, x, u)$ and $t < f$.

Now let $m = \text{Min}(\text{Slice}(Y, x, u))$. By the argument given in the previous paragraph $\text{Min}(F) > m$. For any i, $1 \leq i \leq |F|$, $F_{[i]} - m \notin Y$ because $F_{[i]} \notin Y$ and Y is closed under addition. But $F_{[i]} - m \equiv 0 \pmod{x}$, so $|F| \leq \text{Lack}(Y, x, 0) = \text{Cost}(Y, x) = c$, which is a contradiction. □

We can now outline the remainder of the lower-bound proof. We have shown in Theorem 3.2 that lower bounds on increment costs can be used to obtain lower bounds on the size of Shellsort sorting networks. We have just seen that only large increments are efficient in adding items to the current template. That is, for a given cost a large increment is capable of adding more new items to the template than will a small increment. Therefore, to obtain a small sorting network we should use large increments to add items to the template. Therefore, when using monotonically decreasing increments we should add as many items as possible to the template with the increments in the beginning of the sequence. However, we will show in Theorem 5.2 that there will always be a large number of items missing from the template. As a result, monotonically decreasing increments cannot be efficient. The remaining lemmas and theorems formalize this argument.

5. Elements missing from templates. In this section we will prove a lower bound on the number of elements missing from a template. Throughout this section let Z be a nonempty I-set with $\text{Min}(Z) > 1$ and $\text{Max}(Z) < N$, let Y be the template generated by Z, and let $x = \text{Min}(Z)$.

LEMMA 5.1. *For any k such that* $Y_{[k]} < N$, $Y_{[k]} \geq x \log k / \log(|Z| + 1)$.

Proof. Assume for the sake of contradiction that the claim is false, so there exists a k such that $Y_{[k]} < N$ and $Y_{[k]} < x \log k / \log(|Z|+1)$. Let $E = \{Y_{[i]} \mid 1 \le i \le k\}$. For each $Y_{[i]} \in E$, $Y_{[i]} < N$, so $Y_{[i]} \in \text{Span}(Z)$. Let j be such that $Y_{[j]} \in E$, and for all i, where $1 \le i \le k$, $\text{Size}(Z, Y_{[i]}) \le \text{Size}(Z, Y_{[j]})$. Let $h = \text{Size}(Z, Y_{[j]})$. Note that there are at most $(|Z|+1)^h$ integers $i \in \text{Span}(Z)$ such that $\text{Size}(Z, i) \le h$. Therefore, $k = |E| \le (|Z|+1)^h$, so $h \ge \log k / \log(|Z|+1)$ and $x \log k / \log(|Z|+1) \le xh \le Y_{[j]} \le Y_{[k]}$, which is a contradiction. □

THEOREM 5.2. *If* $1 \le |Z| < \log^2 N$, $N^{1/2} \le x \le N/\log N$, *and* $N \ge 2^{64}$, *then* $|\overline{Y}| > x \log N / 16 \log\log N$.

Proof. Let $u = \text{Max}(\overline{Y})$. Note that for all i, $u + 1 \le i \le u + x$, $i \in Y$. Therefore, $u + x \ge Y_{[x]}$ and $u \ge Y_{[x]} - x$. Because $x \le N/\log N$, $x \log x / \log(|Z|+1) < N$. Therefore, either $Y_{[x]} \ge N > x \log x / \log(|Z|+1)$ or $Y_{[x]} < N$, and from Lemma 5.1 $Y_{[x]} \ge x \log x / \log(|Z|+1)$. Thus in either case $u \ge x(\log x / \log(|Z|+1) - 1)$. Because $\log x \ge \frac{1}{2} \log N$ and $\log(|Z|+1) \le 2 \log\log N$, $\log x / \log(|Z|+1) \ge \log N / 4 \log\log N$ and $u \ge x(\log N / 4 \log\log N - 1)$. Because $N \ge 2^{64}$, $\log N / 4 \log\log N \ge 8/3$ and $(\log N / 4 \log\log N) - 1 \ge (\log N / 4 \log\log N)/2 = \log N / 8 \log\log N$. Therefore, $u \ge x \log N / 8 \log\log N$. Because Y is closed under addition and $u \notin Y$, for each i, $0 \le i \le u$, either $i \notin Y$ or $u - i \notin Y$. Therefore, $|\overline{Y}| > u/2 \ge x \log N / 16 \log\log N$. □

6. Template costs for monotonic increments. In this section we prove a lower bound on the total template costs incurred when monotonically decreasing increments are used. Throughout this section let $N \ge 2^{64}$ and let Z be an I-set where $\text{Min}(Z) = 1$ and $\text{Max}(Z) < N$. Let $k = |Z|$, where $1 \le k < \log^2 N$. Let $R_{k+1} = \emptyset$ and for all i, where $1 \le i \le k$, let $R_i = \{z \in Z \mid z \ge Z_{[i]}\}$. For all i, where $1 \le i \le k+1$, let Y_i be the template generated by R_i and let $b_i = |\overline{Y_i}|$. For all i, where $1 \le i \le k$, let $c_i = \mathit{Cost}(Y_{i+1}, Z_{[i]})$.

LEMMA 6.1. *For all* i, $1 \le i \le k$, *if* $N^{1/2} \le Z_{[i]} \le N/\log N$, *then* $c_i > (b_{i+1}/b_i - 1)(\log N / 16 \log\log N)$.

Proof. From Theorem 4.1 $c_i \ge (b_{i+1} - b_i)/Z_{[i]}$. From Theorem 5.2 $b_i > Z_{[i]} \log N / 16 \log\log N$, so $Z_{[i]} < 16 b_i \log\log N / \log N$ and $c_i > (b_{i+1} - b_i) \log N / 16 b_i \log\log N = (b_{i+1}/b_i - 1)(\log N / 16 \log\log N)$. □

LEMMA 6.2. *For any* u *and* v *where* $N^{1/2} \le Z_{[u]} < Z_{[v-1]} \le N/\log N$,

$$\sum_{i=u}^{v-1} c_i > (\log N / 16 \log\log N)(v-u)((b_v/b_u)^{1/(v-u)} - 1).$$

Proof. From Lemma 6.1

$$\begin{aligned}\sum_{i=u}^{v-1} c_i &> \sum_{i=u}^{v-1} (b_{i+1}/b_i - 1)(\log N / 16 \log\log N) \\ &= (\log N / 16 \log\log N)\left(u - v + \sum_{i=u}^{v-1} b_{i+1}/b_i\right).\end{aligned}$$

Note that

$$\prod_{i=u}^{v-1} b_{i+1}/b_i = b_v/b_u.$$

Because the arithmetic mean is always greater than or equal to the geometric mean,

$$(1/(v-u))\sum_{i=u}^{v-1} b_{i+1}/b_i \geq (b_v/b_u)^{1/(v-u)}$$

and

$$\sum_{i=u}^{v-1} b_{i+1}/b_i \geq (v-u)(b_v/b_u)^{1/(v-u)}.$$

Therefore,

$$\sum_{i=u}^{v-1} c_i > (\log N/16\log\log N)(v-u)((b_v/b_u)^{1/(v-u)}-1). \qquad \square$$

THEOREM 6.3. $\sum_{i=1}^{k} c_i \geq \log^2 N/192\log\log N$.

Proof. If $\operatorname{Max}(Z) \leq N^{1/2}$, then from Theorem 4.1

$$\begin{aligned} N-1 &= \sum_{i=1}^{k}(b_{i+1}-b_i) \\ &\leq \sum_{i=1}^{k} c_i Z_{[i]} \\ &\leq \sum_{i=1}^{k} c_i N^{1/2}, \end{aligned}$$

so $\sum_{i=1}^{k} c_i \geq (N-1)/N^{1/2} \geq \log^2 N/192\log\log N$.

If $\operatorname{Max}(Z) > N^{1/2}$, then let u and v be such that $Z_{[u]}$ is the smallest member of Z that is greater than $N^{1/2}$ and $Z_{[v]}$ is the largest member of Z that is less than or equal to $N/\log N$. There are two cases.

Case 1: $b_u \geq N^{1/2}\log^2 N$. From Theorem 4.1

$$\begin{aligned} N^{1/2}\log^2 N &\leq b_u \\ &= \sum_{i=1}^{u-1}(b_{i+1}-b_i) \\ &\leq \sum_{i=1}^{u-1} c_i Z_{[i]} \\ &\leq \sum_{i=1}^{u-1} c_i N^{1/2}, \end{aligned}$$

so $\sum_{i=1}^{u-1} c_i \geq \log^2 N$ and $\sum_{i=1}^{k} c_i \geq \log^2 N/192\log\log N$.

Case 2: $b_u < N^{1/2}\log^2 N$. Note that if $v = k$, then $b_{v+1} = N-1$, whereas if $v < k$, then because $Z_{[v+1]} > N/\log N$, $b_{v+1} \geq N/\log N - 1$. So in either case $b_{v+1} \geq N/\log N - 1 \geq N/\log^2 N$. From Lemma 6.2

$$\sum_{i=u}^{v} c_i \geq (\log N/16\log\log N)(v-u+1)((b_{v+1}/b_u)^{1/(v-u+1)}-1).$$

Let $t = v - u + 1$, and note that $t \leq k < \log^2 N$. Then

$$\begin{aligned}
&(\log N/16 \log\log N)(v-u+1)((b_{v+1}/b_u)^{1/(v-u+1)} - 1) \\
&\quad \geq (\log N/16 \log\log N)(t)((N^{1/2}/\log^4 N)^{1/t} - 1) \\
&\quad \geq (\log N/16 \log\log N)(t)(N^{1/8t} - 1).
\end{aligned}$$

Because $t(N^{1/8t}-1) = t(e^{\ln N/8t}-1)$ and $e^x = \sum_{i=0}^{\infty} x^i/i!$, $t(e^{\ln N/8t}-1) \geq t(\ln N/8t) = \ln N/8 > \log N/12$. Therefore,

$$\begin{aligned}
\sum_{i=1}^{k} c_i \geq \sum_{i=u}^{v} c_i& \\
&\geq (\log N/16 \log\log N)(t)(N^{1/8t} - 1) \\
&\geq \log^2 N/192 \log\log N. \qquad \square
\end{aligned}$$

From Theorem 6.3 it is clear that any upper bound on the sequential running time that is based on template costs and that uses monotonically decreasing increments will be at least $\Omega(N \log^2 N/\log\log N)$.

7. Network sizes for monotonic increments. This section establishes a lower bound on the size of Shellsort sorting networks when monotonically decreasing increments are used. The proof is based on Theorems 3.2 and 6.3.

THEOREM 7.1. *Shellsort sorting networks with monotonically decreasing increments require* $\Omega(N \log^2 N/\log\log N)$ *comparators to sort N items.*

Proof. It will be shown that when $N \geq 2^{64}$ at least $N \log^2 N/4608 \log\log N$ comparators are required. Let Z be the set of increments that are used, and let $k = |Z|$. Let $R_{k+1} = \emptyset$, and for all i, where $1 \leq i \leq k$, let $R_i = \{z \in Z \mid z \geq Z_{[i]}\}$. For all i, where $1 \leq i \leq k$, let Y_i be the template generated by R_i. For all i, where $1 \leq i \leq k$, let $c_i = \text{Cost}(Y_{i+1}, Z_{[i]})$. It will be assumed that for all i, where $1 \leq i \leq k$, $c_i \geq 1$ because increments that have no cost have no effect on the order of the data items. Two cases will be considered.

Case 1: $k \geq \log^2 N$. Because for each i, where $1 \leq i \leq k$, $c_i \geq 1$, from Theorem 3.2 at least $N/24$ comparators are required to perform each $Z_{[i]}$-sort. Therefore, a total of at least $\sum_{i=1}^{k} N/24 \geq N \log^2 N/24$ comparators are required.

Case 2: $k < \log^2 N$. From Theorem 3.2 at least $(N/24) \sum_{i=1}^{k} c_i$ comparators are required; from Theorem 6.3 $\sum_{i=1}^{k} c_i \geq \log^2 N/192 \log\log N$, so $N \log^2 N/4608 \log\log N$ comparators are required. $\square$

8. Conclusions and open problems. We have shown that all Shellsort sorting networks based on monotonically decreasing increments require $\Omega(N \log^2 N/\log\log N)$ comparators. This lower bound nearly matches the upper bound of $O(N \log^2 N)$ that was proved by Pratt. One open problem is the removal of the $\Theta(\log\log N)$ gap between these lower and upper bounds.

Another interesting open problem consists of removing the restriction that the increments be monotonically decreasing. Although the case of monotonically decreasing increments is an important special case, it is natural to ask if a similar lower bound applies when the increments are not monotonically decreasing. The results of §3, which establish the relationship between template costs and comparators, and the results of

§4, which bound the efficiency of increments, are independent of whether or not the increments are monotonically decreasing. However, the results of §5, which give a lower bound on the number of items missing from a template, are stated in terms of the smallest increment that has been used. As a result, if the increments are not monotonically decreasing, this lower bound is seriously weakened. In fact, a single very small increment will render the lower bound given in §5 worthless. As a result, a stronger theorem on the number of items missing from a template appears to be needed in order to address the issue of nonmonotonic increments.

Acknowledgment. I am very grateful for the helpful comments and recommendations given by the anonymous referee.

REFERENCES

[1] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, *An $O(n \log n)$ sorting network*, Combinatorica, 3(1983), pp. 1–19.

[2] D. GALE AND R. M. KARP, *A phenomenon in the theory of sorting*, J. Comput. System Sci., 6(1972), pp. 103–115.

[3] T. HIBBARD, *An empirical study of minimal storage sorting*, Comm. ACM, 6(1963), pp. 206–213.

[4] J. INCERPI, *A study of the worst-case of Shellsort*, Ph.D. thesis, Department of Computer Science, Brown University, Providence, RI, 1985.

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

[6] R. LAZARUS AND R. FRANK, *A high-speed sorting procedure*, Comm. ACM, 3(1960), pp. 20–22.

[7] T. LEIGHTON, *Tight bounds on the complexity of parallel sorting*, IEEE Trans. Comput., C-34(1985), pp. 344–354.

[8] A. PAPERNOV AND G. STASEVICH, *A method of information sorting in computer memories*, Prob. Inform. Transmission, 1(1965), pp. 63–75.

[9] V. R. PRATT, *Shellsort and sorting networks*, Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA, 1972.

[10] R. SEDGEWICK, *A new upper bound for Shellsort*, J. Algorithms, 7(1986) pp. 159–173.

[11] D. L. SHELL, *A high-speed sorting procedure*, Comm. ACM, 2(1959), pp. 30–32.

[12] M. A. WEISS, *Lower bounds for Shellsort*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1987.

[13] M. A. WEISS AND R. SEDGEWICK, *Tight lower bounds for Shellsort*, Tech. Report CS-TR-137-88, Department of Computer Science, Princeton University, Princeton, NJ, 1988.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 72–78, February 1993

A NOTE ON POSET GEOMETRIES*

JOEL FRIEDMAN†

Abstract. This note describes how varying the geometric representation of a poset can be applied to "poset balancing." It is shown that the 1/3, 2/3 balancing property holds for a certain class of posets whose number of relations is sufficiently small, in a certain sense.

Key words. partial order, linear extension, poset balancing

AMS(MOS) subject classifications. primary 06A10; secondary 68P10

1. Introduction. Given a poset (partially ordered set) for elements x and y, let $p(x < y)$ denote the fraction of completions of the partial order to a total order in which $x < y$. Fredman's conjecture, in connection with [Fre76] (also conjectured by Linial in [Lin84]), is that any nontotal poset (i.e., poset that is not a total order) has two elements, x, y, for which $1/3 \le p(x < y) \le 2/3$. This conjecture arose in the context of studying the information theoretic bound on the complexity of sorting the elements of the poset. The nontrivial nontotal three-element poset shows this conjecture to be as optimistic as possible.

To date the conjecture is unresolved, but using convexity in geometric realizations of the posets, such as the techniques of Stanley in [Sta81], theorems have been proven with the $1/3, 2/3$ replaced by different constants. In [KS84], Kahn and Saks, proved the above conjecture with $3/11, 8/11$ as constants. In [KL88], Kahn and Linial gave a simpler proof of the conjecture with $1/(2e), (2e-1)/(2e)$ as constants (a referee has informed me that Khachian had earlier given a proof similar to that of [KL88] with constants $e^{-2}, 1-e^{-2}$).

Both proofs are based on convex geometry involving a geometric realization of S_n, the group of permutations on n objects. The point of this note is to remark that by varying the geometries one can sometimes get better results. This is definitely true when the poset has few enough relations in a certain sense. The geometries we use are suggested by the standard realization of S_n as its associated Coxeter complex (see, e.g., [Ron89]). We combine the varying geometries with the simplified technique of [KL88] to obtain improved results for such posets. In this note we prove the following theorem.

THEOREM 1.1. *For any $\epsilon > 0$ there is a C such that the following is true. Let P be a poset on $\{x_1, \dots, x_n\}$, and let a_i and b_i denote the number of elements, respectively, $>$ and $<$ than x_i in P. If for every permutation $\sigma \in S_n$ we have*

$$\sum_{i=1}^{n} \left(\frac{\sigma(i)}{a_i + 1} + \frac{n + 1 - \sigma(i)}{b_i + 1} \right) \ge Cn,$$

then P has elements x, y with $(1/e) - \epsilon \le p(x < y) \le 1/2$.

This appears as Theorem 3.6 of §3 and is proven there. This is the precise sense of P having "few enough relations" mentioned earlier. As applications we have Theorem 1.2.

THEOREM 1.2. *For any $\epsilon > 0$ there is a C such that if P either (1) has at least $C\sqrt{n}$ maximal (or minimal) elements or (2) has no chain of length $\ge 2\log_2 \log n - C$, then P has elements x, y with $(1/e) - \epsilon \le p(x < y) \le 1/2$. For any $\epsilon > 0$ and $\nu > 0$, there is a*

*Received by the editors November 1, 1990; accepted for publication (in revised form) November 25, 1991. The author also wishes to acknowledge the National Science Foundation for supporting this research in part under a Presidential Young Investigator Award grant, CCR-8858788.

†Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

$\mu > 0$ such that the same conclusion holds if P has some νn of its elements each unrelated to at least $(1-\mu)n$ other elements.
This appears as Theorems 3.5, 3.7, and 3.8 of §3.

We also note that other special cases of the poset $(1/3, 2/3)$ conjecture have been resolved. In [Lin84], Linial proves the conjecture for posets of width 2. In [Kom], Komlós proves that for any $\epsilon > 0$, there is a function $f_\epsilon(n) = o(n)$ such that any poset with at least $f_\epsilon(n)$ minimal elements has two elements, x, y, such that $|p(x<y) - 1/2| \le \epsilon$; here $f(n)/n$ decreases to zero exponentially fast in the inverse of some Ramsey-type function of n. Also Kahn and Saks [KS84] have conjectured that as the width of the poset tends to infinity, a $|p(x<y) - 1/2| \le o(1)$ balancing result should hold.

2. Variants of a standard model.

A standard geometric model of S_n is its associated Coxeter complex (see [Ron89]). One views this as a triangulation of the $(n-2)$-dimensional sphere. The convex polytope determined as the convex hull of the simplices of this triangulation (this polytope looks like a beachball) is a realization of S_n such that the realization of every poset is convex. Of course, there is no reason to insist that this polytope's vertices all lie on one sphere. By moving certain vertices further or closer to the center, we get different convex polytopes.

So consider n points $v_1, \ldots, v_n \in \mathbf{R}^{n-1}$ not contained in any hyperplane of dimension $n-2$. Every point $v \in \mathbf{R}^{n-1}$ can be uniquely written as

$$v = \sum \alpha_i v_i, \quad \text{with} \quad \sum \alpha_i = 1. \tag{2.1}$$

For a permutation of $\{1, \ldots, n\}$, $\sigma = \{i_1, \ldots, i_n\}$, let

$$A_\sigma = \left\{ \sum \alpha_i v_i \mid \alpha_{i_1} \ge \cdots \ge \alpha_{i_n} \right\}.$$

If U is any convex body, say, containing the v_i, then $\sigma \mapsto U_\sigma \equiv A_\sigma \cap U$ is a realization of S_n in which every poset corresponds to a convex subset of U. U_σ will be *adjacent* to U_τ, i.e., will share a facet (i.e., an $(n-2)$-dimensional face), if and only if σ and τ differ by some transposition (i,j), and in that case the facet lies on the hyperplanes H_{ij} containing $(v_i + v_j)/2$ and all v_k with $k \ne i, j$. When no confusion will occur, we will often simply refer to this facet as H_{ij}.

For future reference, the α_i's in (2.1) are called the *barycentric coordinates* of v (with respect to the v_i's). If U is the simplex spanned by the v_i's, the *barycentric distance* of v to a facet, F, of U (i.e., a simplex spanned by any $n-1$ distinct v_i's), is the barycentric coordinate of v with respect to the v_i not contained in F.

We now make some explicit calculations to describe various choices of U. For simplicity, we perform them in $\mathbf{R}^n$ restricted to the hyperplane $x_1 + \cdots + x_n = 1$. For a subset $S \subset \{1, \ldots, n\}$, let e_S be the vector that is 1 on the ith coordinate if $i \in S$ and 0 elsewhere. Let $Q = (1/n, \ldots, 1/n)$. For positive real $\theta_1, \ldots, \theta_n$, consider the collection of points

$$v_S = \theta_S e_S + \left(1 - |S|\theta_S\right) Q$$

with $\theta_S = \theta_{|S|}$, ranging over all nonempty proper subsets, S, of $\{1, \ldots, n\}$. Clearly all these vertices lie on the hyperplane $x_1 + \cdots + x_n = 1$. Let U be their convex hull.

LEMMA 2.1. *The following two conditions are equivalent*: (1) *no v_S is in the interior of U and* (2) *for all $i < j$*

$$i\theta_i \leq j\theta_j \quad and \quad (n-i)\theta_i \geq (n-j)\theta_j. \tag{2.2}$$

Proof. By symmetry the first condition is equivalent to saying that for any $|S|$, the centers of mass of the sets

$$E_{j,k} = \left\{ v_T \;\middle|\; |T \cap S| = j, |T - S| = k \right\}$$

for all k, j lie between (or on) Q and v_S. Each of these gives an inequality between $\theta_{|S|}$ and θ_{j+k}, which is exactly of the form of those of (2.2) (except when $j + k = |S|$, which is trivial), and conversely each inequality in (2.2) arises in this way. □

For future reference we note that the distance of v_S to a half plane H_{ij} is just $\theta_S/\sqrt{2}$ if exactly one of i, j are in S (and 0 otherwise); this is seen by noting that the reflection through H_{ij} merely exchanges the ith and jth coordinates. We also note some familiar choices of θ_i. The choice $\theta_i = 1/i$ and $\theta_i = 1/(n-i)$ are simplices with vertex sets $\{v_S\}$ ranging over S of respective sizes 1 and $n-1$. Choosing the v_S's to be equidistant from Q gives

$$\theta_i = \sqrt{\frac{n}{i(n-i)}}.$$

We now describe some features of varying the geometry. First, we make the observation that in any poset there exists an ordering of the elements $\sigma = \{x_1, \ldots, x_n\}$ such that $p(x_i > x_{i+1}) \geq 1/2$ for all i. We call such a σ *optimal.* Its existence follows from the fact that any tournament has a Hamiltonian path. This statement also implies that $p(x_i > x_j) > 2/3$ for all $i > j$ if P is a counterexample to Fredman's conjecture. The point to our method is that by fixing P and such an ordering we can choose the geometry best suited to the situation at hand. We will apply this to the centroid method used in [KS84] and [KL88], using the simplified technique of the latter. We explain this in the next section.

It is sometimes easier to visualize the problem and amusing, if not particularly useful, to state the poset problem in the "real-estate" terminology (see [Ron89]). Given a Coxeter complex, there are two natural notions of convexity for a subset of chambers—that of metric convexity and that of being an intersection of half-apartments. It is easy to see that these two notions are equivalent; in the case of S_n, a convex set is merely a poset, and our question is to try to find a wall that divides a given convex set P into roughly equal parts. For example, since any collection of ≥ 2 chambers is nontrivially divided by some wall, it follows by descending induction on k that for all $k \geq 2$ there is a collection of $< k$ chambers all of whose bounding half-apartments contain more than half of P; this again proves the existence of an optimal σ.

3. Centroid type arguments. We review the techniques in [KL88]. They start with the observation given below.

LEMMA 3.1. *Let C be a convex body in $\mathbf{R}^m$ such that the centroid of C has x_1 coordinate $-\alpha$ and contains points with x_1 coordinates u and $-w$, with some $u, w, \alpha \geq 0$. Then*

$$\frac{|C \cap \{x_1 \geq 0\}|}{|C|} \geq \min_{U \geq u, W \geq w} \left(\frac{1}{1 + \frac{1}{m-1} + \frac{(m+1)\alpha - W}{(m-1)U}} \right)^{m-1} \frac{1}{1 + \frac{W}{U}}. \tag{3.1}$$

Furthermore the right-hand side above is minimized at $U = u, W = w$ if $u \geq u^*$ and otherwise at $U = u^*, W = w$, where

$$u^* = \frac{wm\Big(w - \alpha(m+1)\Big)}{w + \alpha(m^2-1)}.$$

Proof. The proof is a simple argument that shows that the worst case C is a "double cone," in the spirit of Mityagin (see [Mit68]). Calculating the worst case volume ratio on this basis is easy and yields (3.1), which is essentially straight from [KL88]. Differentiating in U and W yields the second part, using the fact that $w \geq (\alpha(m+1)+u)/m$ always holds in the above situation. □

If $\alpha = 0$ in the above, the above volume ratio is at least $1/e$, which is Mityagin's result. So we can expect volume ratios close to this if α is small enough.

The argument in [KL88] is as follows. Fix a realization of S_n as in the previous section. Let σ be an optimal total order, and let c be the centroid of the poset P, where we identify P with its realization. We can assume $c \in A_\sigma$ (or else we can apply Mityagin's result), so consider c's barycentric coordinates with respect to the vertices of A_σ. If H_{ij} is a facet of A_σ with i and j related in P (i.e., either $i < j$ or $i > j$ in the partial ordering), then P itself lies to one side of H_{ij} and it easily follows that the barycentric distance of c to H_{ij} is at least $1/n$. Hence, there must exist some facet, H_{ij}, of A_σ whose barycentric distance is $\leq 1/n$ such that i, j are unrelated in P, and in particular $A_{\sigma'}$ lies in P, where σ' is σ followed by the transposition (i, j). Then we can apply Lemma 3.1 with $m = n-1, u = w = 1, \alpha = 1/(m+1)$, which gives a volume ratio $\geq 1/(2e)$.

Actually, the realization used in [KL88], [KS84], and [Sta81] is different from ours; namely, they use the cube $[0,1]^n$ with A_σ being the set $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$. In this realization, when H_{ij} is a facet of A_σ, there is never any point in P that is further away from H_{ij} than one of the vertices of A_σ. If we use our realizations, then it can happen that some of P's points are further away, and we can get better results. This can be guaranteed to be the case when P is "sufficiently sparse."

More precisely, recall from the last section that the distance from a vertex v_S to is proportional to θ_S. The choice of $\theta_i = 1$ would yield a situation like the cube realization, but varying the θ_i's allows some improvement. Varying the θ_i's involves slightly different applications of Lemma 3.1, namely, those outlined in the following lemma.

LEMMA 3.2. *For any positive ϵ there is a positive $\delta = \delta(\epsilon)$ such that if, in Lemma* 3.1, *α is less than $\delta u/m$ or $\delta w/m$ in Lemma* 3.1, *then*

$$\frac{|C \cap \{x_1 \geq 0\}|}{|C|} \geq \frac{1}{e} - \epsilon.$$

Proof. This is an easy calculation. If we have $(m+1)\alpha/U \leq \delta$ for some small δ (slightly different from the δ in the statement of the lemma), then the right-hand side of (3.1) with $U = u$ is bounded below by

$$\left(\frac{1}{1 + \frac{1+\delta}{m-1} + \frac{-W}{(m-1)u}}\right)^{m-1} \frac{1}{1 + \frac{W}{u}}.$$

By substituting $t = W/u$ and differentiating, we can see that for any fixed m there is a δ making the above expression $\geq (1/e) - \epsilon$ for all positive W. On the other hand, for large m the above expression is

$$\geq \left(1+O\left(\frac{1}{m}\right)\right)e^{-(1+\delta)}e^{W/u}\frac{1}{1+\frac{W}{u}},$$

and as before we see that this is $\geq (1/e)-\epsilon$ for m sufficiently large and some positive δ (independent of m). This proves the first case of the lemma. In the second case, we write $(m+1)\alpha - W \leq -(1-\delta)W$ and proceed similarly, showing that a small enough δ yields the desired lower bound. □

Returning to the situation at hand, given a poset P fix settings of the θ_i's, consider any optimal $\sigma=(x_1,\dots,x_n)$, and let c be the centroid of P. Being interested in bounds of the form $1/e, 1-(1/e)$ or worse, we can assume that c lies in A_σ. The vertices of A_σ are v_{S_i} with $S_i=\{x_1,\dots,x_i\}$, $1\leq i\leq n-1$. Let

$$c=\sum_{i=1}^{n-1} v_{S_i}\alpha_{S_i}$$

be the barycentric representation of c in A_σ. For each i fix a set T_i that contains exactly one of x_i, x_{i+1} and with $v_{T_i}\in P$; usually we'll take T_i to be the smallest or largest such set, depending on the choice of θ_i's. T_i is any set whose elements are not $<$ any element not in T_i (in the partial order P).

COROLLARY 3.3. *If $x_i > x_{i+1}$ in P, then*

$$\alpha_{S_i}\geq \frac{1}{n-1}\frac{\theta_{T_i}}{\theta_{S_i}}.$$

If not, then for any $\epsilon>0$ we have

$$\alpha_{S_i}\geq \delta(\epsilon)\frac{1}{n-1}\frac{\theta_{T_i}}{\theta_{S_i}},$$

unless $p(x_i<x_{i+1})\geq (1/e)-\epsilon$, for $\delta=\delta(\epsilon)$ as in Lemma 3.2.

Proof. In the first case, the facet, H, of A_σ opposite v_{S_i}, bounds P, and yet there is a point of P, v_{T_i}, whose distance to H is $\theta_{T_i}/\sqrt{2}$. c's distance to H must be at least $1/(n-1)$ of T_i's distance to H. On the other hand, c's distance to H is precisely $\alpha_{S_i}\theta_{S_i}/\sqrt{2}$.

The second case follows from Lemma 3.2, with similar distance considerations. □

COROLLARY 3.4. *If*

$$\sum_{i=1}^{n-1}\frac{\theta_{T_i}}{\theta_{S_i}}\geq \frac{n-1}{\delta(\epsilon)}, \tag{3.2}$$

then there exists an i with $p(x_i<x_{i+1})\geq (1/e)-\epsilon$.

We now seek situations in which we can guarantee that (3.2) will hold for appropriate θ_i's.

THEOREM 3.5. *For any $\epsilon>0$, there is a C such that if P contains at least $C\sqrt{n}$ maximal elements, then there exists elements x, y*

$$\frac{1}{e}-\epsilon\leq p(x<y)\leq \frac{1}{2}.$$

Proof. Take $\theta_i = 1/i$ in the above. If, in the above circumstances, x_i is maximal, then taking $T_i = \{x_i\}$ gives $\theta_{T_i}/\theta_{S_i} = |S_i|$. Hence,

$$\sum_{i=1}^{n-1} \frac{\theta_{T_i}}{\theta_{S_i}} \geq \sum_{i=1}^{C\sqrt{n}} i \geq \frac{n-1}{\delta}$$

for sufficiently large C. □

More generally we have the following theorem.

THEOREM 3.6. *For any $\epsilon > 0$, there is a C such that the following is true. Let P be a poset on $\{x_1, \ldots, x_n\}$, and let a_i and b_i denote the number of elements in P that are, respectively, $>$ and $<$ than x_i. If for every permutation $\sigma \in S_n$ we have*

$$\sum_{i=1}^{n} \left(\frac{\sigma(i)}{a_i + 1} + \frac{n+1-\sigma(i)}{b_i+1} \right) \geq Cn, \tag{3.3}$$

then P has elements x, y with $(1/e) - \epsilon \leq p(x < y) \leq 1/2$.

Proof. Consider the two choices for θ_i, $1/i$ and $1/(n-i)$. Taking C to be $2/\delta(\epsilon)$, we find that if (3.3) holds, then we can apply Corollary 3.4 for one of these two choices of θ_i. □

While the condition in Theorem 3.6 requires optimizing over σ and is not entirely explicit, in many cases it is not hard to check that it holds, such as when there exist $C\sqrt{n}$ maximal elements. We give some other examples to which Theorem 3.6 can be applied.

THEOREM 3.7. *For any $\epsilon > 0$, there is a C' such that the condition (and therefore conclusion) of Theorem 3.6 holds if every chain in P has length $\leq C' + 2\log_2 \log n$.*

THEOREM 3.8. *For any $\epsilon, \nu > 0$, there is a $\mu > 0$ such that the condition (and therefore conclusion) of Theorem 3.6 holds if some νn of P's elements are each unrelated to more than $(1-\mu)n$ (possibly different) elements of P.*

Proof. For the latter theorem, each element x_i unrelated to more than $(1-\mu)n$ has both a_i and b_i less than μn. Hence, it suffices to chose μ so that ν/μ exceeds $C(\epsilon)$ of Theorem 3.6. To prove the former, let X_i be the set of nodes whose longest chain from a maximal element is of length $i+1$; for example, X_1 is the set of maximal elements. Then any two elements of any X_i are unrelated. Then, setting $n_i = |X_i|$, we have

$$\sum_{i=1}^{n} \frac{\sigma(i)}{a_i+1} \geq \sum_{i=1}^{n_1} i + \sum_{i=n_1+1}^{n_2} \frac{i}{n_1+1} + \sum_{i=n_1+n_2+1}^{n_3} \frac{i}{n_1+n_2+1} + \cdots,$$

which, within a constant, is

$$\geq n_1^2 + \frac{n_2^2}{n_1} + \frac{n_3^2}{n_2} + \cdots. \tag{3.4}$$

A similar estimate holds for the sum in (3.3) involving the b_i's. Now let k be the length of the longest chain in P. If the condition for Theorem 3.6 is not met, then the expression in (3.4) must be bounded by Cn for some constant C. Then we conclude $n_1 \leq \sqrt{Cn}$ and then $n_2 \leq (Cn)^{3/4}$ and more generally

$$n_j \leq (Cn)^{1-(1/2^j)}.$$

Now let k be the length of the longest chain in P. Applying the same argument to the sum in (3.3) involving the b_i's, we conclude that

$$n_{k+1-j} \leq (Cn)^{1-(1/2^j)}$$

for all j. Hence,

$$n = n_1 + \cdots + n_k \leq k(Cn)^{1-\frac{1}{2^{(k+1)/2}}},$$

and thus $(k+1)/2 \geq \log_2 \log n + C'$. Hence, if $k \leq C'' + 2\log_2 \log n$, Theorem 3.6 must apply. $\square$

4. Concluding remarks. There are some other possible variants on these techniques. For one thing, we can vary the θ_S's even over S's of the same size. Of course, it may no longer be true that the A_σ's all have the same volume, but if we are only interested in $1/3, 2/3$ type results we might have some room for slight variations of volume.

On some level it seems appealing to phrase the poset question in terms of finding a wall separating a convex set of chambers in a Coxeter complex into roughly equal sizes, but it is not clear if this is of any use. It is easy to see that any convex set of a general Coxeter complex on k generators has a wall separating it into sets of fractional sizes between $1/(k+1), k/(k+1)$ and that this is the best one can say. From this point of view, it is clear that for the poset problem one is making use of the special fact that most of the generators of S_n commute.

Acknowledgment. The author wishes to thank Nati Linial for useful discussions.

REFERENCES

[Fre76] M. FREDMAN, *How good is the information theory bound in sorting?* Theoret. Comput. Sci., 1 (1976), pp. 355–361.

[KL88] J. KAHN AND N. LINIAL, *Balancing extensions via Brunn-Minkowski*, Tech. Report, Rutgers University, New Brunswick, NJ, November 1988.

[Kom] J. KOMLÓS, *A strange pigeon-hole principle*, preprint.

[KS84] J. KAHN AND M. SAKS, *Balancing poset extensions*, Order, 1 (1984), pp. 113–126.

[Lin84] N. LINIAL, *The information theoretic bound is good for merging*, SIAM J. Comput., 13 (1984), pp. 795–801.

[Mit68] B.S. MITYAGIN, *Two inequalities for volumes of convex bodies*, Math. Notes, 5 (1968), pp. 61–65.

[Ron89] M. RONAN, *Lectures on Buildings*, Academic Press, New York, 1989.

[Sta81] R.P. STANLEY, *Two combinatorial applications of the Aleksandrov–Fenchel inequalities*, J. Combin. Theory Ser. A, 31 (1981), pp. 56–65.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 79–101, February 1993

AN $O(n)$ ALGORITHM FOR DETERMINING THE SUBREGION-TREE REPRESENTATION OF A RECTANGULAR DISSECTION*

SUKHAMAY KUNDU†

Abstract. A rectangular dissection is a partition of a rectangular space R into $n \geq 1$ disjoint rectangles $\{r_1, r_2, \cdots, r_n\}$. A T_*-plan is a dissection that is obtained by repeated application of the (1) horizontal, (2) vertical, (3) left-spiral, and (4) right-spiral partitioning operations. Two common ways of representing a T_*-plan are the wall representation $w(D)$ and the subregion-tree representation $t(D)$. It is known [S. Kundu, *Comm. ACM*, 31 (1988), pp. 752–763] that these two representations are equivalent in that one can be uniquely determined from the other. This paper presents an optimal $O(n)$ algorithm for constructing $t(D)$ from $w(D)$, which improves the previous bound of $O(n^2)$ in [S. Kundu, *Comm. ACM*, 31 (1988), pp. 752–763]. The new algorithm is based on a domination relationship among the walls, which is defined here and represented by a digraph $G_w(D)$. The algorithm exploits the disjoint cycle property of $G_w(D)$ and the relationship between the tree $t(D)$ and the transitive reduction of the acyclic digraph obtained by merging the cycles of $G_w(D)$ into distinct nodes. The new method of constructing the tree $t(D)$ by means of the digraph $G_w(D)$ can be applied to an arbitrary class of dissections D that are generated by a finite family of partitioning operations that satisfies certain natural restrictions. The complexity of the algorithm remains $O(n)$ for many such families.

Key words. rectangular dissection, subregion representation, wall representation, acyclic digraph, transitive reduction, depth-first search

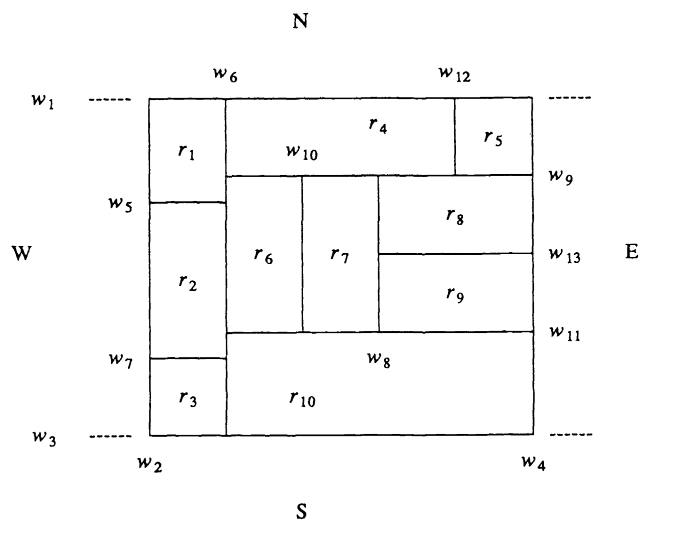
AMS(MOS) subject classifications. 68Q20, 68R10

1. Introduction.

Let R be a rectangular space in a plane. A partition of R into $n \geq 1$ disjoint rectangles $\{r_1, r_2, \cdots, r_n\}$ is called a *dissection*. Each r_j is called a *basic region*. A dissection D is called a *T-plan* [2] if each junction point between a vertical line and a horizontal line is a T-junction. The T-plans form models for space partitioning in very-large-scale-integration (VLSI) design [10], [11] and for floor-space planning in architectural design [2]–[9]. Figure 1(a) shows a T-plan consisting of 10 basic regions. We assume that the top and the bottom horizontal lines of R extend to infinity on both the left and the right (as indicated in the figure by the broken lines), although the extensions are usually not shown. The regions outside R, which are labeled W (for west), E (for east), N (for north), and S (for south), are called the *external regions*. A wall is a maximal horizontal or vertical line segment. The set of walls of a T-plan D forms its wall representation $w(D)$; see Fig. 1(b), where each wall is written as a pair of ordered lists. Each list consists of the basic and external regions adjacent to the wall from one side and ordered from left to right or from top to bottom. A *subregion* of D is a rectangular subspace that is the union of one or more basic regions. In general, a T-plan may not have any nontrivial subregion other than R and the basic regions r_j. Since either any two nontrivial subregions of a T-plan are disjoint or one is contained in the other (with few trivial exceptions; see §2), the subregions of a T-plan D form a tree structure $t(D)$. The basic regions of D form the terminal nodes of $t(D)$, and the whole space R corresponds to the root. The tree $t(D)$ is called the *subregion tree* of D. In Fig. 1(c), the labels h and v for the intermediate nodes of $t(D)$ indicate the type of partitioning operation applied to the associated subregion.

The tree representation $t(D)$ is particularly important for a subclass of T-plans, called T_**-plans*, which are obtained by repeated application of the (1) horizontal, (2) vertical, (3) left-spiral, and (4) right-spiral partitioning operations. (See Fig. 2.) We refer to these operations as h, v, s, and S partitions, respectively. The T_*-plans that are

*Received by the editors July 15, 1988; accepted for publication (in revised form) December 4, 1991.
†Computer Science Department, Louisiana State University, Baton Rouge, Louisiana 70803.



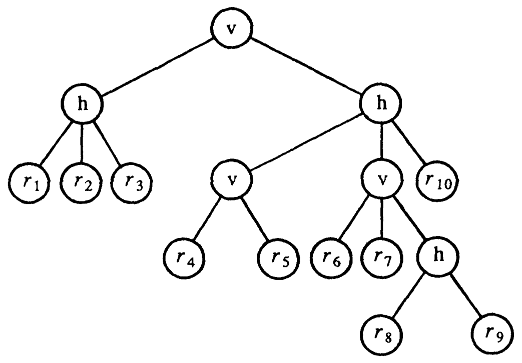
(a)

$w_1 = \langle(N), (W, r_1, r_4, r_5, E)\rangle$
$w_2 = \langle(W), (r_1, r_2, r_3)\rangle$
$w_3 = \langle(W, r_3, r_{10}, E), (S)\rangle$
$w_4 = \langle(r_5, r_8, r_9, r_{10}), (E)\rangle$

$w_5 = \langle(r_1), (r_2)\rangle$
$w_6 = \langle(r_1, r_2, r_3), (r_4, r_6, r_{10})\rangle$
$w_7 = \langle(r_2), (r_3)\rangle$
$w_8 = \langle(r_7), (r_8, r_9)\rangle$

$w_9 = \langle(r_4, r_5), (r_6, r_7, r_8)\rangle$
$w_{10} = \langle(r_6), (r_7)\rangle$
$w_{11} = \langle(r_6, r_7, r_9), (r_{10})\rangle$
$w_{12} = \langle(r_4), (r_5)\rangle$
$w_{13} = \langle(r_8), (r_9)\rangle$

(b)



(c)

FIG. 1. (a) *A T-plan D consisting of* 10 *basic regions* $\{r_1, r_2, \cdots, r_{10}\}$; (b) *walls of the T-plan* D; (c) *subregion tree* $t(D)$ *of* D.

formed by only h and v partitions are called *acyclic*; others are called *cyclic*. In [7]–[9] we present several important properties of the tree representation, including the equivalence between the wall representation and the tree representation. The main contributions of this paper are the following:

(1) It provides a linear $O(n)$ algorithm for constructing the tree representation $t(D)$ of a T_*-plan D from its wall representation $w(D)$, where n is the number of basic regions in D. This is an improvement over the $O(n^2)$ algorithm given in [7]. (The algorithm for converting $t(D)$ to $w(D)$ is $O(n)$, as shown in [7].)

(2) The method developed here for constructing $t(D)$ from $w(D)$ is general. The new method is applicable to an arbitrary class of T-plans that are generated by a finite family of partitioning operations that satisfies certain natural restrictions. The complexity of the algorithm remains $O(n)$ for many such families.

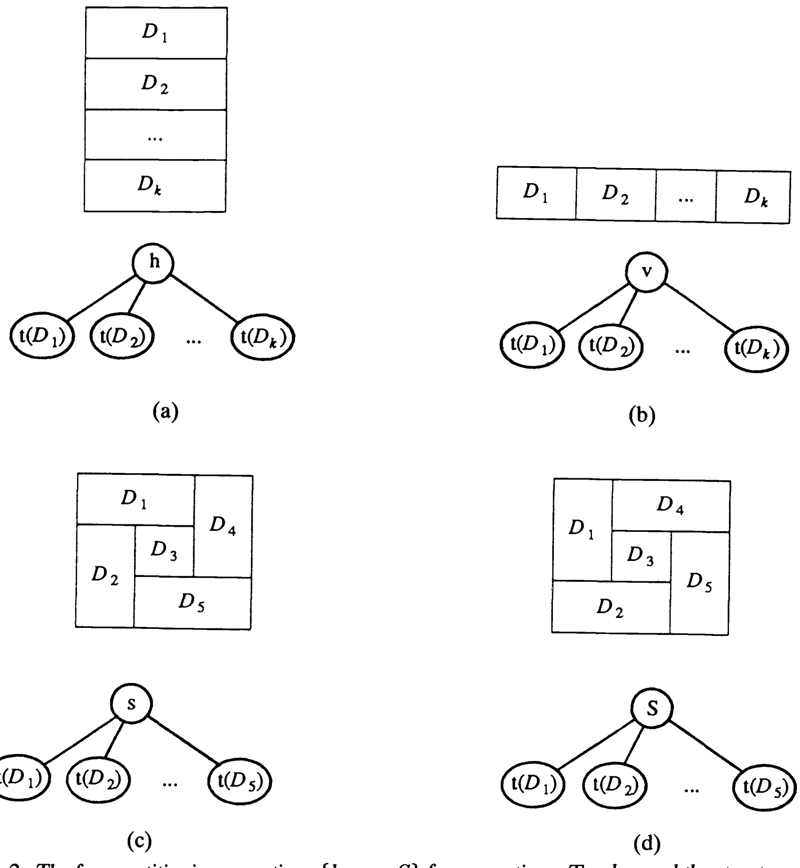


FIG. 2. *The four partitioning operations* $\{h, v, s, S\}$ *for generating a* T_**-plan and the structure of their associated subregion trees* $t(D)$; *partitioning operations*: (a) *horizontal*, (b) *vertical*, (c) *left spiral, and* (d) *right spiral.*

As a brief comparison of the new algorithm given here and the algorithm in [7], we note that the algorithm in [7] constructs the tree $t(D)$ in a top-down fashion, where a basic region r_i is processed once for every subregion containing it. This results in a computation time proportional to the sum of the path lengths in $t(D)$ from the root to the terminal nodes, which can be as much as $O(n^2)$ in the worst case and as small as $O(n \log_k n)$ in the best case. The best case occurs when the tree $t(D)$ is highly balanced and $k \geq 2$ is the minimum number of subregions created by a partitioning operator. The algorithm given here employs a more global approach using the properties of cycles in the digraph $G_w(D)$, which represents the domination properties of walls and basic regions in D (see §2). The digraph $G_w(D)$ is determined directly from the wall representation $w(D)$. The new algorithm builds the tree $t(D)$ in a top-down fashion as in [7], but it does not explicitly construct subregions of D. The two special properties of the digraph $G_w(D)$ that are exploited in the new algorithm are the following:

(1) The cycles in $G_w(D)$ correspond to the spiral partitions in D, and, moreover, no two cycles in $G_w(D)$ have a node in common. This property allows us to determine the cycles in $G_w(D)$ in linear $O(n)$ time.

(2) The transitive reduction of the acyclic graph obtained by merging each cycle of $G_w(D)$ into a distinct node has a tree structure that is isomorphic to $t(D)$ as an unordered tree. The orderings among the children of nodes in $t(D)$ are constructed by using a partial order "$<$" that is also determined from the wall representation $w(D)$. The ordering information in the wall representation $w(D)$ allows us to obtain the relevant part of the ordering "$<$", which is needed in the construction of $t(D)$, by using only $O(n)$ time.

2. Basic concepts. We denote a wall w of a T-plan by a pair of lists $w = \langle L_1, L_2\rangle$, one list for each side of the wall. For a horizontal wall (h-wall) L_1 is the list of basic regions and possibly some external regions adjacent to w from the above (north), and similarly L_2 is the list of basic regions and external regions adjacent from below (south). The regions in both L_1 and L_2 are ordered from the left to right. Likewise, for a vertical wall (v-wall) L_1 is the list of basic regions and external regions adjacent to w from the left (west) and L_2 is the list of basic regions and external regions adjacent to w from the right (east). The regions in both L_1 and L_2 are now ordered from north to south. We sometimes refer to L_1 as the first region list of w and to L_2 as the second region list. In Fig. 1(b) the horizontal walls are labeled $w_1, w_3, w_5, \cdots$, and the vertical walls are labeled $w_2, w_4, w_6, \cdots$. This odd–even labeling convention for the h-walls and v-walls is used throughout the paper. The two horizontal walls bordering the external regions N and S and the two vertical walls bordering the external regions W and E are called the *external* walls. They are also referred to as the north wall, south wall, etc. The other walls are called the *internal* walls. The four external walls are labeled w_1, w_2, w_3, and w_4, respectively. Note that each external wall is easily identified by the presence of N, S, W, and E in the first or the second region list.

We formally define the subregion tree $t(D)$ of a T_*-plan D recursively as follows. If D consists of only one basic region, region r_1, then $t(D)$ consists of a single node r_1. Otherwise, the root node of $t(D)$ that corresponds to the whole rectangular space R is labeled by one of the symbols $\pi = h, v, s$, or S, where $\pi = \pi(R)$ denotes the top-level partitioning operation applied to R. If the partition $\pi(R)$ decomposes R into k subregions $\{R_1, R_2, \cdots, R_k\}$, then the root node has k child nodes $\pi_i, 1 \le i \le k$, one for each subregion R_i. For $\pi(R) = s$ or S we have $k = 5$, and for $\pi(R) = h$ or v we have $k \ge 2$. The left-to-right ordering of the child nodes π_i is determined according to the schemas shown in Fig. 2. The subtree at each child node π_i is now determined recursively by the T_*-plan corresponding to the dissection on the subregion R_i.[1] The tree $t(D)$ is an *ordered* tree. Note that the only subregions of D for which there is no node in $t(D)$ are the ones that are formed by the union of $m, 2 \le m < k$, consecutive subregions of a k-part horizontal or vertical partition. For instance, there is no node in the tree in Fig. 1(c) corresponding to the subregion formed by the union of r_1 and r_2;

[1]We use R_i to denote both a subregion and the dissection of that subregion in D.

such a subregion may be called a *trivial* subregion. Henceforth, by subregion we shall mean a nontrivial subregion, which may be a basic region in the extreme case.[2]

We sometimes refer to an intermediate node of $t(D)$ that has the label π as a π-node. A node with the label s or S is also referred to as a *spiral* node. If R' is a subregion, then we denote by $\pi(R')$ the top-level partitioning operation of R'. The set of walls created by $\pi' = \pi(R')$ is called the *primary walls* of R' and is denoted by $w_p(\pi')$ or $w_p(R')$. We write $w_I(R')$ for the set of all internal walls of R'; $w_I(R') \supset w_p(R')$. For an h-partition π' each wall in $w_p(\pi')$ is an h-wall, and for a v-partition π' each wall in $w_p(\pi')$ is a v-wall. If π' is a spiral partition, then $w_p(\pi')$ consists of two h-walls and two v-walls.

2.1. The partial order "<". We now define a partial ordering "<" among the walls and basic regions of an arbitrary T-plan D. The partial order "<" captures the spatial (left-to-right and north-to-south) relationships among the walls and the basic regions. It is important in the determination of the left-to-right ordering of the children of the nodes in $t(D)$.

If $w = \langle L_1, L_2\rangle$ is an arbitrary wall, $r_i \in L_1$, and $r_j \in L_2$, then we let $r_i < w < r_j$. For an h-wall w the ordering $r_i < w < r_j$ represents the vertical spatial relationships, and for a v-wall the ordering $r_i < w < r_j$ represents the horizontal spatial relationships. In general, we write $x_0 < x_k$, where each of x_0 and x_k is a wall or a basic region, if there is an alternating sequence $(x_0, x_1, \cdots, x_k)$ of basic regions and walls such that $x_0 < x_1, < \cdots x_k$. In Fig. 1(a) we have the following ordering relationships among the walls and basic regions on the south side of the wall w_1:

$$w_2 < r_1 < w_6 < r_4 < w_{12} < r_5 < w_4.$$

Similarly, among the walls and basic regions on the east side of the wall w_2 we have

$$w_1 < r_1 < w_5 < r_2 < w_7 < r_3 < w_3.$$

Note that we get the same ordering "$w_2 < w_6$" by following the south side of w_1 and by following the north side of w_3. Figure 3(c) shows the partial order "<" resulting from the left-spiral operation in Fig. 3(a), where an arc (x_i, x_j) corresponds to the ordering $x_i < x_j$ such that there is no x_k with the property $x_i < x_k < x_j$. It is easy to see that "<" defines a proper partial order.

The algorithm in [7] does not make full use of the partial order "<". Instead, it uses two subpartial orders of "<", namely, $l(r_i, r_j)$ ="r_i left of r_j" and $a(r_i, r_j)$ ="r_i above r_j", both of which were originally defined in [2]. The relation $l(r_i, r_j)$ arises as the special case of the relation "<" for which each of the walls $x_{2i+1}, i \geq 0$, in the sequence $(r_i, x_1, \cdots, x_{k-1}, r_j)$ is a v-wall, and, similarly, $a(r_i, r_j)$ arises as the special case of the relation "<" for which each of the walls x_{2i+1} is an h-wall.

2.2. The wall digraph $G_w(D)$. We now define the domination relationship among the walls and basic regions in a T-plan D. In general, the domination relationship is not a partial order. The main use of the domination relationship is in the determination of the hierarchical structure of the subregions in D, i.e., the parent–child relationships in $t(D)$.

[2]The above definition of $t(D)$ easily generalizes to an arbitrary dissection D that is generated by a family of partitioning operations Π provided these partitioning operations are independent of each other in the sense that if $\sigma \in \Pi$, then for no subset $\Pi' \subseteq \Pi - \{\sigma\}$ a dissection generated by applications of Π' equals the dissection obtained by a single application of σ.

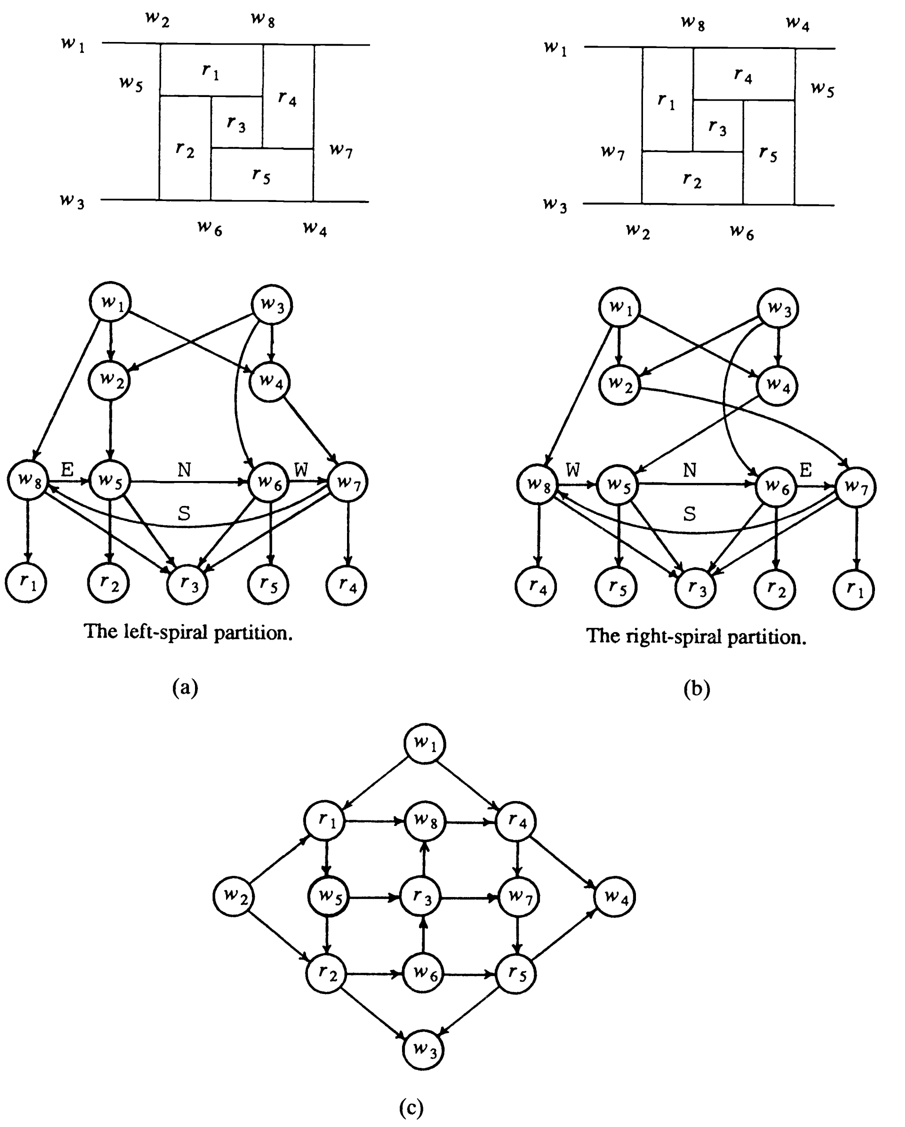


FIG. 3. *Wall graphs G_w for* (a) *left-spiral and* (b) *right-spiral showing only some of the arcs to the region vertices with the exception of r_3;* (c) *acyclic graph of the partial order "<" for the left-spiral partition.*

We say that a wall w_1 *dominates* another wall w_2 if they form a T-junction with w_2 forming the leg of the T. Put another way, if $w_1 = \langle L_{11}, L_{12}\rangle$ and $w_2 = \langle L_{21}, L_{22}\rangle$, then w_1 dominates w_2 if and only if the first members of both L_{21} and L_{22} belong to the second region lists L_{12} of w_1 or the last members of both L_{21} and L_{22} belong to the first region lists L_{11} of w_1. If w_1 is an h-wall and w_2 is a v-wall, then in the first case we say that w_1 dominates w_2 from *above* or *north* and in the second case we say that w_1 dominates w_2 from *below* or *south* (see Figs. 4(a) and (b)). Similarly, if w_1 is a v-wall and w_2 is an h-wall, then in the first case we say that w_1 dominates w_2 from *left* or *west* and in the second case we say that w_1 dominates w_2 from *right* or *east* (see Figs. 4(c) and (d)). We extend the notion of domination to that between a wall and a basic region by saying that the wall $w = \langle L_1, L_2 \rangle$ *dominates* a basic region r_i if $r_i \in L_1$ or L_2, i.e., if w is a boundary wall of r_i. If w is an h-wall, then in the first case we say that w dominates r_i from *below* or *south* and in the second case we say that w dominates r_i from *above* or *north*. The notion of domination of r_i from *left* or *west* (respectively *right* or *east*) by a v-wall is defined similarly. In Fig. 1(a) the basic region r_{10} is dominated by w_{11} from above, by w_3 from below, by w_6 from left, and by w_4 from right.

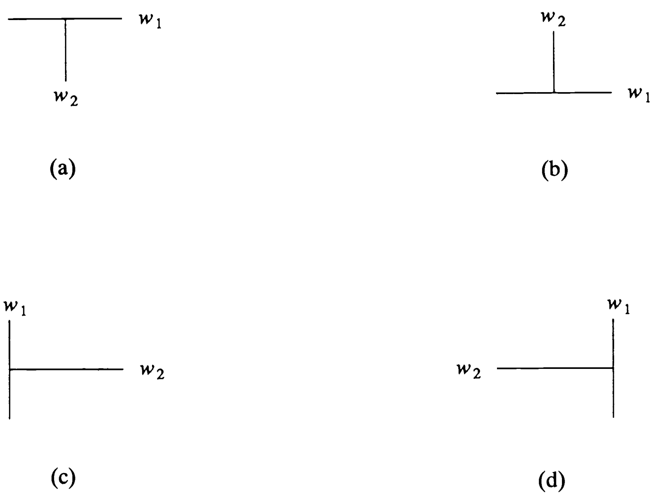


FIG. 4. *The four types of domination relationships between an h-wall and a v-wall:* (a) *the h-wall w_1 dominates the v-wall w_2 from above,* (b) *the h-wall w_1 dominates the v-wall w_2 from below,* (c) *the v-wall w_1 dominates the h-wall w_2 from the left, and* (d) *the v-wall w_1 dominates the h-wall w_2 from the right.*

We represent the domination relationships in a T-plan D as a digraph $G_w(D)$, called the *wall digraph* of D. The digraph $G_w(D)$ contains one node for each wall and basic region of D, and (w_i, w_j) or (w_i, r_j) is an arc of $G_w(D)$ if w_i dominates w_j or w_i dominates r_j. We label each arc (w_i, w_j) and (w_i, r_j) by one of the symbols $\{N, S, W, E\}$ to indicate the direction of domination by w_i. These labels are used primarily for analyzing the spiral partitions, and therefore we often show these labels only for the arcs that join two primary walls of a spiral partition. (We show in Lemma 2 in §3 that these are the only walls that belong to cycles in $G_w(D)$.) A vertex in $G_w(D)$ that corresponds to a wall is called a *wall vertex*, and a vertex that corresponds to a basic region is called a *region vertex*. The wall vertices w_1 and w_3 in $G_w(D)$, corresponding to the external north and south walls, have no incoming arcs and are the only *source nodes* in $G_w(D)$. All other wall vertices in $G_w(D)$, including w_2 and w_4, which correspond to the external west and east walls, have exactly two incoming arcs. Each region vertex r_j in $G_w(D)$ has, on the other hand, four incoming arcs and no outgoing arcs. Figure 5 shows the digraph $G_w(D)$ for the T-plan in Fig. 1(a), where we have shown only some of the arcs to the region vertices r_i, with the exception of r_1 and r_{10} to keep the diagram simple. The wall digraphs for the two spiral partitions are shown in Figs. 3(a) and (b); once again, we show only some of the arcs to the region vertices, with the exception of the central region r_3. Note that the labels of the arcs along the cycle are (N, W, S, E) for the left spiral and (N, E, S, W)

for the right spiral. We use this difference to distinguish a left spiral from a right spiral. If D has n basic regions, then the number of internal walls in D is $n-1$ and hence the digraph $G_w(D)$ has $4+(n-1)+n=2n+3$ nodes and $4+2(n-1)+4n=6n+2$ arcs. If x and y are two nodes in $G_w(D)$, we say that y is *reachable* from x if $y=x$ or there is a (directed) path from x to y.

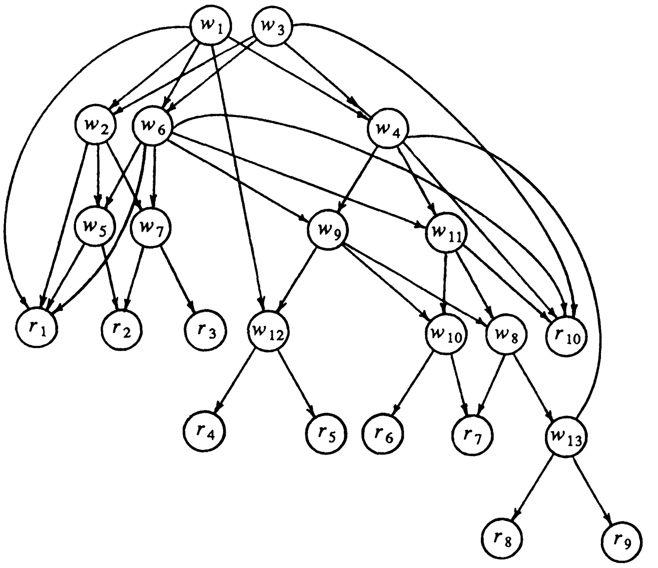


FIG. 5. *The wall graph $G_w(D)$ of the dissection in Fig.* 1(a); *to keep the diagram simple, only some of the arcs to the region vertices r_j, except for r_1 and r_{10}, are shown here.*

If a dissection D is not a T_*-plan, then the digraph $G_w(D)$ may contain cycles with an arbitrarily large number of nodes in it; some examples of such T-plans are given in Fig. 10 and in [4]. It is easy to see that if D is an acyclic T_*-plan and (w_i, w_j) is an arc in $G_w(D)$, then there is a unique subregion R' of R such that w_i is a boundary wall of R' and $w_j \in w_p(R')$; we write $R' = \rho(w_i, w_j)$. In Fig. 1(a) we have $\rho(w_6, w_9) = \{r_4, r_5, \cdots, r_{10}\}$, the subregion consisting of the basic regions $r_4, r_5, \cdots$, and r_{10}; $\rho(w_1, w_{12}) = \{r_4, r_5\}$. For the cyclic T_*-plans in Figs. 3(a) and (b), there is no such R' corresponding to the arc (w_8, w_5).

If w is an h-wall, then let

$$\delta_N(w) = \{x : x \text{ is a } v\text{-wall or a basic region dominated by } w \text{ from north}\},$$

and

$$\delta_S(w) = \{x : x \text{ is a } v\text{-wall or a basic region dominated by } w \text{ from south}\}.$$

For a v-wall w the sets $\delta_W(w)$ and $\delta_E(w)$ are defined similarly. We regard each of the sets $\delta_N(w), \delta_S(w), \delta_W(w)$, and $\delta_E(w)$ as linearly ordered according to "$<$" and refer to them as lists. The items in each list alternate between a wall and a basic region. For an h-wall w the set of nodes adjacent *from* the node w is given by adj$[w] = \{x : (w, x)$ is an arc in $G_w(D)\} = \delta_S(w) \cup \delta_N(w)$, which is a disjoint union. We thus write adj$[w] =$

$\langle \delta_S(w), \delta_N(w)\rangle$. Similarly, for a v-wall w we have adj$[w] = \langle \delta_E(w), \delta_W(w)\rangle$. For each basic region r_j, adj$[r_j] = \oslash$ by the definition of $G_w(D)$. Figure 6(a) shows the adjacency lists adj$[w]$ for the T-plan in Fig. 1(a). We use the notation adj$[w] = \langle \delta_1(w), \delta_2(w)\rangle$, where $\delta_1(w) = \delta_S(w)$ or $\delta_E(w)$ and $\delta_2(w) = \delta_N(w)$ or $\delta_W(w)$, when we do not wish to emphasize whether w is an h-wall or a v-wall. The number of arcs that are incident to a node x in $G_w(D)$ is denoted by indeg$[x]$.

$adj[w_1] = \langle \oslash, (w_2, r_1, w_6, r_4, w_{12}, r_5, w_4)\rangle$
$adj[w_2] = \langle \oslash, (r_1, w_5, r_2, w_7, r_3)\rangle$
$adj[w_3] = \langle (w_2, r_3, w_6, r_{10}, w_4), \oslash\rangle$
$adj[w_4] = \langle (r_5, w_9, r_8, w_{13}, r_9, w_{11}, r_{10}), \oslash\rangle$
$adj[w_5] = \langle (r_1), (r_2)\rangle$
$adj[w_6] = \langle (r_1, w_5, r_2, w_7, r_3), (r_4, w_9, r_6, w_{11}, r_{10})\rangle$
$adj[w_7] = \langle (r_2), (r_3)\rangle$
$adj[w_8] = \langle (r_7), (r_8, w_{13}, r_9)\rangle$
$adj[w_9] = \langle (r_4, w_{12}, r_5), (r_6, w_{10}, r_7, w_8, r_8)\rangle$
$adj[w_{10}] = \langle (r_6), (r_7)\rangle$
$adj[w_{11}] = \langle (r_6, w_{10}, r_7, w_8, r_9), (r_{10})\rangle$
$adj[w_{12}] = \langle (r_4), (r_5)\rangle$
$adj[w_{13}] = \langle (r_8), (r_9)\rangle$

(a)

The wall w processed	The list L for the node-groups formed after processing $\delta_1(w) = \delta_S(w)$ or $\delta_E(w)$	after processing $\delta_2(w) = \delta_N(w)$ or $\delta_W(w)$
w_6	(w_5, w_7)	(w_9, w_{11})
w_5 w_7	(r_1) (r_2)	L = empty (r_3)
w_9 w_{11}	(w_{12}) (w_{10}, w_8)	L = empty (r_{10})
w_{12}	(r_4)	(r_5)
w_{10} w_8	(r_6) (r_7)	L = empty (w_{13})
w_{13}	(r_8)	(r_9)

(b)

FIG. 6. *Illustration of Algorithm* 1 *for the T-plan in Fig.* 1(a): (a) *adjacency lists adj*$[w]$ *for the wall vertices in the digraph* $G_w(D)$ *for the T-plan in Fig.* 1(a); (b) *formation of successive node groups by Algorithm* 1. *The walls in* (b) *are processed in the order shown in the first column*; *the root node corresponds to the node group* $L = (w_6)$.

If G is an acyclic digraph, then the *transitive reduction* of G is the (unique) minimal digraph $G' \subseteq G$ such that G' and G have the same nodes and there is a path from a node x to a node y in G if and only if there is a path from x to y in G'. If G contains cycles, then there may be more than one distinct minimal subdigraph G' as defined above. However, if no two cycles in G have a node in common, then G' is unique.

The fundamental observation behind the algorithm presented here is that if the top-level partition $\pi(R')$ of a subregion R' is an h or v partition, then the source nodes in the subdigraph $G_I(R') \subset G_w(D)$ on the internal walls and basic regions in R' (excluding the vertices for the boundary walls of R') consist of the primary walls $w_p(R')$ created by the top-level partition operation $\pi(R')$ in R'. The situation is slightly more complex if $\pi(R')$ is a spiral partition. In this case we do not have any source nodes in the subdigraph $G_I(R')$. Instead, we have a "source cycle" formed by the set of walls $w_p(R')$ such that the only arc (w_i, w_j) in $G_I(R')$ to a wall $w_j \in w_p(R')$ is from another wall $w_i \in w_p(R')$. Fortunately, the source cycles in the various $G_I(R')$ are precisely the cycles in $G_w(D)$; moreover, they are easily identified because they are disjoint from each other (see Lemma 2, §3). Identifying the subsets of nodes of the form $w_p(R')$ without constructing the subregions R' themselves is also possible by exploiting certain special properties of the wall digraph $G_w(D)$ that are described in §3.

3. Properties of wall digraph $G_w(D)$. The following lemmas summarize some of the important properties of the digraph $G_w(D)$ for a T_*-plan D. These properties form the basis of the algorithms given in §§4 and 5 for the construction of $t(D)$.

LEMMA 1. *If w_i is a boundary wall of a subregion R', then there is a path from w_i to each internal wall w_j of R'. Also, if w_i is an internal wall of R', then the paths from w_i to other vertices in $G_w(D)$ are limited to only the internal walls and basic regions of R'.*

Proof. The second part of the lemma is immediate from the fact that for any arc (w_i, w_k) from an internal wall $w_i \in w_I(R')$, w_k is necessarily an internal wall of R' and, in particular, w_k is not a boundary wall of R'. If (w_i, r_j) is an arc from the internal wall w_i of R', then r_j is necessarily a basic region of R'.

We prove the first part of the lemma by induction on the number of partition operations $m \geq 1$ applied to R'. If w_j is one of the primary walls in $w_p(R')$ created by the top-level partition operation $\pi(R')$ of R' and $\pi(R') = h$ or v, then either (w_i, w_j) is an arc in $G_w(D)$ or there is another boundary wall w_k of R' such that (w_i, w_k) and $(w_k, w_j) \in G_w(D)$. The latter occurs, for example, if R' is created by an h (respectively, v) partition and $\pi(R') = v$ (respectively, h). On the other hand, if $\pi(R') = s$ or S, then there is a path of length ≤ 4 from w_i to each primary wall $w_j \in w_p(R')$. Thus the lemma is true for $m = 1$. Now suppose that the lemma is true for subregions having fewer than $m \geq 2$ partition operations, and suppose that R' has m partition operations. If $w_j \in w_I(R') - w_p(R')$, then one of the subregions $R'' \subset R'$ created by $\pi(R')$ contains w_j as an internal wall. Let $w' \in w_p(R')$ be a boundary wall of R''. Then w' is reachable from w_i (by the case $m = 1$), and w_j is reachable from w' by the induction hypothesis. Hence there is a path from w_i to w_j. □

COROLLARY 1. *The digraph $G_w(D)$ is connected as an undirected graph.*

Proof. Each vertex in $G_w(D)$ is reachable by an undirected path from the node w_1, say. □

LEMMA 2. *If D is an acyclic T_*-plan, then $G_w(D)$ is an acyclic graph. If D is not acyclic, then each cycle of $G_w(D)$ corresponds to the set of primary walls $w_p(R')$ for some subregion R', where $\pi(R') = s$ or S, and the converse is also true. No two-cycles in $G_w(D)$ have a node in common.*

Proof. Clearly, any cycle of $G_w(D)$ involves only the wall vertices. In view of Lemma 1, if $G_w(D)$ contains a cycle that involves w_i and if w_i is a boundary wall of a subregion R', the cycle must not involve any internal wall of R'. Thus the only way that a cycle may be formed is by the boundary walls of the subregions created by a single partition, i.e., the set of primary walls $w_p(R')$ for some (R'), where $\pi(R') = s$ or S. The converse and disjointness of the cycles are now immediate. □

LEMMA 3. *Let D be an acyclic T_*-plan. If the wall w_i dominates the walls w_j and w_k from two different directions (i.e., below and above or left and right), then there is no wall w' such that there is a path from w_i to w' through w_j and a path from w_i to w' through w_k.*

Proof. Let $R_1 = \rho(w_i, w_j)$ and $R_2 = \rho(w_i, w_k)$. The subregions R_1 and R_2 lie on two different sides of the wall w_i. Since w_j is an internal wall of R_1, it follows from Lemma 1 that a path leading from the arc (w_i, w_j) cannot reach any wall that is not internal to R_1. A similar property holds for the paths leading from the arc (w_i, w_k) and the subregion R_2. This proves the lemma. □

Lemma 3 may not be true, in general, for T_*-plans containing spiral partitions. For instance, in Fig. 7(a) the wall w_{14} dominates the walls w_{17} and w_{19} from two different directions, but $p = (w_{14}, w_{17}, w_{10})$ is a path from w_{14} to w_{10} via the arc (w_{14}, w_{17}) and $p' = (w_{14}, w_{19}, w_{18}, w_5, w_6, w_{17}, w_{10})$ is a path from w_{14} to w_{10} via the arc (w_{14}, w_{19}).

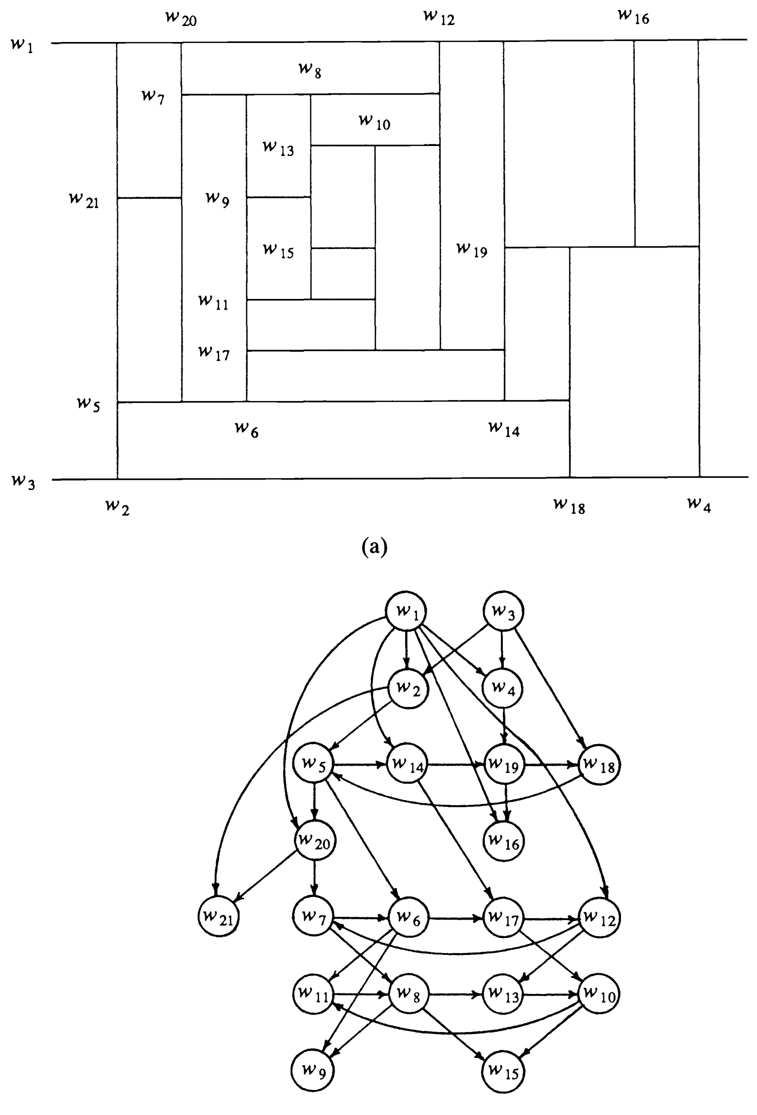


FIG. 7. (a) *A T_*-plan D with nested spiral partitions and its wall digraph $G_w(D)$*; (b) *a part of the digraph $G_w(D)$ for the T_*-plan D in* (a); *only the wall vertices and the arcs among them are shown.*

LEMMA 4. *Let D be an acyclic T_*-plan. Then the arcs in the transitive reduction of $G_w(D)$ are given by the following*:

(1) $(w_1, w_2), (w_1, w_4), (w_3, w_2)$, *and* (w_3, w_4);

(2) $(w_i, w_j) \in G_w(D)$, *where w_i is a boundary h-wall of a subregion R' such that $\pi(R') = v$ and $w_j \in w_p(R')$*;

(3) $(w_i, w_j) \in G_w(D)$, *where w_i is a boundary v-wall of a subregion R' such that $\pi(R') = h$ and $w_j \in w_p(R')$*;

(4) (w_i, r_j), *where $w_i = \langle L_1, L_2\rangle$ and $L_1 = (r_j)$ or $L_2 = (r_j)$.*

Proof. Let $G'(D)$ denotes the transitive reduction of $G_w(D)$. That the arcs in group (1) are the only arcs to w_2 and w_4 in $G'(D)$ is easily verified. Now assume that w_i is an

h-wall and $w_j \in w_p(R')$ as in (2). We need to show that there is no other path from w_i to w_j. If possible, let $p = (w_i, w_k, \cdots, w_j)$ be a path from w_i to w_j. By Lemma 3 w_k must be on the same side of w_i as w_j, and by Lemma 1 w_k must be internal to R'. If $w_k \notin w_p(R')$, then w_k is an internal wall of a subregion $R'' \subset R'$ created by the partition operation $\pi(R')$, and thus the path p cannot reach w_j, which is not internal to R''. Thus $w_k \in w_p(R')$. However, this means that the path p enters a subregion $R'' \subset R'$ following the arc (w_i, w_k) and that w_j is not an internal wall of R''. Once again, the path p cannot continue to w_j. Thus no path $p = (w_i, w_k, \cdots, w_j)$ exists, and hence $(w_i, w_j) \in G'(D)$. The proof for group (3) is similar. On the other hand, if $(w_i, w_j) \in G_w(D), w_j \notin w_p(R')$, and $w_j \neq w_2$ or w_4, then there is a subregion $R'' \subset R'$ such that w_j is an internal wall of R'' and by Lemma 1 there is a wall $w_k \in w_p(R')$ and a path from w_i to w_j via the arc (w_i, w_k). This shows that the only arcs (w_i, w_j) in $G'(D)$ are those in groups (1)–(3).

Finally, consider the arc (w_i, r_j), where $L_1 = (r_j)$ or $L_2 = (r_j)$. Let $(w_i, w_k) \in G_w(D)$. Then r_j and w_k are on two different sides of the wall w_i. If we write $\rho(w_i, w_k) = R''$, then a path leading from the arc (w_i, w_k) cannot reach any basic region that is outside R'' and hence cannot reach r_j. This shows that $(w_i, r_j) \in G'(D)$. On the other hand, if $r_j \in L_1$ and $L_1 \neq (r_j)$, then there is a wall w_j such that both the arcs (w_i, w_j) and (w_j, r_j) belong to $G_w(D)$ and hence $(w_i, r_j) \notin G'(D)$. Similar reasoning applies for the case $r_j \in L_2$ and $L_2 \neq (r_j)$. □

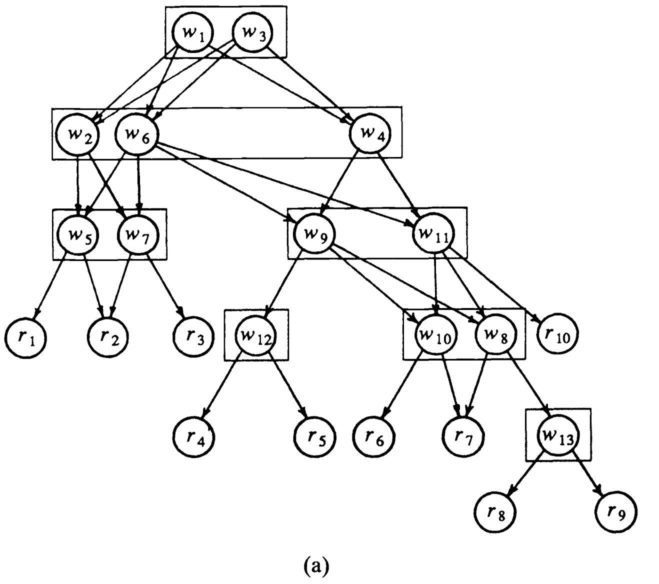
4. Construction of $t(D)$ for an acyclic T_*-plan. The following theorem shows the relationship between $t(D)$ and $G_w(D)$ for an acyclic T_*-plan D and forms the basis of Algorithm 1 for the construction of $t(D)$ from the wall representation $w(D)$.

THEOREM 1. *Let D be an arbitrary acyclic T_*-plan, and let $G_I(D) = G_w(D) - \{w_1, w_2, w_3, w_4\}$ be a subdigraph that shows the domination relationships among the internal walls and basic regions of D. Then the subregion tree $t(D)$ is isomorphic as an unordered tree to the digraph obtained from the transitive reduction of $G_I(D)$ by merging each node group of primary walls $w_p(R')$, where R' is a subregion, into a single node.*

Proof. If D consists of a single basic region, then there is nothing to prove. Suppose that D has two or more basic regions. It is easy to see that the source nodes of $G_I(D)$ are the same as the set of primary walls $w_p(R) = \{w_{j_1}, w_{j_2}, \cdots, w_{j_{k-1}}\}$, say, that are created by the top-level partition operation $\pi(R)$ corresponding to the root node of $t(D)$. By Lemma 1 it follows that the removal of nodes $w_p(R)$ from $G_I(D)$ decomposes $G_I(D)$ into k disconnected components (as an undirected graph), one for each of the subregions $\{R_1, R_2, \cdots, R_k\}$ created by $\pi(R)$. (The wall w_{j_i} may be regarded as the common boundary wall between R_i and R_{i+1}.) The connected component of $G_I(D)$ corresponding to the subregion R_j equals $G_I(R_j), 1 \leq j \leq k$. The theorem now follows by a simple induction argument applied to each R_j. □

Figure 8(a) shows the transitive reduction of the wall digraph $G_w(D)$ for the T-plan in Fig. 1(a); a part of $G_w(D)$ was shown earlier in Fig. 5. Each node group of the form $w_p(R')$ is shown here enclosed in a rectangular box. The node groups $w(R')$ are easily identified successively as follows. Initially, the external walls $\{w_1, w_3\}$ form the source nodes of $G_w(D)$. The nodes $\{w_1, w_3\}$ may be thought of as the primary walls of a hypothetical h-partition of the whole infinite plane (without any boundary lines) creating the three subregions N, S, and the area $R^+ = R \cup W \cup E$ between the walls w_1 and w_3. The removal of $\{w_1, w_3\}$ from $G_w(D)$ (see Fig. 5) gives rise to the source nodes $\{w_2, w_4, w_6\}$, which consist of the external wall $\{w_2, w_4\}$ and the node group $w_6(R) = \{w_6\}$. Here the walls $\{w_2, w_4, w_6\}$ may be thought of as the primary walls of an extended v-partition of R^+ corresponding to $\pi(R) = v$. If $\pi(R)$ were an h-partition, then $\{w_2, w_4\}$ would be the only source nodes in $G_w(D) - \{w_1, w_3\}$ and the source nodes in $G_I(D) =$

$G_w(D) - \{w_1, w_2, w_3, w_4\}$ would equal the set of primary walls $w_p(R)$. Each successive node group is obtained by removing the current node group and taking the source nodes of a connected component of the resulting digraph.[3] Figure 8(b), save the labels h and v for the nonterminal nodes, shows the tree obtained by merging each node group in the transitive reduction of $G_I(D)$ into a distinct single node. This tree is isomorphic to the tree $t(D)$ in Fig. 1(c) as an unordered tree.



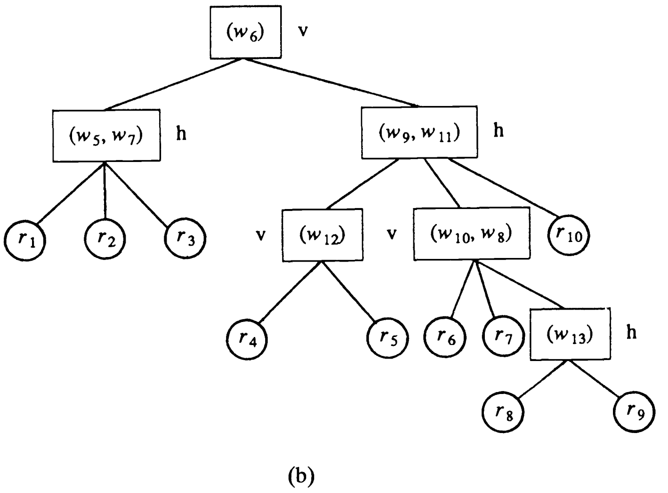


FIG. 8. (a) *The transitive reduction of $G_w(D)$ for the T-plan in Fig.* 1(a); *each node group consists of the primary walls $w_p(R')$ for some subregion R'.* (b) *The result of merging each node group $w_p(R')$ into a single node after deleting the nodes $\{w_1, w_2, w_3, w_4\}$ in the transitive reduction. The tree is identical to the tree in Fig.* 1(c) *except for the h and v labels of the nonterminal nodes.*

[3]The method of forming the successive node groups in this way may be likened to the topological sorting algorithm in [1]. It essentially amounts to computing the transitive reduction of $G_I(D)$ because of the special structure of $G_I(D)$.

To complete the construction of the tree $t(D)$, we now show how to determine the ordering of the children of each node in $t(D)$ by using the ordering "$<$" together with the digraph $G_w(D)$. We argue below the case for the root node only; the same argument holds for all other nodes of $t(D)$ as well. Let $\{R_1, R_2, \cdots, R_k\}$ be the subregions created by $\pi(R)$, in the left-to-right or top-to-bottom order, according to whether $\pi(R) = v$ or h. If w_{j_i} is the wall between subregions R_i and R_{i+1}, then $w_{j_i} < w_{j_2} < \cdots w_{j_{k-1}}$ and $w_p(R) = \{w_{j_1}, w_{j_2}, \cdots, w_{j_{k-1}}\}$. If we disconnect the node w_{j_1} from each node in $\delta_1(w_{j_1}) \subset adj[w_{j_1}]$, then in view of Lemmas 1 and 3 we have that $G_I(R_1)$ becomes isolated as one connected component and the second component consists of the union of $w_p(R)$ and $G_I(R_i), 2 \leq i \leq k$. The component $G_I(R_1)$ is identified as the one that does not contain w_{j_1}. We now complete the deletion of w_{j_1} from $G_I(R)$ by disconnecting w_{j_1} from the remaining nodes $\delta_2(w_{j_1})$ in $adj[w_{j_1}]$ and by removing w_{j_1}. This, however, causes no further decomposition of the second component into two or more components if $k \geq 2$.[4] The process now continues by first disconnecting w_{j_2} from the nodes in $\delta_1(w_{j_2})$ and isolating the component $G_I(R_2)$ as the one that does not contain w_{j_2}. The deletion of w_{j_2} is then completed by disconnecting it from the nodes in $\delta_2(w_{j_2})$ and removing w_{j_2} and so on. The last component remaining after the removal of $w_{j_{k-1}}$ is the component $G_I(R_k)$. This completes the determination of the ordering of the children of the root node, which correspond to the subregions $R_i, 1 \leq i \leq k$. In Algorithm 1 below, we determine only the source nodes in the various components R_i in the order $i = 1, 2, \cdots, k$ instead of determining the whole components R_i. This helps to minimize the computation and keep the overall computation at the level $O(n)$.

Algorithm 1 constructs the subregion tree $t(D)$ for an acyclic T_*-plan D in a top-down, breadth-first fashion by successively identifying the various node groups of the form $w_p(R')$ as described above. It assumes that the linear ordering "$<$" of each of the lists $\delta_1(w_j)$ and $\delta_2(w_j)$ has already been determined. It is from these orderings that we directly obtain the ordering $w_{j_1} < w_{j_2} < \cdots < w_{j_{k-1}}$ of the primary walls $w_p(R')$ in each node group, without having to order the set $w_p(R') = \{w_{j_1}, w_{j_2}, \cdots, w_{j_{k-1}}\}$ separately for each R' after the determination of $w_p(R')$. The advantage of using the orderings of $\delta_1(w_j)$ and $\delta_2(w_j)$ is that they can be determined for all walls w_j by using only $O(n)$ time (see §6). Algorithm 1 uses a queue Q to hold the source-node groups that have been identified but that have not been deleted yet from $G_w(D)$. In terms of the tree $t(D)$, these node groups correspond to the nodes whose children are yet to be determined. (The use of a stack instead of a queue results in constructing the tree $t(D)$ in the depth-first fashion.) Each item in Q is a pair $(p(z), L(z))$, where $p(z)$ is a pointer to the node z in $t(D)$ and $L(z)$ = the node group at z in the order "$<$". The list $L(z)$ consists of either a basic region or the set of primary walls $w_p(R')$ for some subregion R'. The pointer $p(z)$ is used for connecting the children of z to the node z. When an arc (w_i, w_j) is deleted from $G_w(D)$, we label w_j as an h-wall or a v-wall according to whether w_i is a v-wall or an h-wall. Initially, the external north and south walls (w_1 and w_3) are labeled as h-walls. The h–v labels are maintained in an array label$[w_j]$. A node in $t(D)$ is labeled h or v according to whether the walls in that node group are h-walls or v-walls.

[4]To see this, one may argue as follows. If R_2 is a basic region, then it is adjacent from both w_{j_1} and w_{j_2}. On the other hand, if R_2 is not a basic region, then the primary walls $w_p(R_2)$ of R_2 are adjacent from both w_{j_1} and w_{j_2} and all basic regions and internal walls in R_2 can be reached from w_{j_2} via the primary walls $w_p(R_2)$.

ALGORITHM 1

Construction of $t(D)$ from $w(D)$ of an acyclic T_*-plan D.

```
Input:   The set of walls $w(D)$ of an acyclic $T_*$-plan $D$.

Output:  The subregion tree $t(D)$ of $D$.
1. If there are only four walls in $w(D)$, then $D$ consists of a single basic region $r_j$
   and $t(D)$ consists of a single node $r_j$. Otherwise, for each wall $w$ determine the
   adjacency lists in the form adj$[w] = \langle\delta_1(w), \delta_2(w)\rangle$, including the ordering "<"
   in each of the lists $\delta_1(w)$ and $\delta_2(w)$. Let indeg$[w_j] = 2$ for each wall $w_j$ except
   for indeg$[w_1] = 0 =$ indeg$[w_3]$, and let indeg$[r_j] = 4$ for each basic region $r_j$.
   (Here $w_1$ is the external north wall and $w_3$ is the external south wall.)
2. Let $w_2$ and $w_4$ be the external west and east walls, respectively.
3. Remove $w_1$ and $w_3$ from $G_w(D)$, and adjust indeg$[x]$ appropriately for each
   vertex $x$ that is adjacent from one or both of $w_1$ and $w_3$.
4. (Determine the label of the root node of $t(D)$.) Let $L_0 =$ the list of the source
   nodes other than $w_2$ and $w_4$, if any. Remove the nodes $w_2$ and $w_4$, and adjust
   indeg$[x]$ for all vertices that are adjacent to one or both of $w_2$ and $w_4$. Perform
   step (a) or (b) as appropriate.
   (a) If $L_0 \neq \oslash$, then the label of the root node of $t(D)$ is $v$. In this case $L_0 \subset$
       $\delta_2(w_1)$; assume that $L_0$ is ordered according to $\delta_2(w_1)$.
   (b) If $L_0 = \oslash$, then the label of the root node of $t(D)$ is $h$; let $L_0 =$ the set of
       new source nodes after removal of $w_2$ and $w_4$. In this case $L_0 \subset \delta_2(w_2)$;
       assume that $L_0$ is ordered according to $\delta_2(w_2)$.
5. Create the root node $z$ of $t(D)$ with the associated node group $L(z) = L_0$.
   Initialize the queue $Q$ with the item $(p(z), L(z))$, where $p(z)$ is a pointer to the
   node $z$.
6. While $Q$ is nonempty do (a) and (b):
   (a) Remove the first item $(p(z), L(z))$ from $Q$.
   (b) Process each wall $w \in L(z)$ according to the order "<" in $L(z)$ as follows,
       where adj$[w] = \langle\delta_1(w), \delta_2(w)\rangle$.
       (i) For each $x \in \delta_1(w)$ reduce indeg$[x]$ by one. Let $L \subseteq \delta_1(w)$ be the list
           of new source vertices in the order as in $\delta_1(w)$.
       (ii) If $L \neq \oslash$, then create a new node $z'$ in the tree $t(D)$ with the associated
           node group $L(z') = L$; make $z'$ the current rightmost child of the node
           $z$ using the pointer $p(z)$. If $L$ is not a single basic region, then add the
           item $(p(z'), L(z'))$ to the end of $Q$ and let label$[z']$ be the same as the
           labels of the nodes in $L$ (all nodes in $L$ have the same label, $v$ or $h$).
       (iii) Repeat steps (i) and (ii) using $\delta_2(w)$ in place of $\delta_1(w)$. (In this case $L$
           is empty except when $w$ is the last item in $L(z)$; in processing $\delta_1(w)$, $L$
           is always nonempty.)
```

Fig. 6(b) illustrates Algorithm 1 and shows the various node groups L formed for the set of walls in Fig. 1(b). The final tree obtained by the algorithm is the same as that in Fig. 1(c).

5. Construction of $t(D)$ for an arbitrary T_*-plan. Suppose now that the T_*-plan D contains one or more spiral partitions. As we noted in §2, if the partition operation $\pi(R') = s$ or S, then the digraph $G_I(R') \subset G_w(D)$ on the internal walls and basic regions of R' does not contain a source node because the primary walls $w_p(R')$ form a cycle C. By Lemma 1, cycle C has the distinguishing property that no internal wall in

$w_I(R')$ dominates a wall in C. In this sense we may say that cycle C is a "source cycle" in $G_I(R')$. We determine all cycles in $G_w(D)$ by a single depth-first traversal [1] of $G_w(D)$ starting at the external north wall w_1, say. Since the region vertices have no outgoing arcs, it suffices to traverse only the arcs to wall vertices. Also, because the cycles are disjoint (Lemma 2), each cycle is formed by a back arc (w_i, w_j) and the path from w_j to w_i in the depth-first tree. We assume that the arcs (w_i, w_j) from a wall vertex w_i are processed in the depth-first search first for $w_j \in \delta_1(w_i)$ and then for $w_j \in \delta_2(w_i)$, in both cases following the ordering "$<$" in the lists $\delta_1(w_i)$ and $\delta_2(w_i)$. Figure 9(a) shows the depth-first tree for the wall digraph of the T_*-plan D in Fig. 7(a) which contains three spiral partitions, three h-partitions, and two v-partitions. Here, one of the spiral partitions is nested inside another one and the third spiral partition is disjoint from the first two. The three cycles corresponding to the spiral partitions are given by

$$\begin{aligned} C_1 &= (w_{19}, w_{18}, w_5, w_{14}), && \text{a right spiral,} \\ C_2 &= (w_7, w_6, w_{17}, w_{12}), && \text{a left spiral,} \\ C_3 &= (w_{13}, w_{10}, w_{11}, w_8), && \text{a right spiral.} \end{aligned}$$

The following theorem (similar to Theorem 1) is now easily proved by using the lemmas in §2 and is stated here without proof. Note that the special properties of $G_w(D)$, including the disjointness of its cycles, imply that the transitive reduction of $G_I(D) = G_w(D) - \{w_1, w_2, w_3, w_4\}$ is uniquely defined.

THEOREM 2. *Let D be an arbitrary T_*-plan acyclic or not. Then the subregion tree $t(D)$ is isomorphic as an unordered tree to the digraph obtained from the transitive reduction of $G_I(D) = G_w(D) - \{w_1, w_2, w_3, w_4\}$ by merging each node group $w_p(R')$, where R' is a subregion, into a single node.*

We are now ready to present Algorithm 2 for constructing the tree $t(D)$ from $w(D)$ for a general T_*-plan D that may contain one or more spiral partitions. We regard a cycle C_j as a *composite* vertex consisting of the vertices $w_i \in C_j$. For each cycle C_j we define indeg$[C_j] = 4$, which is simply the number of arcs from nodes not in C_j to the nodes in C_j. We augment the array indeg$[x]$ for the wall and region vertices x in $G_w(D)$ by adding an entry indeg$[C_j]$ for each cycle C_j in $G_w(D)$. When an arc $(w_i, w_k), w_i \notin C_j$ and $w_k \in C_j$, is deleted from $G_w(D)$, we reduce indeg$[C_j]$ by one; we say the cycle C_j is adjacent from w_i. The test $w_k \in C_j$ is performed by using an additional array cycle$[w_k]$, which gives for each wall vertex w_k the unique cycle in $G_w(D)$ containing w_k, if any (if w_k does not belong to any cycle, then we let cycle$[w_k]$ = nil). We assume that each cycle C_j is written in the *standardized* form, which is defined to be $C_j = (w_a, w_b, w_c, w_d)$, where the vertices are listed in the cyclic order and the wall vertex w_a dominates w_b from north. We define label$(C_j) = s$ or S according to whether the labels of the arcs along the cycle are (N, W, S, E) or (N, E, S, W). As in Algorithm 1, on processing an arc (w_i, w_j) we identify the wall w_j as an h-wall or a v-wall according to whether w_i is a v-wall or an h-wall.

The main difference between Algorithm 1 and Algorithm 2 lies in the processing of the vertices in a source cycle C. The processing of C [i.e., the deletion of the nodes in C from $G_w(D)$] generates five separate node lists $L_k, 1 \leq k \leq 5$, corresponding to the five regions formed by the associated spiral partition. If x is a cycle, then the test "$x < w_j$" in steps 6(c)(i) and (ii) is considered to be true if a vertex w_i belonging to the cycle x satisfies $w_i < w_j$ (in which case $w_i < w_j$ holds for all vertices w_i in the cycle x).

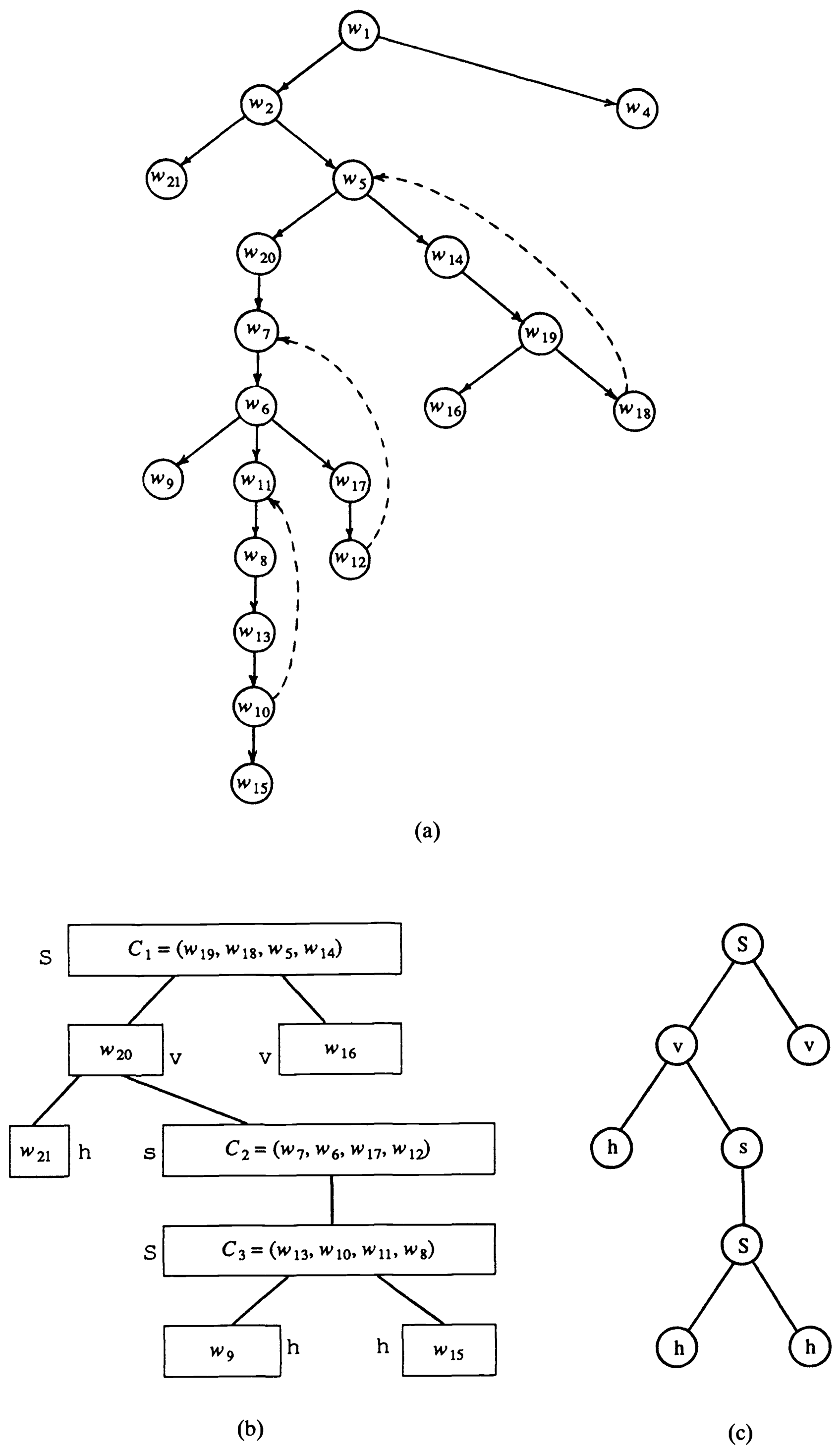


FIG. 9. *Illustration of Algorithm 2 for the T_*-plan in Fig.* 7(a): (a) *Depth-first traversal of the wall vertices of $G_w(D)$ shown in Fig.* 7(b) *with the back arcs shown in broken lines and vertex w_3 not visited*; (b) *node groups L constructed by Algorithm* 2 *corresponding to the nonterminal nodes of $t(D)$ and the arcs among them in the transitive reduction*; (c) *nonterminal nodes in the tree $t(D)$*.

ALGORITHM 2

Determination of $t(D)$ from $w(D)$ for an arbitrary T_*-plan D.

Input: The set of walls $w(D)$ of an arbitrary T_*-plan D, which may contain spiral partitions.

Output: The subregion tree $t(D)$ of D.

1. If there are only four walls in $w(D)$, then $t(D)$ consists of a single node $r_j = D$. Otherwise, for each wall w determine the adjacency lists in the form adj$[w] = \langle\delta_1(w), \delta_2(w)\rangle$, including the ordering "$<$" in each of the lists $\delta_1(w)$ and $\delta_2(w)$. Let indeg$[w_j] = 2$ for each wall w_j except for indeg$[w_1] = 0 =$ indeg$[w_3]$, and let indeg$[r_j] = 4$ for each basic region r_j.
2. Let w_1, w_2, w_3, and w_4 be the external north, west, south, and east walls, respectively. Determine the cycles C in $G_w(D)$ by a depth-first traversal starting from the node w_1. Let indeg$[C] = 4$ for each cycle, and compute the array cycle$[w_j]$ by scanning each cycle C once.
3. Remove w_1 and w_3 from $G_w(D)$, and adjust indeg$[x]$ for all vertices and cycles x that are adjacent from one or both of w_1 and w_3.
4. (Determine the label of the root node of $t(D)$.) Let $L_0 =$ the list of the source nodes other than w_2 and w_4, if any. Remove the nodes w_2 and w_4, and adjust indeg$[x]$ for all vertices and cycles x. Perform one of the steps (a)–(c) as appropriate.
   (a) If $L_0 \neq \oslash$, then the label of the root node of $t(D)$ is v. In this case $L_0 \subset \delta_2(w_1)$; assume that L_0 is ordered according to $\delta_2(w_1)$.
   (b) If $L_0 = \oslash$ and there is a wall w_i that is a source node (after removal of w_2 and w_4), then the label of the root node of $t(D)$ is h; let $L_0 =$ the set of source nodes after removal of w_2 and w_4. In this case $L_0 \subset \delta_2(w_2)$; assume that L_0 is ordered according to $\delta_2(w_2)$.
   (c) Otherwise, there is a unique source cycle $C = (w_{j_1}, w_{j_2}, w_{j_3}, w_{j_4})$, in the standard notation, with indeg$[C] = 0$. Let $L_0 = C$, and let the label of the root node of $t(D)$ be label(C).
5. Create the root node z of $t(D)$ with the associated node group $L(z) = L_0$. Initialize the queue Q with the item $(p(z), L(z))$, where $p(z)$ is a pointer to the node z.
6. While Q is nonempty do (a) and (b) or (a) and (c) as appropriate:
   (a) Remove the first item $(p(z), L(z))$ from Q.
   (b) If $L(z)$ is not the vertices of a cycle or, equivalently, if it contains only h-walls or only v-walls, then process each wall $w \in L(z)$ according to the order "$<$" by (i)–(iii), where adj$[w] = \langle\delta_1(w), \delta_2(w)\rangle$.
      (i) For each $x \in \delta_1(w)$ reduce indeg$[x]$ by one and reduce indeg$[C]$ by one, where $C =$ cycle$[x]$, if any. Let L be the new source cycle, if any, or let $L \subseteq \delta_1(w)$ be the list of new source vertices in the order as in $\delta_1(w)$.
      (ii) If $L \neq \oslash$, then create a new node z' in the tree with the associated node group $L(z') = L$; make z' the current rightmost child of the node z using the pointer $p(z)$. If L is not a basic region, then add the item $(p(z'), L(z'))$ to the end of Q and let label$[z'] =$ label(L) if L consists of a cycle; otherwise, let label$[z']$ be the same as the labels of the nodes in L (all nodes in L have the same label, h or v).

(iii) Repeat steps (i)–(ii) using $\delta_2(w)$ in place of $\delta_2(w)$. (In this case L is empty except when w is the last item in $L(z)$; in processing $\delta_1(w)$, L is always nonempty.)

(c) If $L(z) = (w_{j_1}, w_{j_2}, w_{j_3}, w_{j_4})$ is a cycle, then process the lists $\delta_1(w_{j_i})$, and $\delta_2(w_{j_i}), 1 \leq i \leq 4$, in the order "<" as shown in the table below. The processing of $\delta_1(w_{j_i})$ or $\delta_2(w_{j_i})$ means that for each x in that list, indeg[x] and indeg[C], where C = cycle[x] $\neq$ nil, are reduced by one. Moreover, if the indegree of a node or a cycle becomes zero after it is reduced by one, then that node or the nodes of that cycle are added to one of the lists $L_k, 1 \leq k \leq 5$. This is indicated in the table by "$x \rightarrow L_k$". The lists L_k are initially assumed to be empty. The final result of this step are the five lists $L_k, 1 \leq k \leq 5$. (See Fig. 3(a) with the walls w_5, w_6, w_7, and w_8 regarded as $w_{j_1}, w_{j_2}, w_{j_3}$, and w_{j_4}, respectively.) Now, create five child nodes $z_k, 1 \leq k \leq 5$, of the node z in the tree in that order, with the associated node groups $L(z_k) = L_k$. For each $L(z_k)$ that is not a basic region add to Q the item $(p(z_k), L(z_k))$, where $p(z_k)$ is a pointer to the node z_k.

(i)	The case label[z] = s	(ii)	The case label[z] = S
$\delta_1(w_{j_1})$:	$x \rightarrow L_1$	$\delta_1(w_{j_1})$:	$x \rightarrow L_4$
$\delta_2(w_{j_1})$:	if $x < w_{j_2}$ then $x \rightarrow L_2$	$\delta_2(w_{j_1})$:	if $x < w_{j_2}$ then $x \rightarrow L_3$
	else $x \rightarrow L_3$		else $x \rightarrow L_5$
$\delta_1(w_{j_2})$:	$x \rightarrow L_2$	$\delta_1(w_{j_2})$:	if $x < w_{j_3}$ then $x \rightarrow L_3$
$\delta_2(w_{j_2})$:	if $x < w_{j_3}$ then $x \rightarrow L_3$		else $x \rightarrow L_2$
	else $x \rightarrow L_5$	$\delta_2(w_{j_2})$:	$x \rightarrow L_5$
$\delta_1(w_{j_3})$:	if $x < w_{j_4}$ then $x \rightarrow L_3$	$\delta_1(w_{j_3})$	if $x < w_{j_4}$ then $x \rightarrow L_1$
	else $x \rightarrow L_4$		else $x \rightarrow L_3$
$\delta_2(w_{j_3})$:	$x \rightarrow L_5$	$\delta_2(w_{j_3})$:	$x \rightarrow L_2$
$\delta_1(w_{j_4})$:	if $x < w_{j_1}$ then $x \rightarrow L_1$	$\delta_1(w_{j_4})$:	$x \rightarrow L_1$
	else $x \rightarrow L_3$	$\delta_2(w_{j_4})$:	if $x < w_{j_1}$ then $x \rightarrow L_4$
$\delta_2(w_{j_4})$:	$x \rightarrow L_4$		else $x \rightarrow L_3$

Figures 9(b) and (c) show the results of applying Algorithm 2 to the set of walls for the T_*-plan in Fig. 7(a). We show only the various node groups formed by Algorithm 2 corresponding to $w_p(R')$ and the cycles in the wall digraph, i.e., the nonterminal nodes of the tree $t(D)$.

6. Complexity analysis. Let n be the number of basic regions in the T_*-plan D. To argue that the time complexity of both Algorithms 1 and 2 are $O(n)$, it suffices to show that for all walls $w_j \in w(D)$ the lists $\delta_1(w_j)$ and $\delta_2(w_j)$ together with their linear ordering "<" can be determined in time $O(n)$. As for the other computations, we note that the total number of of arcs in $G_w(D)$ is $O(n)$ and the various processings of each arc (for the depth-first traversal, updates to indeg[x] for the vertices and cycles x and computation of label[w_j] for the walls) take at most a constant time c, for some c. Also, the number of cycles in $G_w(D)$ is at most $n/4$. This shows that the total time required by each algorithm is $O(n)$.

We determine the ordered lists $\delta_1(w_j)$ and $\delta_2(w_j)$ for all w_j in time $O(n)$ as follows. For each basic and external region x_i let first[x_i] denote the list of walls $w_j = \langle L_1, L_2 \rangle$, in some order, such that x_i is the first member of L_1 or L_2. We define the lists last[x_i] similarly. Each of the lists first[x_i] and last[x_i] has at most four items. The lists first[x_i] and last[x_i] can be constructed by a simple scan of L_1 and L_2 for each wall $w_j \in w(D)$ and by noting the first and last items in L_1 and L_2. The time required for this is proportional to the sum of lengths of all the lists L_1 and L_2 for $w_j \in w(D)$, which equals $4n + 8$ ($8 =$ the number of times the external regions N, S, W, and E appear in the various L_1 and L_2). Table 1 shows first$[x_i]$ and last$[x_i]$ for the walls in Fig. 1(b) when the walls are processed in the order $w_1, w_2, \cdots, w_{13}$.

TABLE 1
The lists first$[x]$ *and last*$[x]$ *corresponding to the walls in Fig.* 1(b).

x	first$[x]$	last$[x]$
N	(w_1)	(w_1)
S	(w_3)	(w_3)
W	(w_1, w_2, w_3)	(w_2)
E	(w_4)	(w_1, w_3, w_4)
r_1	(w_2, w_5, w_6)	(w_5)
r_2	(w_5, w_7)	(w_5, w_7)
r_3	(w_7)	(w_2, w_6, w_7)
r_4	(w_6, w_9, w_{12})	(w_{12})
r_5	(w_4, w_{12})	(w_9, w_{12})
r_6	(w_9, w_{10}, w_{11})	(w_{10})
r_7	(w_8, w_{10})	(w_8, w_{10})
r_8	(w_8, w_{13})	(w_9, w_{13})
r_9	(w_{13})	(w_8, w_{11}, w_{13})
r_{10}	(w_{11})	(w_4, w_6, w_{11})

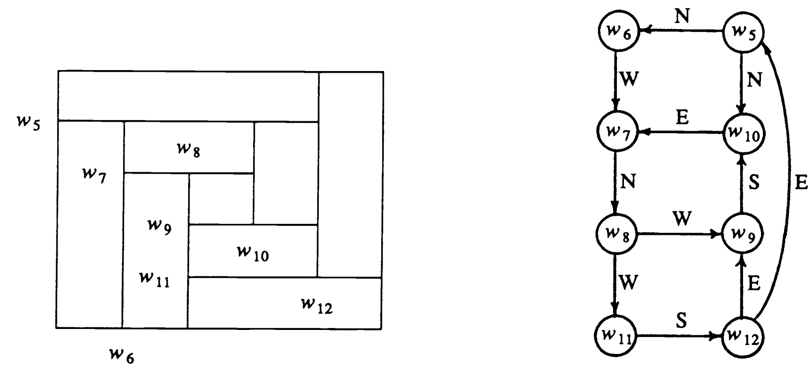
If $w_j = \langle L_1, L_2 \rangle$, then we construct the ordered list $\delta_1(w_j)$ by scanning L_1 in left-to-right order as follows. Let $L_1 = (x_1, x_2, x_3, \cdots, x_k)$ and $w_{j_i} =$ last$[x_i] \cap$ last$[x_{i+1}], 1 \leq i \leq k - 1$. Then $\delta_1(w_j) = (x_1, w_{j_1}, x_2, w_{j_2}, \cdots, w_{j_{k-1}}, x_k)$, except that we may have to remove one or both of x_1 and x_k, whichever equals an external region. The ordered list $\delta_2(w_j)$ is constructed similarly by using the list L_2 and the lists first$[x_i]$. For example, to compute $\delta_2(w_1)$ for the T_*-plan in Fig. 1(a), we start with the second list $L_2 = (W, r_1, r_4, r_5, E)$ for w_1 in Fig. 1(b) and this gives

$$(W, \text{first}[W] \cap \text{first}[r_1], r_1, \text{first}[r_1] \cap \text{first}[r_4], r_4, \text{first}[r_4] \cap \text{first}[r_5], r_5, \text{first}[r_5] \cap \text{first}[E], E)$$
$$= (W, w_2, r_1, w_6, r_4, w_{12}, r_5, w_4, E)$$

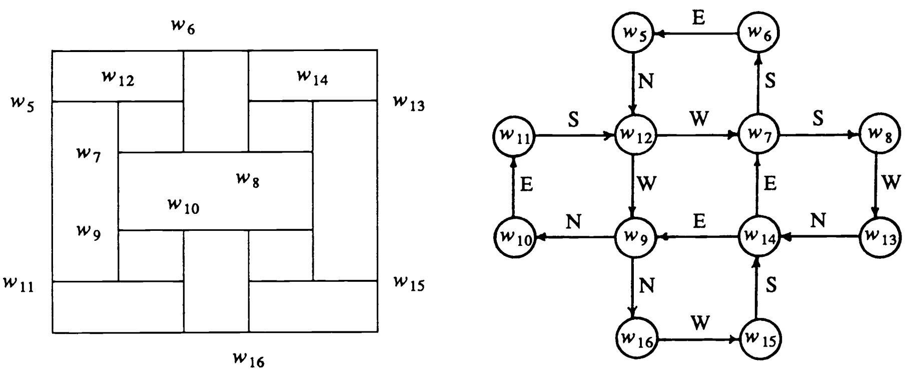
from Table 1. By removing the external regions W and E from this list, we get $\delta_2(w_1) = (w_2, r_1, w_6, r_4, w_{12}, r_5, w_4)$.

THEOREM 3. *The time complexity of each of Algorithms* 1 *and* 2 *is* $O(n)$, *where* $n =$ *number of basic regions in the* T_**-plan* D.

7. Generalization to arbitrary partitioning operators. The technique described in §§4 and 5 for the construction of the subregion tree $t(D)$ from $w(D)$ actually applies to more general classes of T-plans, namely, those generated by a finite family Π of partitioning operators that satisfy some natural conditions. If $\pi \in \Pi$ is an arbitrary partitioning operator, let D_π denote the dissection obtained by a single application of π and let $G_0(\pi) \subset G_I(D\pi)$ denote the subdigraph on the internal walls of D_π, excluding the region vertices. An operator $\pi, \pi \neq h$ or v, is called *primitive* if D_π contains no proper subregion other than a basic region. This is the same as saying that D_π cannot be obtained by combining two or more applications of other partitioning operators (in Π or otherwise). The spiral partitions are examples of primitive partitioning operators. Figures 10(a) and 10(b) show two other primitive partitioning operations $\Sigma =$ the double left spiral and $\Omega =$ the weave partition; also shown are the digraphs $G_0(\Sigma)$ and $G_0(\Omega)$.



(a)



(b)

FIG. 10. *Examples of primitive partition operations that cannot be generated by* $\{h, v, s, S\}$ *partition operations. The internal h-walls are labeled as* w_{2i-1}, *and the v-walls are labeled as* $w_{2i}, i \geq 1$: (a) *double-left-spiral partitioning operation* Σ *and its wall graph* $G_0(\Sigma)$ *on the internal walls*; (b) *weave-pattern partitioning operation* Ω *and its wall graph* $G_0(\Omega)$ *on the internal walls.*

A digraph G is called *strongly* connected if there is a directed path from each vertex to every other vertex in G. If G is not strongly connected, then a maximal strongly connected subdigraph of G is called a *strong component*. The strong components form a disjoint partition of the nodes of G. If we merge each strong component of G to a distinct vertex, then the result is an acyclic digraph. A strong component of G that corresponds to a source node in that acyclic digraph is called *source component*.

LEMMA 5. *If* $\pi \neq h$ *or* v *is a primitive partition operator, then the digraph* $G_0(\pi)$ *is strongly connected and has at least four vertices.*

Proof. Let $\{w_1, w_2, w_3, w_4\}$ be the external walls of the dissection D_π, and let C be a source component of $G_0(\pi)$. We claim that the walls in C form the internal walls of a dissection of the rectangular space R of D_π. The only way that this may be false is if there is a wall $w_j \in C$ that is dominated by a wall $w_i \in G_0(\pi) - C$. However, because C is a source component, there cannot be any w_i in $G_0(\pi) - C$ that dominates a wall $w_j \in C$. Thus the walls in C define a dissection, and hence must have $C = G_0(\pi)$ because otherwise one of the subregions of the dissection defined by C is partitioned by one or more of the remaining walls in $G_0(\pi) - C$, which contradicts that π is primitive.

Now if $C = \{w_j\}$ consists of a single wall, then $\pi = h$ or v according to whether w_j is an h-wall or a v-wall, and this is a contradiction. Finally, to show that C has at least four vertices, we simply note that $G_0(\pi)$ is a bipartite graph with each arc joining an h-wall and a v-wall. (The number of nodes in $G_0(\pi)$ may be odd or even. In Fig. 10(a) if we delete the h-wall w_9 and extend the v-wall w_{10} downward to meet with the h-wall w_{11}, then we get a variant Σ' of the double-left spiral partition, which is primitive, and there are seven nodes in $G_0(\Sigma')$.) □

For the techniques used in Algorithm 2 to work correctly for a general class of T-plans, which are obtained by the partition operators Π, we must assume that conditions (1) and (2) below hold. Condition (1) is a global property of the family Π as it relates different members of Π. Condition (2), on the other hand, constrains each individual member of Π separately. The family of operators $\Pi_0 = \{h, v, s, S\}$ for generating the T_*-plans clearly satisfies conditions (1) and (2); the same is true for the family $\Pi' = \{h, v, s, S, \Sigma, \Omega\}$.

(1) No two digraphs in the family $\{G_0(\pi) : \pi \in \Pi \text{ and } \pi \neq h, v\}$ are isomorphic as labeled digraphs (where each arc is labeled appropriately by one of the symbols in $\{N, S, W, E\}$).

(2) There is no nontrivial isomorphism of $G_0(\pi)$ to itself as a labeled digraph if $\pi \neq h$ or v.

Note that although both $G_0(\Sigma)$ and $G_0(\Omega)$ contain as a subdigraph an isomorphic copy of $G_0(s)$, this does not cause any problem in recognizing whether a particular occurrence of the subdigraph $G_0(s)$ in a wall digraph $G_w(D)$ is due to an s-partition or is due to one of the partition operators Σ and Ω. The former is the case if and only if that subdigraph $G_0(s)$ is a strong component of $G_w(D)$.

The main reason that Algorithm 2 can be extended to the general class of T-plans obtained by partition operations Π without increasing the time complexity $O(n)$ is that one can determine the strong components of the wall digraph $G_w(D)$ in time $O(n)$ [1]. As for the correctness of such an algorithm, we only need to note that Lemma 1 remains valid for the general T-plans in view of the strong-connectedness property of $G_0(\pi)$ shown in Lemma 5. This also means that Theorem 2 remains valid for the general T-plans. The role of the source cycles in Algorithm 2 is now replaced by the more general notion of source components. The details of the modification of step 6(c) in Algorithm 2 will depend on the specific partitioning operators in Π. For example, if $\pi =$ the double left spiral, then there will be nine lists L_k obtained in step 6(c).

We conclude the paper with two conjectures: (i) If π and π' are two primitive partitioning operators other than h and v, then $G_0(\pi)$ and $G_0(\pi')$ are not isomorphic as labeled digraphs. (ii) If $\pi \neq h$ or v is a primitive partition operator, then there is no nontrivial isomorphism from $G_0(\pi)$ to itself as labeled digraph. We assume here that two partitioning operators that have the same wall representation are not distinguished. For example, if the h-wall w_{13} in Fig. 10(b) is moved slightly up or down in relation to the h-wall w_5, then we do not consider the result to be a partitioning operator different from Ω. However, if we rotate the dissections in Fig. 10 by 90 degrees, then the corresponding partition operators are considered to be distinct from Σ and Ω. The h-walls now correspond to the original v-walls and vice versa. In particular, the wall representations of the new T-plans can be distinguished from those of Σ and Ω if we consider both the internal walls and the external walls (and only then). Also, the digraphs $G_0(\Sigma)$ and $G_0(\Omega)$ are distinguished from those of the new partition operators by the arc labels.

Acknowledgment. The author gratefully acknowledges the comments of the anonymous referees, which greatly helped in improving the presentation.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Algorithms*, Addison Wesley, Reading, MA, 1974.

[2] U. FLEMMING, *Wall representation of rectangular dissections and their use in automated space allocation*, Environ. Planning B, 5 (1978), pp. 215–232.

[3] ———, *On the representation and generation of loosely packed rectangles*, Environ. Planning B, 13 (1986), pp. 189–205.

[4] ———, *Wall representations of rectangular dissections: additional results*, Environ. Planning B, 7 (1980), pp. 247–251.

[5] P. GALLE, *An algorithm for exhaustive generation of building floor plans*, Comm. ACM, 24 (1981), pp. 813–824.

[6] ———, *Abstraction as a tool of automated floor-plan design*, Environ. Planning B, 13 (1986), pp. 21–46.

[7] S. KUNDU, *The equivalence of the subregion representation and the wall representation for a certain class of rectangular dissections*, Comm. ACM, 31 (1988), pp. 752–763.

[8] S. KUNDU AND R. SINGH, *Spatial reasoning in rectangular dissection*, in Proceedings of the Workshop on Spatial Reasoning and Multi-Sensor Fusion, IL, 1987, pp. 82–91.

[9] S. KUNDU, *A non-backtracking hierarchical placement algorithm for T_*-plans*, Tech. Report, Computer Science Department, Louisiana State University, Baton Rouge, LA, 1989.

[10] J. E. HASSETT, *Automated layout in ASHLER: an approach to the problem of "general cell" layout for* VLSI, in Proc. 19th Design Automation Conference, 1982, pp. 777–784.

[11] B. T. PREAS AND W. M. VANCLEEMPUT, *Placement algorithms for arbitrary shaped blocks*, in Proc. 16th Design Automation Conference, San Diego, CA, 1979, pp. 474–480.

[12] J. ROTH, R. HASHIMSHONY, AND A. WACHMAN, *Turning a graph into a rectangular floor plan*, Build. Environ., 17 (1982), pp. 163–173.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 102–113, February 1993

TALLY VERSIONS OF THE SAVITCH AND IMMERMAN–SZELEPCSÉNYI THEOREMS FOR SUBLOGARITHMIC SPACE*

VILIAM GEFFERT†

Abstract. It is shown that for each $s(n)$-space-bounded nondeterministic Turing machine recognizing a language $L \subseteq 1^*$ there exists an equivalent deterministic $O(s^2(n))$-space-bounded machine, and also a nondeterministic $O(s(n))$-space-bounded machine recognizing the complement of L, for any $s(n)$, independent of whether $s(n)$ is below $\log(n)$ or is space constructible. In other words, the Savitch [*J. Comput. System Sci.*, 4(1970), pp. 177–192] and Immerman–Szelepcsényi [*SIAM J. Comput.*, 17(1988), pp. 935–938], [*Acta Inform.*, 26(1988), pp. 279–284] theorems can be extended to any space bound $s(n)$ for languages over a single-letter alphabet.

Key words. space bounded computation, nondeterministic Turing machine, nondeterministic space, tally sets

AMS(MOS) subject classifications. 68Q15, 68Q75, 68Q05

1. Introduction.

Two of the most important results in space-bounded complexity theory, Savitch's simulation of nondeterministic space-bounded Turing machines by deterministic space-bounded machines [4] and the Immerman–Szelepcsényi proof that nondeterministic space is closed under complement [3], [6], were proved for only space bounds $s(n) \geq \log(n)$. If these facts are taken into consideration, the question of whether these two results can be extended to sublogarithmic-space bounds naturally arises. Although we do not give a complete answer, we show that for tally sets, i.e., for languages over a single-letter alphabet, the Savitch and Immerman–Szelepcsényi theorems are valid for any space bound $s(n)$, independent of whether $s(n)$ is below $\log(n)$ or is space constructible.

In fact, these two theorems are also valid for languages over the binary-tape alphabet with low information content, e.g., if there exists a constant d such that each word in L of length n contains at most $d^{s(n)}$ zeros.

The situation is much more complicated for space-complexity classes below $\log(n)$ than for those above, because we do not have enough space to count the number of reachable configurations or the length of a computation path and hence we cannot use the deterministic simulation [4] or the inductive counting method [3], [6] directly. Moreover, each machine using less than $\log(n)$ space is physically incapable of verifying that it has returned to the same input-tape position after moving its input-tape head too far, because it does not have enough space to remember a position of the input-tape head.

We shall begin in §2 by showing that the result of any computation on tally input is completely determined by the machine's behavior near the input-tape end markers. For the detailed proofs of the lemmas presented in §2 the reader is referred to [2]. Section 3 deals with the closure under complement. The construction is based on a modified version of the standard inductive counting algorithm [3], [6], which calls as its subprogram another version of the inductive counting algorithm. In §4 we shall show that the same technique can be applied to Savitch's deterministic simulation [4] as well, that is, one version of the deterministic simulation algorithm calls another one. Section 5 discusses some extensions and the fundamental difference between the methods presented here and the standard Savitch and Immerman–Szelepcsényi algorithms for $s(n) \geq \log(n)$.

*Received by the editors May 24, 1991; accepted for publication (in revised form) January 24, 1992. This work was supported by research grants SPZV I-1-5/08 and MSMS SR 01/46.

†Department of Computer Science, University of P. J. Šafárik, Jesenná 5, 04154 Košice, Czechoslovakia.

2. Nondeterministic computations of tally inputs. We shall consider the standard Turing machine model having a two-way read-only input tape, a finite control, and a separate semi-infinite two-way read-write work tape. This model was introduced in [5] for studying computations requiring less than linear space. A nondeterministic Turing machine is $s(n)$-*space bounded* if all computation paths on all inputs of length n use at most $s(n)$ tape squares on the work tape.

DEFINITION 1. (a) A *memory state* of a Turing machine is an ordered triple $q = \langle r, u, j\rangle$, where r is a state of the machine's finite control, u is a string of work-tape symbols written on the work tape (not including the left end marker and blank symbols), and j is a position of the work tape head.

(b) A *configuration* is an ordered pair $k = \langle q, i\rangle$, where q is a memory state and i is a position of the input-tape head.

It is not too difficult to verify that for each $s(n)$-space-bounded nondeterministic Turing machine there exists a constant $c \geq 6$ such that the number of different memory states for inputs of length n is at most $c^{s(n)+1}$. Similarly, the number of configurations can be bounded by $nc^{s(n)+1}$ for each $n \geq 1$. Thus we have a constant c such that for each $n \geq 1$

$$\begin{aligned} &6 \leq c, \\ (1)\qquad &\text{number of memory states} \leq c^{s(n)+1}, \\ &\text{number of configurations} \leq nc^{s(n)+1}. \end{aligned}$$

Note that we need only $\Theta(s(n))$ space to remember a memory state, but we need $\Theta(s(n)+\log(n))$ space to remember a configuration. The same amounts of space are required to count the number of memory states and the number of configurations, respectively. That is why the Savitch and Immerman–Szelepcsényi algorithms can be applied only to space bounds satisfying $\log(n) \in O(s(n))$.

In particular, both the Savitch and Immerman–Szelepscényi theorems are valid if $n \leq \left(c^{s(n)+1}\right)^6$, since then the position of the input-tape head can be stored in $O(s(n))$ space, as can be a particular configuration or number of reachable configurations. Therefore, we shall now concentrate on nondeterministic Turing machines using very little space, i.e., with

$$(2)\qquad (c^{s(n)+1})^6 < n.$$

The computations taking place very close to the input-tape end markers play a dominant role for such machines when they are computing on tally inputs. In fact, the result of any computation [2] on a tally input is completely determined by computation paths not moving the input head farther than $(c^{s(n)+1})^5$ positions away from the tape end markers. To make these ideas more precise, we shall need some lemmas. For the detailed proofs of Lemmas 1, 3, and 4 the reader is referred to [2]. All proofs are based on the assumption that $(c^{s(n)+1})^6 < n$. In what follows we shall therefore assume that the input word is 1^n and that the space bound $s(n)$ satisfies $\left(c^{s(n)+1}\right)^6 < n$. Define

$$M = c^{s(n)+1}$$

to be an upper bound on the number of reachable memory states for input 1^n.

LEMMA 1 [2, Lem. 3]. *If there exists a computation path from the configuration $k_1 = \langle q_1, i\rangle$ to the configuration $k_2 = \langle q_2, i\rangle$, i.e., beginning and ending at the same position on* *the input tape, such that the input head never visits the right end marker, then the shortest computation path from $\langle q_1, i\rangle$ to $\langle q_2, i\rangle$ never moves the input head farther than M^2 positions to the right of i.*

By symmetry a similar statement holds for computation paths from $\langle q_1, i\rangle$ to $\langle q_2, i\rangle$ that never visit the left end marker.

The next lemma states that the optimal computation paths beginning and ending very close to the end markers never move the input head too far, nor do they consume too much time.

LEMMA 2. *If there exists a computation path from the configuration $k_1 = \langle q_1, i_1\rangle$ to $k_2 = \langle q_2, i_2\rangle$, where $i_1 \leq M^4 + M$ and $i_2 \leq M^4 + M$, such that the input head never visits the right end marker, then the shortest computation path from k_1 to k_2* (a) *never moves the input head more than M^5 positions away from the left end marker and* (b) *never executes more than M^6 steps. The same holds for computation paths beginning and ending closer than $M^4 + M$ positions from the right end marker.*

Proof. (a) Suppose that the rightmost configuration in the shortest computation path from k_1 to k_2 is $k = \langle q, i\rangle$, where $i > M^5$. Because $M = c^{s(n)+1} \geq c \geq 6$, by (1), we have that $M^5 - (M^4 + M) > M^2$. Since both i_1 and i_2 lie to the left of the position $M^4 + M$, we can find two configurations $k_1' = \langle q_1', M^4 + M\rangle$ and $k_2' = \langle q_2', M^4 + M\rangle$ such that the shortest computation path from k_1' to k_2' moves the input head to the position i, i.e., more than M^2 positions to the right (see Fig. 1), but this is a contradiction by Lemma 1.

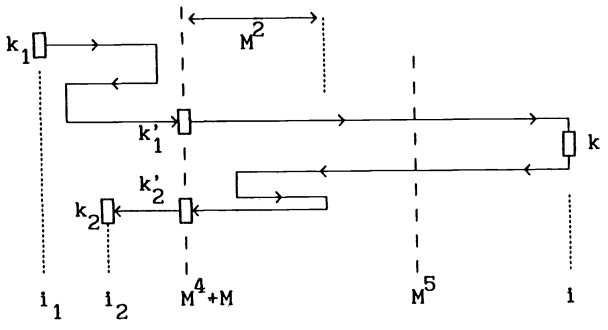


FIG. 1

(b) Furthermore, since there are at most M^6 different configurations with the input-head positions bounded by M^5, the shortest computation path from k_1 to k_2 executes at most M^6 steps. If not, some configuration would have been repeated. □

The following lemma shows that computation paths on tally inputs are position independent provided that they begin and end at least $M^2 + 1$ positions away from either end marker.

LEMMA 3 [2, Lem. 4]. *If machine A can get from configuration $\langle q_1, i\rangle$ to $\langle q_2, i + \ell\rangle$ by a computation path visiting neither of the end markers, then A can get from $\langle q_1, j\rangle$ to $\langle q_2, j + \ell\rangle$ for each j satisfying*

$$M^2 + 1 \leq j \leq n - (M^2 + 1),$$
$$M^2 + 1 \leq j + \ell \leq n - (M^2 + 1).$$

The proof uses Lemma 1 and the fact that the input head scans the same symbols (ones) everywhere along the tape with a tally input.

The following lemma asserts that the nondeterministic machine A, having a computation path that traverses the whole input from left to right (or vice versa), has one path that repeats a single loop on most of the input, beginning and ending in the same configurations at the endmarkers. *A loop of length ℓ* is a computation path starting in configuration $\langle q, i\rangle$ and ending in $\langle q, i+\ell\rangle$ for some memory state q and tape position i. Moreover, neither of the end markers is visited by the input head during this computation.

LEMMA 4 [2, Thm. 1]. *Each computation path beginning in configuration $\langle q_1, 0\rangle$ and ending in $\langle q_2, n+1\rangle$ such that the end markers are visited only in $\langle q_1, 0\rangle$ and $\langle q_2, n+1\rangle$ can be replaced by an equivalent computation path visiting the end markers only in $\langle q_1, 0\rangle$ and $\langle q_2, n+1\rangle$ such that machine A,*

(a) *having traversed s_1 input-tape positions,*

(b) *gets into a loop of length ℓ that is iterated r times and*

(c) *then traverses the rest of the input tape of length s_2*

for some s_1, ℓ, r, s_2 such that

$$\begin{aligned} &M^2+1 \le s_1 \le M^4, \\ (3)\qquad &M^2+1 \le s_2 \le M^4, \\ &1 \le \ell \le M. \end{aligned}$$

The same holds for traversals from right to left (*see also Fig.* 2).

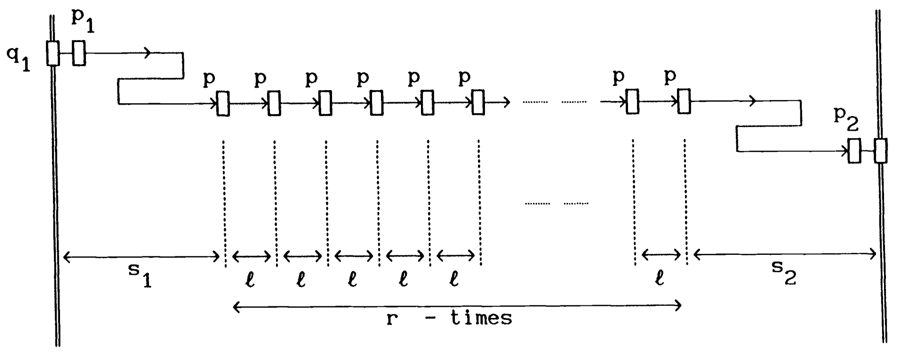


FIG. 2

A configuration is *extending* if it has used space h on the work tape and is going to use space $h+1$ on the next computation step (by rewriting the leftmost blank on the work tape by a nonblank symbol).

LEMMA 5. *For each $s(n)$-space-bounded nondeterministic Turing machine A there exists an equivalent machine A' such that for each $h = 0, \cdots, s(n)$ there exists a configuration having used exactly h space on the work tape with the input head positioned at the left end marker* (*reachable from the initial configuration*). *Moreover, A' accepts with the input head at the left end marker.*

Proof. We can replace the original machine A by a new machine A' that simulates A but that, each time A gets into an extending configuration, nondeterministically decides whether to carry on the simulation or to move the input head to the left end marker, extend the work tape space, and then halt and reject the input. Machine A' has more computation paths than does the original machine A, but all new computation paths are terminated in nonaccepting configurations, and hence both A and A' recognize the same language. A similar idea can be used for accepting configurations as well. $\square$

3. Inductive counting. The reader is assumed to be familiar with the Immerman–Szelepcsényi proofs that nondeterministic space is closed under complement for each space bound $s(n) \geq \log(n)$ [3], [6]. The idea is that if for each $s(n)$-space-bounded nondeterministic machine A we could compute d, the exact number of distinct configurations reachable from the initial configuration, then we could recognize $\overline{L(A)}$ in $s(n)$ space.

Simulate, d times, a computation of A from the very beginning along a (nondeterministically chosen) computation path, and check whether these d simulations are terminated in d different configurations in a lexicographically increasing order. This can be done in $O(s(n)+\log(n))$ space because all we have to remember is d, a variable i to count from 1 to d, a current configuration along some computation path, and the configuration resulting from the previous simulation in order to check that these d configurations are generated in increasing order so that none of them is generated twice. For the right sequence of nondeterministic guesses we shall reach d distinct configurations in increasing order and accept the input if and only if none of these d configurations is an accepting configuration.

The proof that the number of reachable configurations can be computed in space $O(s(n) + \log(n))$ is shown by induction on the number of steps. We shall present this result in a slightly modified form. The algorithm computing the number of reachable configurations is used to test whether a configuration k_2 can be reached from k_1 by a computation path executing at most $t\ell$ steps for any given k_1, k_2 and time limit $t\ell$.

To do this we need to compute d, defined as the number of configurations reachable from a configuration k_1 by computation paths executing *exactly* t steps, for any given k_1 and t.

Clearly, $d = 1$ for $t = 0$. Having computed d, we can compute d', the number of configurations reachable from k_1, by computations executing exactly $t + 1$ steps. To do this, for each target configuration k' generate the d distinct configurations reachable from k_1 in t steps in lexicographically increasing order. Verify whether, among these d configurations, there is a configuration k such that k' is reachable from k in one step, and increment d' in this case.

Now, if we want to check whether a configuration k_2 can be reached from k_1 by a computation path executing at most $t\ell$ steps, we must successively check for each $t = 0, \ldots, t\ell$ if k_2 is equal to any of the configurations reachable from k_1 in exactly t steps.

Because this algorithm will be used in several modifications on different levels, we shall present it using a more formal notation, in the form of a Boolean function test$(k_1, k_2, t\ell)$ such that (a) if a configuration k_2 is reachable from k_1 by a computation path executing at most $t\ell$ steps, then the test will return TRUE in at least one computation path and no computation path will return FALSE; (b) if k_2 is not reachable from k_1 in $t\ell$ steps, then the test will return FALSE in at least one computation and no computation will return TRUE. Note that the test may also halt the computation and reject the input returning no value if it finds that a wrong nondeterministic choice has been made.

```
1   function test($k_1, k_2, t\ell$)
2       $d' := 1$
3       for $t := 0$ to $t\ell$ do
4           $d := d'; d' := 0$
5           for each $k'$ do
6               $k_{\text{prev}} := k_{\text{zero}}$; increment := 0
7               for $i := 1$ to $d$ do
8                   $k :=$ simulation_result($k_1, t$)
9                   if $k = k_2$ then return TRUE
10                  if $k \le k_{\text{prev}}$ then reject
11                  if step($k, k'$) then increment := 1
12                  $k_{\text{prev}} := k$
13              end
14              $d' := d'$ + increment
15      end   end
16      return FALSE
17  end
```

In the algorithm above, "**for each** k' **do** ... **end**" denotes the loop that is executed for each configuration k' not using more than s space, where s is a global variable stored on a separate work tape track. k_{zero} is a constant representing a dummy configuration that lexicographically precedes any other configuration. k_{zero} is not reachable from any other configuration. simulation_result(k_1, t) is a nondeterministic function returning a configuration that is reachable from k_1 by a computation path executing exactly t steps. It simply simulates machine A from k_1 and counts the simulated steps up to t. This function may halt the entire computation and reject the input if it finds that the computation path ends, not having executed t steps. The reject procedure halts the computation and rejects the input, and Boolean function step(k, k') returns the value TRUE if and only if k' is reachable from k in one computation step.

Clearly, if we want to determine whether k_2 is reachable from k_1, it is sufficient to use test($k_1, k_2, nc^{s(n)+1}$) because the shortest computation path from k_1 to k_2 cannot enter the same configuration twice and there are at most $nc^{s(n)+1} = nM$ distinct configurations. In particular, we can test whether any of the accepting configurations is reachable from the initial configuration. It can easily be seen that all variables then require at most $O(s(n) + \log(n))$ space.

We shall now show that the space used can be reduced to $O(s(n))$ if we consider computations on the input 1^n. By the lemmas presented in §2 the actions of the nondeterministic machine on tally inputs can be understood as a combination of moves that are close to one of the end markers of the input tape (at a distance of at most M^4) together with long marches between the end markers. Moreover, these long marches can be replaces by fixed loops of lengths less than or equal to M. Space $O(s(n))$ suffices to describe these situations.

Using inductive counting, by induction on the number of times the input head hits an end marker (instead of single computation steps), we shall construct a procedure that checks whether there is a computation path connecting two configurations that have the input head positioned at the end markers. This procedure calls as its subprogram another version of the inductive counting algorithm that is used to analyze computations that are close to one of the end markers.

First, we shall show that the space $O(s(n))$ is sufficient for analysis of computations taking place entirely within positions 1 and M^5. The following modifications are required in the procedure test:

Line 5. The loop is iterated for each configuration k' not using more than s space, such that the input head is between positions 1 and M^5, where s and M are global variables stored on separate work tape tracks and M is equal to c^{s+1}.

Line 8. The function simulation_result(k_1, t) is replaced by a nondeterministic function L_simulation_result(k_1, t), which returns a configuration reachable from k_1 by a computation path executing exactly t steps, neither moving the input head to the right of M^5 nor visiting the left end marker (position 0). This procedure simulates A from k_1 and counts the simulated steps. It rejects the input (returning no configuration) if machine A moves the input head to the right of position M^5 or scans the left end marker or if the chosen computation path terminates too early, not having executed t steps.

The above modifications give function L_test($k_1, k_2, t\ell$), which checks whether configuration k_2 is reachable from k_1 by a computation executing at most $t\ell$ steps taking place entirely within positions 1 and M^5 or rejects the input because of a wrong nondeterministic decision. The procedure simply uses inductive counting to compute the number of configurations reachable by computation paths taking place between positions 1 and M^5.

Note that by Lemma 2 we can check whether configuration k_2 is reachable from k_1 by a computation path visiting neither of the end markers by the use of L_test(k_1, k_2, M^6) for each $k_1 = \langle q_1, i_1 \rangle$ and $k_2 = \langle q_2, i_2 \rangle$ such that $1 \leq i_1 \leq M^4 + M, 1 \leq i_2 \leq M^4 + M$, because the shortest computation path from k_1 to k_2 never moves the input head to the right of position M^5 nor does it execute more than M^6 steps, since both the initial and final positions are to the left of $M^4 + M$. It is not difficult to verify that L_test then uses at most $O(s(n))$ space.

Now we can easily construct function L_L_path(q_1, q_2), which checks whether there exists a computation path from memory state q_1 to q_2 beginning and ending at the left end marker such that the end markers are visited only in q_1 and q_2:

```
function L_L_path($q_1, q_2$)
    if step($\langle q_1, 0\rangle$ , $\langle q_2, 0\rangle$) then return TRUE
    for each $p_1, p_2$ do
        if step($\langle q_1, 0\rangle$ , $\langle p_1, 1\rangle$) and step($\langle p_2, 1\rangle$ , $\langle q_2, 0\rangle$) and
          L_test($\langle p_1, 1\rangle$ , $\langle p_2, 1\rangle$ , $M^6$)
        then return TRUE
    end
    return FALSE
end
```

The construct "**for each** p_1, p_2 **do** ... **end**" denotes two nested loops that are iterated for all memory states not using more than s space.

DEFINITION 2. An *R-configuration* is an ordered pair $k = \langle q, i \rangle$, where q is a memory state and i is a distance between the input-tape head and the right end marker.

Clearly, for each configuration $\langle q, i \rangle$ there exists a corresponding *R-configuration* $\langle q, n + 1 - i \rangle$, and vice versa. We introduce this notion only to have a space-efficient coding for configurations with the input head positioned close to the right end marker.

We can now design procedures R_step, R_simulation_result, R_test, and R_R_path that, by symmetry, mirror step, L_simulation_result, L_test, and L_L_path, respectively. That is, R_step(k, k') returns TRUE if and only if R-configuration k' is reachable from R-configuration k in one computation step, and R_test($k_1, k_2, t\ell$) checks whether R-configuration k_2 is reachable from k_1 by a computation path executing at most $t\ell$ steps, neither moving the input head more than M^5 positions away from the right end marker nor visiting the right end marker. For R-configurations with the input head less than $M^4 + M$ positions to the left of the right end marker, R_test(k_1, k_2, M^6) checks whether k_2 is reachable from k_1 by a computation path visiting neither of the end markers. R_R_path(q_1, q_2) checks whether there exists a path from memory state q_1 to q_2 beginning and ending at the right end marker, visiting the end markers only in q_1 and q_2.

We shall now construct a function L_R_path(q_1, q_2), which checks whether there exists a computation path from memory state q_1 to q_2 traversing the entire input from left to right such that the end markers are visited only in q_1 and q_2. By Lemma 4 it is sufficient to check whether there exist s_1, ℓ, r, s_2 satisfying (3) and a memory state p such that

(a) $\langle p, s_1\rangle$ is reachable from $\langle q_1, 0\rangle$,

(b) $\langle p, s_1 + (i+1)\ell\rangle$ is reachable from $\langle p, s_1 + i\ell\rangle$, for each $i = 0, \cdots, r-1$, and

(c) $\langle q_2, n+1\rangle$ is reachable from $\langle p, n+1-s_2\rangle = \langle p, s_1 + r\ell\rangle$ (see Fig. 2). Because the end markers are visited only in q_1 and q_2, condition (a) can be verified by using L_test($\langle p_1, 1\rangle, \langle p, s_1\rangle, M^6$) for each memory state p_1 such that $\langle p_1, 1\rangle$ is reachable from $\langle q_1, 0\rangle$ in one step. The same holds for (c), but we shall use R-configurations instead to reduce the space used.

From Lemma 3 it follows that, having verified that $\langle p, s_1 + \ell\rangle$ is reachable from $\langle p, s_1\rangle$, we have verified that $\langle p, s_1 + (i+1)\ell\rangle$ is reachable from $\langle p, s_1 + i\ell\rangle$ for each $i \geq 0$ provided that both $s_1 + i\ell$ and $s_1 + (i+1)\ell$ are at least $M^2 + 1$ positions away from either end marker. This is satisfied for each $i = 0, \cdots, r-1$, since $s_1 \geq M^2 + 1$ and $s_2 \geq M^2 + 1$ by (3) (see Lemma 4). It only remains to check whether $s_1 + r\ell + s_2 = n+1$. Note that we must store s_1, s_2 and ℓ, but not r, on the work tape. We can verify whether $(n+1-s_1-s_2) \bmod \ell = 0$ by moving the input head $s_1 + s_2$ positions to the right from the left end marker and checking whether the rest of the tape can be divided into segments of equal length ℓ. We are now ready to present the algorithm:

```
function L_R_path($q_1, q_2$)
    for each $p, p_1, p_2$ do
        for $s_1, s_2 := M^2 + 1$ to $M^4$ do
            for $\ell := 1$ to $M$ do
                if step($\langle q_1, 0\rangle, \langle p_1, 1\rangle$) and
                   L_test($\langle p_1, 1\rangle, \langle p, s_1\rangle, M^6$) and
                   L_test($\langle p, s_1\rangle, \langle p, s_1 + \ell\rangle, M^6$) and
                   R_test($\langle p, s_2\rangle, \langle p_2, 1\rangle, M^6$) and
                   R_step($\langle p_2, 1\rangle, \langle q_2, 0\rangle$) and
                   $(n + 1 - s_1 - s_2) \bmod \ell = 0$
                then return TRUE
    end end end
    return FALSE
end
```

In the algorithm "for each p, p_1, p_2 do $\cdots$ end" denotes three nested loops iterated for all memory states not using more than s space.

A similar algorithm can be used for the function R_L_path(q_1, q_2), which checks traversals from right to left.

DEFINITION 3. An *M-configuration* is an ordered pair $k = \langle q, m\rangle$, where q is a memory state and $m \in \{L, R\}$, where L and R denote the left and right end markers, respectively.

M-configurations will be used to identify configurations with the input head scanning the end markers. Combining L_L_path, R_R_path, L_R_path, and R_L_path, we obtain

```
function M_step($\langle q_1, m_1\rangle, \langle q_2, m_2\rangle$)
    return $m_1$_$m_2$_path($q_1, q_2$)
end
```

which checks whether there exists a computation path from M-configuration $\langle q_1, m_1\rangle$ to M-configuration $\langle q_2, m_2\rangle$ by visiting the end markers only in $\langle q_1, m_1\rangle$ and $\langle q_2, m_2\rangle$.

Now, using inductive counting again, we can construct a function M_test($k_1, k_2, t\ell$), which checks whether there exists a computation path from M-configuration k_1 to M-configuration k_2 such that the input head visits the end markers at most $t\ell$ times.

The algorithm uses induction not on the number of single computation steps but, rather, on the number of times the input head visits the end markers. Having computed d, the number of M-configurations reachable from k_1 by computation paths visiting the end markers exactly t times, we can compute d', the number of M-configurations reachable from k_1 by computations visiting the end markers exactly $t + 1$ times. For each target M-configuration k' generate the d distinct M-configurations reachable from k_1 by exactly t visits to the end markers and verify whether there is an M-configuration k such that k' is reachable from k by a computation path visiting the end markers only in k and k' by using the function M_step. Increment d' in this case.

We shall need some modifications in the procedure test:

Line 5. The loop is not iterated for each configuration; rather it is iterated for each M-configuration k' not using more that s space on the work tape.

Line 11. The function step(k, k') is replaced by M_step(k, k').

Line 8. The function simulation_result(k_1, t) returning a configuration reachable from k_1 by a computation executing t steps is replaced by nondeterministic function M_simulation_result(k_1, t), which returns an M-configuration reachable from M-configuration k_1 by a computation path visiting the end markers exactly t times not including k_1 itself. This procedure simulates the original machine A from k_1 and counts the number of times the head visits the end markers instead of counting single computation steps. The input is rejected and the entire computation is halted if the chosen path terminates without having visited the end markers t times. The only M-configuration returned for $t = 0$ is k_1 itself.

Clearly, M_test($k_1, k_2, 2M$) checks whether M-configuration k_2 is reachable from M-configuration k_1, since there are at most $2M$ distinct M-configurations and therefore the shortest computation path from k_1 to k_2 cannot visit the end markers more than $2M$ times. By using this it can easily be shown that M_test uses at most $O(s(n))$ space.

By Lemma 5 there is no loss of generality is assuming that the original machine A accepts with the input head at the left end marker. Therefore, it is sufficient to check whether any accepting M-configuration with the head at the left end marker is reachable from the initial M-configuration $\langle q_I, L\rangle$. However, two problems arise here. First, we do not know a priori how much space machine A is going to use. This can be overcome by testing for $s = 1, 2, \cdots$ if A is going to use at least that much space. By Lemma 5 we may assume without loss of generality that if A uses $s(n)$ space on the work tape, then for each $h = 0, \cdots, s(n)$ there exists a configuration reachable from the initial configuration that uses exactly h space *with the input head at the left end marker.* Hence it is sufficient to iterate over all M-configurations having used exactly space s with the input head at the left end marker and to check whether any of these is reachable from $\langle q_I, L\rangle$. We shall interrupt the work-tape space extension when we reach s such that no M-configuration using exactly space s at the left end marker is reachable from the initial M-configuration. Second, our strategy is based on Lemmas 1, 3, and 4 [2], i.e., on the assumption that $M^6 = (c^{s(n)+1})^6 < n$. Fortunately, if $n \leq (c^{s(n)+1})^6 = M^6$, then the STANDARD inductive counting algorithm [3], [6] uses at most $O(s(n))$ space, since then $\log(n) \in O(s(n))$. Therefore, each time the work-tape space s is extended we should check whether $M^6 = (c^{s+1})^6 < n$ and decide whether to proceed further or to switch to the standard inductive counting procedure.

Combining the ideas above, we get the following main program:

```
procedure main
    $s := 0$; $M := c$; extend := 1
    while extend = 1 and $M^6 < n$ do
        extend := 0
        for each $q$ do
            if worktape_space_used ($q$) = $s$ and
               M_test($\langle q_I, L\rangle, \langle q, L\rangle, 2M$)
            then extend := 1
        end
        $s := s + 1$; $M := c^{s+1}$
    end
    if $M^6 \geq n$ then STANDARD else begin
        for each $q$ do
            if accepting($q$) and
               M_test($\langle q_I, L\rangle, \langle q, L\rangle, 2M$)
            then reject
        end
        accept
end end
```

By the argument above we have the following theorem:

THEOREM 1. *For each $s(n)$-space-bounded nondeterministic Turing machine recognizing a language $L \subseteq 1^*$ there exists a nondeterministic $O(s(n))$-space-bounded Turing machine recognizing the complement of L for each $s(n)$, independent of whether it is below* $\log(n)$ *or is space constructible.*

4. Deterministic simulation. The same approach can be applied to Savitch's theorem [4]. Consider the deterministic version of the function test:

```
1   function test($k_1, k_2, t\ell$)
2       if $k_1 = k_2$ then return TRUE
3       if $t\ell = 1$ then return step($k_1, k_2$)
4       for each $k$ do
5           if test($k_1, k, \lfloor t\ell/2 \rfloor$) and test($k, k_2, \lceil t\ell/2 \rceil$)
6           then return TRUE
7       end
8       return FALSE
9   end
```

We have used the same function step but no nondeterministic function like simulation_result. The procedure uses $O(s(n) + \log(n) + \log(t\ell))$ *local* space, but it calls itself recursively. The depth of recursion is bounded by $\lceil \log(t\ell) \rceil$.

The basic idea of the proof is the same as it was for the nondeterministic simulation. The deterministic version of L_test will use $O(s(n))$ local space to store configurations that have input-head positions bounded by M^5. The same amount of local space is sufficient for the time limit $t\ell$ bounded by M^6. The depth of recursion for L_test is then bounded by $\lceil \log(M^6) \rceil \in O(s(n))$, which gives the total space $O(s^2(n))$. The same amount of space is sufficient for R_test and also for the deterministic recursive variant of M_test, calling M_step instead of step (cf. the deterministic function test, line 3) with $t\ell$ bounded by $2M$. Savitch's STANDARD procedure that is called in the main program also uses at most $O(s^2(n))$ space if $n \leq M^6$.

We can now establish the following theorem:

THEOREM 2. *For each $s(n)$-space-bounded nondeterministic Turing machine recognizing a language $L \subseteq 1^*$ there exists an equivalent deterministic Turing machine that is space bounded by $O(s^2(n))$ for each $s(n)$, independent of whether it is below* $\log(n)$ *or is space constructible.*

5. Conclusion. The above idea can be used not only for tally inputs but also for binary inputs with a low information content, e.g., for strings not containing too many zeros:

DEFINITION 4. A language $L \subseteq \{0,1\}^*$ is *$f(n)$-zero-bounded* if each $w \in L$ of length n contains at most $f(n)$ zeros.

THEOREM 3. *For each $s(n)$-space-bounded nondeterministic Turing machine recognizing an $f(n)$-zero-bounded language L, such that the value of $f(n)$ can be computed and stored in $O(s(n))$ space, there exists a nondeterministic $O(s(n))$-space-bounded Turing machine recognizing the complement of L and also an equivalent $O(s^2(n))$-space-bounded Turing machine that is deterministic.*

Note that we have to compute $f(n)$ by using $O(s(n))$ space only, but $s(n)$ itself need not be fully space constructible.

Proof. We state only the basic ideas for the inductive counting, since the proof is very similar to the arguments presented above. (A similar argument can be used for the deterministic simulation.)

First, compute $f(n)$, and store this value on a separate work-tape track. Accept the input if the number of zeros on the input tape exceeds $f(n)$.

Otherwise, we must simulate the original machine A. Because $f(n)$ can be stored in $O(s(n))$ space, there is a constant d such that $f(n) \leq d^{s(n)}$. Since there are at most $d^{s(n)}$ zeros on the input tape, there can be at most $c^{s(n)+1}d^{s(n)}$ so-called Z-configurations, i.e., with the head scanning a symbol zero on the input tape.

No optimal computation path enters the same Z-configuration twice, and therefore we shall consider only computation paths scanning zeros on the input tape at most $c^{s(n)+1}d^{s(n)}$ times. (In fact, the value of $c^{s+1}f(n)$ will be computed and stored on a separate work-tape track. Because s is not known at the beginning of the computation, this value is recomputed each time s is extended.)

Moreover, each Z-configuration can be stored on the work tape by using only $O(s(n))$ space because the input tape position can be coded by the number of zeros lying to the left of the current position.

Using inductive counting, we can now generate and count all Z-configurations that are reachable from the initial configuration by induction on the number of times the input head scans zero.

We must call a nondeterministic Z-configuration generator, i.e., a function Z_simulation_result(k, t) returning a Z-configuration reachable from Z-configuration k by a computation path scanning zeros on the input exactly t times.

Second, we shall need a single Z-step verifier, i.e., a function Z_step(k_1, k_2) to check whether Z-configurations k_1 and k_2 can be connected by a computation path scanning zeros only in k_1 and k_2. Such a computation path can be found only if k_1 and k_2 scan the same zero on the input tape or if they scan two adjacent zeros, i.e., two zeros separated by 1^k, for some $k \geq 0$. Return FALSE if this is not satisfied. Now consider the two adjacent zeros. (The argument for the first case is very similar—by Lemma 1—and therefore is omitted.) Compare k, the distance between these two zeros, with M^6. If $k \leq M^6$, then we shall call the standard inductive counting procedure and use the tape segment between the two zeros as its effective input, measuring off the input-head positions from the left zero. Otherwise, we shall use the strategy based on Lemma 4 (see §3, procedure L_R_path).

The basic difference from the tally input recognition is that M^6 is compared not at the beginning of the main program but each time the procedure Z_step is called to analyze computation paths between any two adjacent zeros. □

For example, the Savitch and Immerman–Szelepcsényi theorems hold for each $d^{s(n)}$-zero-bounded language, where d is a constant, if $s(n)$ is fully space constructible.

These two theorems hold for each $L \subseteq a^*b^*$ for each $s(n)$, independent of whether $s(n)$ is above $\log(n)$ or is space constructible, since there is a straightforward correspondence between L and the 1-zero-bounded language $L' = \{1^m 0 1^n; a^m b^n \in L\}$.

However, there is a very important difference between inductive counting in sublogarithmic space and the standard Immerman–Szelepcsényi result. In fact, the Immerman–Szelepcsényi algorithm for $s(n) \geq \log(n)$ can be used to generate all configurations reachable from *any* given configuration k, not only from the initial configuration. It is this fact that is needed to prove that the alternating hierarchy of $s(n)$-space-bounded machines collapses to $\Sigma_1-\text{SPACE}(s(n))$. $\Sigma_k-\text{SPACE}(s(n))$ and $\Pi_k-\text{SPACE}(s(n))$ denote the classes of languages recognizable by alternating $s(n)$-space-bounded machines making fewer than k alternations between universal and existential states with the initial state existential and universal, respectively. This is not possible in sublogarithmic space because we do not have enough space to remember the input-head position. Once we have moved the input head too far, the tape position is lost. We cannot restart, over and over again, computation paths beginning in the same configuration. On tally inputs this is possible only for a restricted set of configurations with the input-head positions close to the end markers. Fortunately, the initial configuration belongs to this set, and that is why we were able to check whether there is an accepting reachable configuration.

However, we still have that $\Pi_2 - \text{SPACE}(s(n)) - \Sigma_1 - \text{SPACE}(s(n)) \neq \emptyset$ for each $s(n)$ between $\log\log(n)$ and $\log(n)$ [7], [1]. An example [7] of a language over a single-letter alphabet in $\Pi_2 - \text{SPACE}(s(n)) - \Sigma_1 - \text{SPACE}(s(n))$ for each unbounded fully space-constructible $s(n)$ below $\log(n)$ is

$$L = \{a^n; n \leq \text{ least common multiple of } 1, \cdots, d^{s(n)}\}.$$

There exist unbounded fully space-constructible functions below $\log(n)$, e.g., $s(n) = \log(\min\{i; i \text{ does not divide } n\}) \in O(\log\log(n))$. However, none of these functions can be monotone increasing [1], [2].

Acknowledgment. The author thanks Branislav Rovan for some remarks concerning this work.

REFERENCES

[1] J. H. Chang, O. H. Ibarra, B. Ravikumar, and L. Berman, *Some observations concerning alternating Turing machines using small space*, Inform. Process. Lett., 25 (1987), pp. 1–9; Erratum, 27 (1988), p. 53.

[2] V. Geffert, *Nondeterministic computations in sublogarithmic space and space constructibility*, SIAM J. Comput., 20 (1991), pp. 484–498.

[3] N. Immerman, *Nondeterminstic space is closed under complement*, SIAM J. Comput., 17 (1988), pp. 935–938.

[4] W. J. Savitch, *Relationship between nondeterministic and deterministic tape complexities*, Comput. System Sci., 4 (1970), pp. 177–192.

[5] R. E. Stearns, J. Hartmanis, and P. M. Lewis II, *Hierarchies of memory limited computations*, 1965 IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, pp. 179–190.

[6] R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

[7] A. Szepietowski, *Some remarks on the alternating hierarchy and closure under complement for sublogarithmic space*, Inform. Process. Lett., 33 (1989), pp. 73–78.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 114–135, February 1993

NV-SEQUENTIALITY: A DECIDABLE CONDITION FOR CALL-BY-NEED COMPUTATIONS IN TERM-REWRITING SYSTEMS*

MICHIO OYAMAGUCHI†

Abstract. In 1979 Huet and Levy introduced the class of sequential term-rewriting systems in which call-by-need computations are possible (without look-ahead) and defined the subclass called *strongly sequential systems* for which needed redexes in a given term are effectively found [chapter in *Computational Logic: Essays in Honor of Alan Robinson*, J.-L. Lassez and G. Plotkin, eds., MIT Press, Cambridge, MA, 1991]. This paper introduces a larger subclass that is a natural extension of strong sequentiality and is based on the analysis of both the left-hand sides and part of the right-hand sides (i.e., the nonvariable parts) of systems, whereas strong sequentiality is based on the analysis of left-hand sides alone. This new sequentiality is called *NV-sequentiality*. It is shown that (i) the class of NV-sequential systems properly includes the class of strongly sequential systems, (ii) there exists an algorithm for finding needed redexes for a given term when a system is NV-sequential, and (iii) it is decidable whether an arbitrary left-linear system is NV-sequential.

Key words. term-rewriting system, call-by-need computation, sequentiality, strong sequentiality, left-linear system

AMS(MOS) subject classification. 68Q50

1. Introduction.

A term-rewriting system (TRS) is a set of directed equations (called *rewrite rules*). TRSs are used as abstract interpreters of programming languages and as formula-manipulating systems in various applications, such as theorem proving, reasoning about specifications, program optimization, and transformation (see [5], [6], [9], [13]).

The notion of *call-by-need* computations has turned out to be fruitful in implementation of interpreters for (applicative) programming languages [5], [10], [21], [22]. Here a computation step is said to be call-by-need if the step is necessary to reach a normal form (or to obtain an answer). (This notion is closely related to that of *strictness*, which is important in functional programming (see [1]).) Huet and Levy [8] gave a more complete, theoretical treatment of this subject for TRSs and used the notion of sequentiality given in [11], [22] to define a subclass of TRSs for which the call-by-need computations are possible (without look-ahead). Intuitively, a TRS is said to be sequential if, for any term M that is not in normal form but can reduce to a normal form, there exists a redex in M that we need to compute to obtain to a normal form of M and this redex (which is said to be *needed*) can be determined without look-ahead. However, sequentiality was shown to be undecidable for TRSs in [8]. This undecidability stems from the undecidability of the reachability problem for TRSs. Here the reachability problem is the problem of deciding, for two terms M and N, whether there exists a reduction sequence from M to N. To overcome this difficulty a sufficient condition for sequentiality, called *strong sequentiality*, was introduced in [8]. The notion of strong sequentiality is based on the analysis of left-hand sides of rewrite rules alone. (In other words, it is similar to the sequentiality condition for the set of new rewrite rules defined under the assumption that any redex can reduce to an arbitrary term.) It was shown in [8] that needed redexes for a given term are effectively found when we consider a strongly sequential system. Moreover, whether a TRS is strongly sequential was shown to be decidable in [8]. For a refinement of the notion of strong sequentiality, see [20], and for a related result on strong sequentiality

*Received by the editors December 18, 1989; accepted for publication (in revised form) December 17, 1991. This research was supported in part by the Alexander von Humboldt Foundation.

†Faculty of Engineering, Mie University, Tsu-shi 514, Japan (mo@info.mie-u.ac.jp).

with constructors, see [19]. For an alternative analysis of strong sequentiality and two simplified proofs of the decidability of this property, see [12].

In this paper we introduce a new notion of sequentiality that is a natural extension of strong sequentiality and is based on the analysis of both the left-hand sides and part of the right-hand sides (i.e., the nonvariable parts) of systems, whereas strong sequentiality is based on the analysis of left-hand sides alone. (In other words, this new sequentiality condition is sufficient for ensuring sequentiality under the assumption that for each rewrite rule $\alpha \to \beta$ every redex $\sigma(\alpha)$, where σ is a substitution, can reduce to any term that contains the nonvariable part of β as the prefix (or upper) part. Note that reducing $\sigma(\alpha)$ to $\sigma'(\beta)$, where σ' is another substitution, is permitted, but reducing $\sigma(\alpha)$ to any term is not permitted. Thus this new sequentiality is based on relatively more precise (or detailed) analysis than is strong sequentiality.) Henceforth this new sequentiality is called *NV-sequentiality*, which is an abbreviation of sequentiality with respect to nonvariable parts of right-hand sides.

We first show that the class of NV-sequential systems properly includes the class of strongly sequential systems and that NV-sequentiality is a sufficient condition for sequentiality. (These results show that NV-sequentiality gives a wider class of TRSs, in which call-by-need computations are possible, than does strong sequentiality.)

Next we show that for an NV-sequential system there exists a redex selection algorithm deriving normal forms from given terms whenever they possess normal forms. To show this we prove that the problem of deciding, for a term M and an occurrence (position) u, whether u is a needed redex occurrence for obtaining the normal form of M is reducible to the reachability problem for quasi-ground TRSs, which has been shown to be decidable in [4], [15], [16]. Here a TRS is a quasi-ground system if for every rewrite rule in the TRS the left-hand side is linear and the right-hand side contains no variables. Thus we obtain a redex selection algorithm for NV-sequential TRSs. The algorithm operates in polynomial time. Moreover, whether a left-linear TRS is NV-sequential is shown to be decidable. The proof of this decidability leads to a new (perhaps simplified) proof of the decidability of strong sequentiality that is different from the more algorithmic one given in [8].

This paper is organized as follows. Section 2 is devoted to standard definitions. The notions of sequentiality and strong sequentiality are explained in §3, and the definition of NV-sequentiality is given in §4. In §5 we give a redex selection algorithm for NV-sequential TRSs, and in §6 we show that NV-sequentiality is decidable.

2. Definitions. We use ϵ to denote the empty string and $\emptyset$ to denote the empty set. For a set U we let $\|U\|$ be the cardinality of U. We use $\mathbb{N}$ to denote the set of positive integers and $\mathbb{N}_0$ to denote $\mathbb{N} \cup \{0\}$.

The following definitions and notations are similar to those in [7]. Let X be a finite set of *variables*, and let F be a finite set of *operation symbols* graded by an arity function $a : F \to \mathbb{N}_0$. Let T be the set of terms constructed from X and F. That is, a term either is a variable or is of the form $fM_1 \cdots M_n$ for some $f \in F$ with arity $a(f) = n$ and $M_1, \cdots, M_n \in T$. For any M in T we define $V(M)$ as the set of variables that occur in M. A term M is called a *ground* term if $V(M) = \emptyset$ and is called a *linear* term if no variables occur more than once in M.

Let $F_n = \{f \in F | a(f) = n\}$. We define several functions on terms: For any M in T let $M = x$, where $x \in X$, or $M = fM_1 \cdots M_n$, where $f \in F_n$. Then

(i) the *size* $|M| : |x| = 1$,

$$|fM_1 \cdots M_n| = 1 + |M_1| + \cdots + |M_n|;$$

(ii) the set of *occurrences* $\mathcal{O}(M)$: $\mathcal{O}(x) = \{\epsilon\}$,

$$\mathcal{O}(fM_1 \cdots M_n) = \{\epsilon\} \cup \{iu \,|1 \leq i \leq n, u \in \mathcal{O}(M_i)\};$$

(iii) the subterm M/u at occurrence $u : x/\epsilon = x$,

$$fM_1 \cdots M_n/\epsilon = fM_1 \cdots M_n,$$
$$fM_1 \cdots M_n/iu = M_i/u, \qquad 1 \leq i \leq n;$$

(iv) $M[u \leftarrow N] \in T$, where $u \in \mathcal{O}(M)$ and $N \in T$:

$M[\epsilon \leftarrow N] = N$

$fM_1 \cdots M_n[iu \leftarrow N] = fM_1 \cdots M_{i-1}(M_i[u \leftarrow N])M_{i+1} \cdots M_n, \qquad 1 \leq i \leq n;$

(v) the *height* $h(M)$: $h(x) = 0$,

$$h(fM_1 \cdots M_n) = \begin{cases} 1 + \mathrm{Max}\{h(M_1), \cdots, h(M_n)\} & \text{if } n > 0; \\ 0 & \text{otherwise;} \end{cases}$$

(vi) the operation symbol Occ(M, u) at occurrence u:

$$\mathrm{Occ}(x, \epsilon) = x, \qquad \mathrm{Occ}(fM_1 \cdots M_n, \epsilon) = f,$$
$$\mathrm{Occ}(fM_1 \cdots M_n, iu) = \mathrm{Occ}(M_i, u), \qquad 1 \leq i \leq n;$$

(vii) sub$(M) = \{M/u|u \in \mathcal{O}(M)\}$.

The set of occurrences $\mathcal{O}(M)$ is partially ordered by the prefix ordering $u \leq v$ if and only if $\exists w$ such that $uw = v$. In this case we denote w by v/u. If $u \not\leq v$ and $v \not\leq u$, then u and v are said to be disjoint and are denoted $u|v$. If $u \leq v$ and $u \neq v$, then $u < v$.

Let $\mathcal{O}_X(M)$ be the set of variable occurrences in $\mathcal{O}(M)$, i.e., $\mathcal{O}_X(M) = \{u \in \mathcal{O}(M)|M/u \in X\}$.

A *substitution* σ is a mapping $\sigma : X \to T$, and σ is extended to a mapping from terms to terms: $\sigma(fM_1 \cdots M_n) = f\sigma(M_1) \cdots \sigma(M_n)$ for $f \in F_n$.

A TRS is a finite set E of rules $\alpha \to \beta$ such that $\alpha, \beta \in T$, and $V(\beta) \subseteq V(\alpha)$. Let $L_E = \{\alpha \,|\alpha \to \beta \in E\}$ and $R_E = \{\beta \,|\alpha \to \beta \in E\}$. For any substitution σ and $\alpha \in L_E$, $\sigma(\alpha)$ is called a *redex*. A term M reduces to N at occurrence u if and only if $M/u = \sigma(\alpha)$ and $N = M[u \leftarrow \sigma(\beta)]$ for some substitution σ and rule $\alpha \to \beta \in E$. In this case u is called the redex occurrence of the reduction. We denote this reduction by $M \overset{u}{\to}_E N$. In this notation u and E may be omitted (i.e., $M \to N$) and $\to$ is regarded as a relation over T. Let $\to^+$ and $\to^*$ be the transitive closure and the reflexive-transitive closure of $\to$, respectively. Let $\to^0$ be the identity relation, and let $\to^k = \to \cdot \to^{k-1}$ for $k > 0$. Let $\to^{(k)} = \cup_{i=0}^k \to^i$ for $k \geq 0$.

We define the size of system E by $\sum_{\alpha \to \beta \in E} |\alpha| + |\beta|$, which is denoted size$(E)$.

DEFINITION 2.1. Let $\gamma : M \overset{u_1}{\to} M_1 \cdots \overset{u_k}{\to} M_k$ be a reduction sequence. Then $v \in \mathcal{O}(M)$ is said to be *safe for* γ if there exists no $u_i, 1 \leq i \leq k$, such that $u_i \leq v$.

DEFINITION 2.2. For a term M we use $\mathcal{R}(M)$ to denote the set of redex occurrences in M, i.e., $\mathcal{R}(M) = \{u \in \mathcal{O}(M)|M/u = \sigma(\alpha)$ for some $\alpha \in L_E$ and mapping $\sigma : X \to T\}$. The minimal subset $\mathcal{MR}(M)$ of $\mathcal{R}(M)$ consists of the minimal occurrences of $\mathcal{R}(M)$ in a sense of $\leq$. For example, if $\mathcal{R}(M) = \{1, 2, 21\}$, then $\mathcal{MR}(M) = \{1, 2\}$.

DEFINITION 2.3. A term M is a *normal form* if and only if $\mathcal{R}(M) = \emptyset$. In this case M is also said to be *irreducible*. We denote the set of terms in normal form by NF_E. We use NF if E is clear in the context.

DEFINITION 2.4. A TRS E is said to be *left-linear* if every term α in L_E is linear and is said to be a *quasi-ground* system if for every rule $\alpha \to \beta$ in E, α is linear and β is a ground term. TRS E is said to be nonoverlapping if there are no critical pairs [7], [13].

Notation. For a term M let $U = \{u_1, \cdots, u_{n-1}, u_n\} \subseteq \mathcal{O}(M)$, where $u_1, \cdots, u_n$ are pairwise disjoint. We use $M[u_1 \leftarrow N_1, \cdots, u_{n-1} \leftarrow N_{n-1}, u_n \leftarrow N_n]$ to denote $(M[u_1 \leftarrow N_1, \cdots, u_{n-1} \leftarrow N_{n-1}])[u_n \leftarrow N_n]$. We also use $M[u_i \leftarrow N_i, u_i \in U]$ to denote this term.

Henceforth we will be dealing with a fixed TRS $E = \{\alpha_i \to \beta_i | 1 \leq i \leq n_0\}$ and will that E satisfies the following conditions:

(i) $L_E \cap X = \emptyset$,

(ii) E is left-linear and nonoverlapping.

Note that by (ii) E is confluent, so that if a term M can reduce to a term in normal form, then the normal-form term is unique (see [7], [9]).

3. Sequentiality and needed redex. We will first explain the notion of sequentiality. Intuitively, a TRS E is sequential if and only if for any term M not in normal form there exists a redex in M that we must compute to get to the normal form of M, and this redex can be determined without looking at the subparts of M, which are not yet computed. For a more precise explanation, we need some preliminaries.

First, a new constant symbol Ω (i.e., $a(\Omega) = 0$) will be added to the set F of operation symbols, and the augmented set of terms will be denoted by T_Ω. We denote the set of non-Ω-terms constructed from $F \cup X$ by T. Members of T_Ω will be called Ω-terms, and an irreducible Ω-term will be called an Ω-normal form. Only irreducible terms in T are said to be normal forms. Intuitively, Ω denotes the absence of information and is the least information symbol (in denotational semantics [18]).

An information ordering on T_Ω is defined as follows.

DEFINITION 3.1 [8], [18].

(i) $\Omega \leq M$ for all $M \in T_\Omega$.

(ii) For $f \in F \cup X, fM_1 \cdots M_n \leq fN_1 \cdots N_n$ if and only if $M_i \leq N_i$ for all $i, 1 \leq i \leq n$. Here $n = a(f)$ if $f \in F$, and $n = 0$ if $f \in X$.

(iii) For $f, g \in F \cup X, fM_1 \cdots M_n \not\leq gN_1 \cdots N_m$ if $f \neq g$. Note that if $M \in T$, then $M \leq N$ if and only if $M = N$. We write $M < N$ if $M \leq N$ and $M \neq N$.

By this definition, however, we can easily show the following lemma concerning the ordering $\leq$.

LEMMA 3.1. *Let $M, N \in T_\Omega$. Then the following three statements are equivalent*:

(i) $M \leq N$;

(ii) $\exists u \in \mathcal{O}(M)$ *such that* $M/u \leq N/u$ *and* $M[u \leftarrow \Omega] \leq N[u \leftarrow \Omega]$;

(iii) $M = N$ *or there exists a set* $U \subseteq \mathcal{O}(N)$ *such that occurrences in* U *are pairwise disjoint and* $M = N[u \leftarrow \Omega, u \in U]$.

The proof is straightforward, so it is omitted.

DEFINITION 3.2 [8]. For a term $M \in T$ we use M_Ω to denote the term obtained from M by replacing all variables in M by Ω, i.e., $M_\Omega = M[u \leftarrow \Omega, u \in \mathcal{O}_X(M)]$.

DEFINITION 3.3. For an Ω-term $N \in T_\Omega$ we use $\mathcal{O}_\Omega(N)$ to denote the set of Ω occurrences in $\mathcal{O}(N)$, i.e., $\mathcal{O}_\Omega(N) = \{u \in \mathcal{O}(N) | N/u = \Omega\}$.

Next we define a *normal-form predicate* nf_E as follows:

DEFINITION 3.4 [8]. For $M \in T_\Omega$, $\mathrm{nf}_E(M) = \mathrm{TRUE}$ if and only if $\exists N \in \mathrm{NF}_E$ such that $M \to_E^* N$, that is, M can reduce to a normal form. $\mathrm{nf}_E(M) = \mathrm{FALSE}$ if and only if

$\mathrm{nf}_E(M) \neq$ TRUE. We may omit the subscript E if E is clear in the context.

Note that NF_E is a monotonic predicate under the assumption FALSE $\leq$ TRUE, i.e., $M \leq N$ implies $\mathrm{NF}_E(M) \leq \mathrm{NF}_E(N)$.

Assume that $\mathrm{nf}_E(M) =$ TRUE and that M contains Ω. Then note that all Ω's in M must be eliminated in the reductions to the normal form. We now define needed redex occurrences as follows.

DEFINITION 3.5. Let $\mathrm{nf}_E(M) =$ TRUE for $M \in T_\Omega$. Then a redex occurrence $u \in \mathcal{R}(M)$ is said to be *needed for the normal form* if and only if $\mathrm{nf}_E(M[u \leftarrow \Omega]) =$ FALSE. (This definition is equivalent to that in [5].) A reduction $M \xrightarrow{u} N$ is said to be a *call-by-need* reduction (or computation) if u is needed for the normal form of M.

It has been shown that in λ-calculus the leftmost reduction for any term is call-by-need, but in TRSs the leftmost outmost reduction is usually not call-by-need (see [2], [8]).

We now consider the case for which TRS E is left-linear and nonoverlapping. Let M be an Ω-term not in normal form. If there exists a reduction sequence $\gamma : M \to^* N$ for some $N \in \mathrm{NF}_E$, then at least one outermost redex of M (i.e., M/u for some u in $\mathcal{MR}(M)$ (see Definition 2.2)) must be evaluated in γ. As a stronger result, it was shown in [8, Lemma 4.3] that there exists u in $\mathcal{MR}(M)$ needed for the normal form (i.e., M/u must be evaluated for all reduction sequences $\gamma : M \to^* N$ such that $N \in \mathrm{NF}_E$). Generally, such an occurrence u depends on the subterms $M/u', u' \in \mathcal{MR}(M)$. For example, in the case for which $E \supseteq \{f(A,B,x) \to C, f(x,A,B) \to C, f(B,x,A) \to C\}$ a needed occurrence of an Ω-term $M = f(M_1, M_2, M_3)$ depends on the contexts M_1, M_2, M_3 (e.g., if M_1, M_2, M_3 are redexes and there is no reduction sequence such that $M_3 \to^* A$ or $M_3 \to^* B$, then occurrences 1 and 2 are needed, but if $M_3 \to^* B$, then occurrence 1 may not be needed [8]).

We now explain the notion of sequentiality of the predicate nf_E. The notion is based on the assumption that there exists a needed occurrence independent of the subterms $M/u, u \in \mathcal{MR}(M)$. Let $M' = M[u \leftarrow \Omega, u \in \mathcal{MR}(M)]$, where $\mathcal{MR}(M) \neq \emptyset$. Note that $\mathrm{nf}_E(M') =$ FALSE. Then the sequentiality condition is that there exists an occurrence $u \in \mathcal{MR}(M)$ such that if $N \geq M'$ and $\mathrm{nf}_E(N) =$ TRUE, then $N/u \neq \Omega$. That is, sequentiality ensures that this occurrence u is a needed occurrence of M and that it can be determined independently of the contents of the M/v's, $v \in \mathcal{MR}(M)$. The following definition of a sequential predicate is given in [11] and [8].

DEFINITION 3.6. Let the truth values be ordered by FALSE $\leq$ TRUE. Let $\mathcal{P}$ be a monotonic predicate on T_Ω. An occurrence u of a term M is said to be an *index* of $\mathcal{P}$ in M if and only if (i) $M/u = \Omega$ and (ii) $N \geq M$ and $\mathcal{P}(N) =$ TRUE (where $N \in T_\Omega$) imply $N/u \neq \Omega$. (Note that $\mathcal{P}(M)$ is false if M has an index of $\mathcal{P}$.) We use $I(\mathcal{P}, M)$ to denote the set of indices of $\mathcal{P}$ in M. $\mathcal{P}$ is said to be *sequential at* M if and only if whenever $\mathcal{P}(M) =$ FALSE and there exists N such that $N \geq M$ and $\mathcal{P}(N) =$ TRUE, it follows that there exists an index of $\mathcal{P}$ in M.

DEFINITION 3.7 [8]. TRS E is sequential if and only if the predicate nf_E is sequential at every Ω-normal form.

Unfortunately, sequentiality of TRSs is known to be undecidable, and indices of nf_E are not computable in general [8]. A sufficient condition for sequentiality, called strong sequentiality, is given in [8]. The notion of strong sequentiality is based on the analysis of the left-hand sides of rules alone, and the contents of the right-hand sides are ignored. Precisely, strong sequentiality is defined as follows.

DEFINITION 3.8 [8]. From the set E of rewrite rules, a new reduction $\to_{\mathrm{st}}$ is defined as follows: $M \to_{\mathrm{st}} N$ if and only if $M/u = \sigma(\alpha)$ and $N = M[u \leftarrow N']$ for some

$u \in \mathcal{O}(M)$, redex $\sigma(\alpha)$ (where $\alpha \in L_E$ and $\sigma : X \to T_\Omega$), and $N' \in T_\Omega$. Note that any term N' is allowed as the right-hand-side term of the rule, and L_E is the set of left-hand-side terms of E. We define a predicate snf_E as follows: $\mathrm{snf}_E(M) = \mathrm{TRUE}$ if and only if $\exists\ N$ such that $M \to_{\mathrm{st}}^* N$ and N is in normal form; (i.e., $N \in \mathrm{NF}_E$). (Note that $\mathrm{snf}_E = \mathrm{nf}_{\mathrm{st}}$ if we denote by st the set of the new rules $\{M \to N | M \to_{\mathrm{st}} N\}$.) A TRS E is said to be *strongly sequential* if and only if snf_E is sequential at every Ω-normal form.

We will see in §4 that if a TRS E is strongly sequential, then E is sequential, and for every Ω-normal form N if an occurrence u is an index of snf_E in N, then u is also an index of nf_E in N, although these results have already been shown in [8]. Thus strong sequentiality is a sufficient condition for sequentiality, and Huet and Levy [8] showed an efficient algorithm to compute indices of snf_E in a given Ω-term. Hence by the previous arguments it follows that a needed occurrence of a given Ω-term M is efficiently computed if TRS E is strongly sequential, $\mathrm{nf}_E(M) = \mathrm{TRUE}$, and $\mathcal{R}(M) \neq \emptyset$. Moreover, it is shown in [8] that if $\mathrm{nf}_E(M) = \mathrm{TRUE}$, then every call-by-need reduction sequence from M is normalizing, i.e., reaches the normal form in finite steps. Thus we can obtain a redex selection algorithm that derives the normal form from a given term whenever it has the normal form. Also, whether a TRS is strongly sequential has been shown to be decidable [8].

As a related work, it has been shown that in left-linear and nonoverlapping TRSs if a given term has the normal form, then every sequence of parallel-outermost reductions (i.e., simultaneously rewriting all outermost redexes) from the term is normalizing (see [3], [14]).

4. NV-sequentiality. In this section we introduce a condition called NV-sequentiality, which is more general than strong sequentiality. The notion of NV-sequentiality is based on the analysis of both the left-hand sides and part of the right-hand sides (i.e., the nonvariable parts) of systems, whereas strong sequentiality is based on the analysis of the left-hand sides alone.

DEFINITION 4.1. From the set E of rewrite rules, a new reduction $\to_{\mathrm{nv}}$ is defined as follows: $M \to_{\mathrm{nv}} N$ if and only if $\exists u \in \mathcal{O}(M) \exists \alpha \leftarrow \beta \in E$ such that $M/u \geq \alpha_\Omega$ and $N = M[u \leftarrow N']$ for some N', where $N' \geq \beta_\Omega$. Note that Q is a redex of E if and only if $Q \geq \alpha_\Omega$ for some $\alpha \in L_E$, since E is left-linear. So under the reduction $\to_{\mathrm{nv}}$ any redex $Q \geq \alpha_\Omega$ can reduce to any term N' such that $N' \geq \beta_\Omega$, where $\alpha \to \beta \in E$.

Note that $\to_{\mathrm{st}} \supseteq \to_{\mathrm{nv}} \supseteq \to_E$ hold, and these reductions have the same set of redexes (and the same normal forms). Using $\to_{\mathrm{nv}}$, we now define NV-sequentiality as follows.

DEFINITION 4.2. A predicate *term* is defined as follows: $\mathit{term}(M) = \mathrm{TRUE}$ if and only if $\exists N$ such that $M \to_{\mathrm{nv}}^* N$ and $N \in T$. Note that N is a term and N does not contain Ω. A TRS E is said to be NV-sequential if and only if *term* is sequential at every Ω-normal form, that is, for any M in Ω-normal form if (i) $\mathit{term}(M) = \mathrm{FALSE}$ and (ii) there exists a term N such that $N \geq M$ and $\mathit{term}(N) = \mathrm{TRUE}$, then there exists an index of *term* in M. Note that condition (ii) is always true. So from now on we will omit (ii) from the definition of NV-sequentiality. The predicate *term* is defined over the new reduction $\to_{\mathrm{nv}}$, but it is not defined over $\to_E$.

Note. To define NV-sequentiality we used the predicate *term*. Of course, we can use a predicate similar to nf_E to define another sufficient condition for sequentiality. That is, we can define a predicate $\mathrm{nvnf}_E(M)$ as follows: $\mathrm{nvnf}_E(M) = \mathrm{TRUE}$ if and only if $\exists N \in \mathrm{NF}_E$ such that $M \to_{\mathrm{nv}}^* N$. Using this predicate, we can define a class of TRSs such that nvnf_E is sequential at every Ω-normal form. In this paper we did not adopt this definition, since it will be very difficult to obtain an (efficient) algorithm for finding indices of nvnf_E.

(We conjecture that the class of TRSs satisfying this sequentiality properly includes the class of NV-sequential TRSs.)

We will give a relationship between sequential TRSs, NV-sequential TRSs, and strongly sequential TRSs. We will show that the class of NV-sequential TRSs is included in the class of sequential TRSs and properly includes the class of strongly sequential TRSs. The following two technical lemmas are used to obtain this result.

LEMMA 4.1. *Assume that* $\mathrm{NF}_E \neq \emptyset$. *Then the following statements hold*:

(i) $\mathrm{term}(M) = \mathrm{TRUE} \Rightarrow \mathrm{snf}_E(M) = \mathrm{TRUE}$;

(ii) $\mathrm{nf}_E(M) = \mathrm{TRUE} \Rightarrow \mathrm{term}(M) = \mathrm{TRUE}$.

Proof. (i) By $\mathrm{term}(M) = \mathrm{TRUE}$ we have $M \to_{\mathrm{nv}}^* N$ for some $N \in T$, so that $M \to_{\mathrm{st}}^* N$ holds. If N/u is a redex for some $u \in \mathcal{O}(N)$, then we have a reduction $N \to_{\mathrm{st}} N[u \leftarrow N']$, where $N' \in \mathrm{NF}_E$. Hence $\mathrm{snf}_E(M) = \mathrm{TRUE}$ by repeating the above reductions. (ii) By $\mathrm{nf}_E(M) = \mathrm{TRUE}$ we have $M \to_E^* N$ for some $N \in \mathrm{NF}_E$. By $N \in T$ $\mathrm{term}(M)$ is true. □

LEMMA 4.2. *Let M be an* Ω*-normal form. Let* $\mathrm{NF}_E \neq \emptyset$. *Then the following statements hold*:

(a) $u \in I(\mathrm{snf}_E, M) \Rightarrow u \in I(\mathrm{term}, M)$;

(b) $u \in I(\mathrm{term}, M) \Rightarrow u \in I(\mathrm{nf}_E, M)$.

Proof. By Definition 3.6, $u \in I(\mathcal{P}, M)$ if and only if (i) $M/u = \Omega$ and (ii) $N \geq M$ and $\mathcal{P}(N) = \mathrm{TRUE} \Rightarrow N/u \neq \Omega$ for a predicate $\mathcal{P}$.

Proof of (a). Let $u \in I(\mathrm{snf}_E, M)$. That is, the above (i) and (ii) hold where $\mathcal{P} = \mathrm{snf}_E$. Assume that $N \geq M$ and $\mathrm{term}(N) = \mathrm{TRUE}$. Then $\mathrm{snf}_E(N) = \mathrm{TRUE}$ holds by (i) of Lemma 4.1. So $N/u \neq \Omega$ by (ii) above, where $\mathcal{P} = \mathrm{snf}_E$. Thus the above (i) and (ii) also hold in the case for which $\mathcal{P} = \mathrm{term}$.

Proof of (b). This proof is similar to that of (a), except that Lemma 4.1(ii), i.e., $\mathrm{nf}_E(M) = \mathrm{TRUE} \Rightarrow \mathrm{term}(M) = \mathrm{TRUE}$ is used (instead of Lemma 4.1(i)). □

LEMMA 4.3. *Let* TRS *E be left-linear and nonoverlapping. Let* $\mathrm{NF}_E \neq \emptyset$. *Then the following statements hold*:

(i) *E is strongly sequential* $\Rightarrow$ *E is* NV*-sequential*;

(ii) *E is* NV*-sequential* $\Rightarrow$ *E is sequential.*

Proof. Let M be an Ω-normal form.

Prooff of (i). Assume that

$$(1) \qquad \mathrm{term}(M) = \mathrm{FALSE} \text{ and } \exists N, (N \geq M) \wedge (\mathrm{term}(N) = \mathrm{TRUE}).$$

By (1) $\mathrm{snf}(M) = \mathrm{FALSE}$, since $\mathrm{term}(M) = \mathrm{FALSE}$ implies that M contains Ω and M is in Ω-normal form. Obviously, $\exists N'$ such that $(N' \geq M) \wedge (\mathrm{snf}_E(N') = \mathrm{TRUE})$. (For example, choose $N' \in T$ such that $N' \geq M$.) Hence strong sequentiality of E implies that there exists an index $u \in I(\mathrm{snf}_E, M)$. Hence by (a) of Lemma 4.2, $u \in I(\mathrm{term}, M)$. Thus E is NV-sequential.

Proof of (ii). Assume that

$$(2) \qquad \mathrm{nf}_E(M) = \text{ FALSE and } \exists N \text{ such that } (N \geq M) \wedge (\mathrm{nf}_E(N) = \text{ TRUE}).$$

We first show that $\mathrm{term}(M) = \mathrm{FALSE}$. Since M is an Ω-normal form, there is no redex in M. So $\mathrm{nf}_E(M) = \mathrm{FALSE}$ in (2) implies that M contains at least one Ω. Thus $\mathrm{term}(M) = \mathrm{FALSE}$, as claimed. Hence NV-sequentiality of E implies that there exists an index $u \in I(\mathrm{term}, M)$. So by (b) of Lemma 4.2, $u \in I(\mathrm{nf}_E, M)$. Thus E is sequential. □

We can obtain a TRS E that is NV-sequential but not strongly sequential.

Example 4.1. Let

$$E = \{f(f(a,x), f(b,y)) \to f(e,e), f(f(x,a), f(c,y)) \to f(e,e), f(d,d) \to f(e,e)\},$$

where $a, b, c, d, e \in F_0, f \in F_2$, and $x, y \in X$. Note that E is similar to that given in [8, p. 27], and E is not strongly sequential since $f(f(\Omega,\Omega), f(f(\Omega,\Omega),\Omega))$ has no index. (However, note that occurrence 22 in this term is an index of term.) We can show that E is NV-sequential. We omit the proof. (A reader who is interested in the proof may refer to the previous version of this paper [17].) A general algorithm for checking NV-sequentiality will be given in §6. □

By Lemma 4.3 and Example 4.1 we have the following theorem.

THEOREM 4.1. (i) *The class of NV-sequential TRSs properly includes the class of strongly sequential TRSs.*

(ii) *The class of NV-sequential TRSs is included in the class of sequential TRSs.*

Note. Properness of (ii) in Theorem 4.1 can be proved since NV-sequentiality is shown to be decidable in §6, but sequentiality is undecidable in general. As a concrete candidate for a sequential TRS that is not NV-sequential, we give the following TRS E_2.

Example 4.2. Let

$$E_2 = \{f(g(a,x),a) \to c, f(g(x,a),b) \to c, f(k(a),x) \to c, g(b,b) \to h(b), h(x) \to k(x)\},$$

where $a, b, c \in F_0, f, g \in F_2, h, k \in F_1$, and $x \in X$. Note that E_2 is similar to that given in [8, p. 26]. We can show that E_2 is not NV-sequential, because $f(g(\Omega,\Omega),\Omega)$ has no index of term. (Note that occurrence 2 is not an index, since $f(g(b,b),\Omega) \to_{\mathrm{nv}} f(h(b),\Omega) \to_{\mathrm{nv}} f(k(a),\Omega) \to_{\mathrm{nv}} c$.) The author thinks that E_2 is sequential, although the formal proof has not been given.

5. Redex selection algorithm. In this section we consider the problem of deciding, for an Ω-term N and an occurrence $u \in \mathcal{O}(N)$, whether u is an index of term in N, i.e., $u \in I(\text{term}, N)$. We show that this problem is decidable. From now on we call this problem the *term I-problem*.

Notation. For $N \in T_\Omega$ let $I_t(N) = I(\text{term}, N)$.

Let M be a term in T, where $\mathcal{MR}(M) \neq \emptyset$, and let TRS E be NV-sequential, where E is left-linear and nonoverlapping. Let $M' = M[u \leftarrow \Omega, u \in \mathcal{MR}(M)]$. Note that $\text{term}(M') = \text{FALSE}$ by $\mathcal{MR}(M) \neq \emptyset$. So NV-sequentiality of E ensures that some occurrence u in $\mathcal{MR}(M)$ is an index of term in M' (i.e., $u \in I_t(M')$). By Lemma 4.2(b), u is also an index of nf in M', so that u is a needed redex of M for the normal form by Definition 3.5. Let $M \xrightarrow{u}_E N$ be the call-by-need reduction. If N is reducible, i.e., $\mathcal{MR}(N) \neq \emptyset$, then we repeat the above procedure for N (instead of M) until we obtain the normal form. As was explained in §3, it is shown in [8] that if $\mathrm{nf}_E(M) = \text{TRUE}$, then every call-by-need reduction sequence from M is finite-terminating and eventually leads to the normal form. Thus, using an algorithm for deciding the above term I-problem, we can obtain a redex selection algorithm that gives a reduction sequence from a term to the normal form whenever the term can reduce to the normal form.

To obtain an algorithm for deciding the term I-problem, we need some preliminaries. The following definition is given in [8].

DEFINITION 5.1 [8]. For Ω-terms $M, N \in T_\Omega$, M and N are said to be *compatible*, written $M \uparrow N$, if and only if there exists an Ω-term Q such that $M \leq Q$ and $N \leq Q$. We write $M \uparrow\!+ N$ if $M \uparrow N$ and $M \neq \Omega$.

DEFINITION 5.2 [8]. The least upper bound of two compatible Ω-terms M and N, written $M \sqcup N$, is defined as follows:

$$\Omega \sqcup M = M \sqcup \Omega = M$$

$(fM_1 \cdots M_n) \sqcup (fN_1 \cdots N_n) = f(M_1 \sqcup N_1, \cdots, M_n \sqcup N_n)$, where $f \in F \cup X$. Note that if $M \uparrow N$, then $M \sqcup N$ is defined and $M \leq M \sqcup N$ and $N \leq M \sqcup N$.

We now define a new kind of rewrite rules associated with a TRS E as follows.

DEFINITION 5.3. For Ω-terms $M, N \in T_\Omega, M \rightarrow_\omega N$ if and only if $M/u \uparrow + \alpha_\Omega$ and $N = M[u \leftarrow \beta_\Omega]$ for some $u \in \mathcal{O}(M), \alpha \rightarrow \beta \in E$. We call this reduction an ω-reduction, and we call M/u an ω-redex.

Note. The reduction $\rightarrow_\omega$ in [8] is the same as that of Definition 5.3., except that Ω is substituted for β_Ω.

Note. Let M be an Ω-normal form. Then there is no redex $\sigma(\alpha)$ in M, where $\alpha \in L_E$ and $\sigma : X \rightarrow T_\Omega$, but M may have an ω-redex, i.e., $M/u \uparrow + \alpha_\Omega$ for some $u \in \mathcal{O}(M)$ and $\alpha \in L_E$.

We first explain a relationship between this new reduction $\rightarrow_\omega$ and $\rightarrow_{\text{nv}}$ (defined in §4). Using this relationship, we will show that the term I-problem reduces to the reachability problem for quasi-ground TRSs, which is shown to be decidable in [4], [15], [16].

LEMMA 5.1. (a) *Let $M \rightarrow_\omega \beta_\Omega$ where $M \uparrow + \alpha_\Omega$ and $\alpha \rightarrow \beta \in E$. Then there exists $M' \geq M$ such that $M' \rightarrow_{\text{nv}} Q$ for any $Q \geq \beta_\Omega$.*

(b) *Let $M \rightarrow_{\text{nv}} M'$ and $N \leq M$ where $N, M, M' \in T_\Omega$. Then either $N \leq M'$ or there exists $N' \in T_\Omega$ such that $N \rightarrow_\omega N'$ and $N' \leq M'$. (This implies that $\exists N'$ such that $N \rightarrow_\omega^* N'$ and $N' \leq M'$.)*

Proof. (a) The proof is obvious. (b) By $M \rightarrow_{\text{nv}} M'$ we have $M/u \geq \alpha_\Omega$ and $M' = M[u \leftarrow Q]$ for some $u \in \mathcal{O}(M), \alpha \rightarrow \beta \in E$, and $Q \geq \beta_\Omega$. We first consider the case where there exists $v \in \mathcal{O}(N)$ such that $v \leq u$ and $N/v = \Omega$. In this case $M[v \leftarrow \Omega] \geq N[v \leftarrow \Omega] = N$ holds, so that $M' = M[v \leftarrow M'/v] \geq N$ holds, as claimed. Next consider the case where there exists no $v \in \mathcal{O}(N)$ such that $v \leq u$ and $N/v = \Omega$. In this case, by $M \geq N$ we have

$$(3) \qquad u \in \mathcal{O}(N) \wedge \text{Occ}(M, u) = \text{Occ}(N, u) \neq \Omega.$$

Since $M/u \geq \alpha_\Omega$ and $M/u \geq N/u$, we have $N/u \uparrow \alpha_\Omega$, so that $N/u \uparrow + \alpha_\Omega$ by (3). Thus $N/u \rightarrow_\omega \beta_\Omega$ by the definition of $\rightarrow_\omega$. Let $N' = N[u \leftarrow \beta_\Omega]$. Obviously, $N \rightarrow_\omega N'$ and $N' \leq M'$ hold, since $M' = M[u \leftarrow Q] \geq M[u \leftarrow \beta_\Omega] \geq N[u \leftarrow \beta_\Omega] = N'$. □

Using this relationship between $\rightarrow_{\text{nv}}$ and $\rightarrow_\omega$, we will first give a characterization of $I_t(M)$, where M is an Ω-term. That is, we will prove Lemma 5.3, which says that for an Ω-term $M \in T_\Omega - \{\Omega\}$ an occurrence $u \in \mathcal{O}(M)$ (where $M/u = \Omega$) is no index of term, i.e., $u \notin I_t(M)$ if and only if there exist $v \in \mathcal{O}(M)$, where $v < u$, and $N \in T_\Omega$ such that $(M[u \leftarrow x])/v \rightarrow_\omega^* N$ and $N \uparrow + \alpha_\Omega$ for some $\alpha \in L_E$, where x is a variable and L_E is the set of left-hand sides of E. By using this characterization the term I-problem will be shown to be decidable. For this purpose we will need some definitions and lemmas.

DEFINITION 5.4. For Ω-terms M', M and an occurrence $u \in \mathcal{O}(M')$, if $M' \geq M \wedge M'/u = \Omega \wedge \text{term}(M') = \text{TRUE}$, then we say that M' witnesses $u \notin I_t(M)$.

The following technical lemma comprises the *if* part of Lemma 5.3.

LEMMA 5.2. *Let $M \in T_\Omega$, and let $u \in \mathcal{O}(M)$, where $M/u = \Omega$. Let $M[u \leftarrow x] \rightarrow_\omega^* N$ and $N \uparrow + \alpha_\Omega$, where x is a variable and $\alpha \in L_E$. Then $u \notin I_t(M)$.*

Proof. Let $M_1 = M[u \leftarrow x]$ and $M_1 \rightarrow_\omega^k N$ for some $k \geq 0$, where $N \uparrow + \alpha_\Omega$. We will prove this lemma by induction on k.

Basis: $k = 0$. In this case $M_1 = N \uparrow + \alpha_\Omega$. Let

$$(4) \qquad Q = M_1 \sqcup \alpha_\Omega.$$

Then $Q/u = x$ holds by $M_1/u = x$. Moreover, note that α_Ω contains no variable, so that by (4), $\exists u' \leq u$ such that $\alpha_\Omega/u' = \Omega$. Thus we have

$$(5) \qquad Q[u \leftarrow \Omega] \geq \alpha_\Omega.$$

We now show that $Q[u \leftarrow \Omega]$ witnesses $u \notin I_t(M)$, i.e., (i) $Q[u \leftarrow \Omega] \geq M$, (ii) $Q[u \leftarrow \Omega]/u = \Omega$, and (iii) $\mathrm{term}(Q[u \leftarrow \Omega]) = \mathrm{TRUE}$. Obviously, (ii) holds. Since $Q[u \leftarrow \Omega] \geq M_1[u \leftarrow \Omega] = M$ holds by (4), (i) holds. It remains to show (iii). By (5) the definition of $\rightarrow_{\mathrm{nv}}$ ensures that $Q[u \leftarrow \Omega] \rightarrow_{\mathrm{nv}} N'$ for any $N' \geq \beta_\Omega$, where $\alpha \rightarrow \beta \in E$. Thus $\mathrm{term}(Q[u \leftarrow \Omega]) = \mathrm{TRUE}$, i.e., (iii) holds, as claimed.

Induction step: $k > 0$. Let $M_1 = M[u \leftarrow x] \rightarrow_\omega M_2 = (M[u \leftarrow x])[v \leftarrow \bar{\beta}_\Omega] \rightarrow_\omega^{k-1} N$, where

$$M[u \leftarrow x]/v \uparrow\!+\, \bar{\alpha}_\Omega \quad \text{for some } v \in \mathcal{O}(M[u \leftarrow x]) \text{ and } \bar{\alpha} \rightarrow \bar{\beta} \in E.$$

Let

(6) $$Q = M[u \leftarrow x]/v \sqcup \bar{\alpha}_\Omega.$$

There are two subcases: (a) $u|v$ and (b) $v < u$. Note that $u \leq v$ is impossible by $M[u \leftarrow x]/u = x$.

Case (a): $u|v$. Note that $M_2 = (M[v \leftarrow \bar{\beta}_\Omega])[u \leftarrow x]$ and $M_2 \rightarrow_\omega^{k-1} N$. So the induction hypothesis ensures that there exists Q_1 witnessing $u \notin I_t(M[v \leftarrow \bar{\beta}_\Omega])$, i.e.,

(7) $$Q_1 \geq M[v \leftarrow \bar{\beta}_\Omega] \wedge Q_1/u = \Omega \wedge \mathrm{term}(Q_1) = \mathrm{TRUE}.$$

Using (6) and (7), we can show that $Q_1[v \leftarrow Q]$ witnesses $u \notin I_t(M)$ for Q in (6), i.e., (i) $Q_1[v \leftarrow Q] \geq M$, (ii) $Q_1[v \leftarrow Q]/u = \Omega$, and (iii) $\mathrm{term}(Q_1[v \leftarrow Q]) = \mathrm{TRUE}$. Note that $Q \geq M[u \leftarrow x]/v = M/v$ by (6) and $u|v$, so that (i) holds, since $Q_1 \geq M[v \leftarrow \bar{\beta}_\Omega] \geq M[v \leftarrow \Omega]$ by (7). Since $Q_1[v \leftarrow Q]/u = Q_1/u = \Omega$ by (7), (ii) holds. It remains to show (iii). Note that $Q_1/v \geq \bar{\beta}_\Omega$ holds by (7) and $Q \geq \bar{\alpha}_\Omega$ holds by (6), where $\bar{\alpha} \rightarrow \bar{\beta} \in E$. So the definition of $\rightarrow_{\mathrm{nv}}$ ensures that $Q \rightarrow_{\mathrm{nv}} Q_1/v$. Thus $Q_1[v \leftarrow Q] \rightarrow_{\mathrm{nv}} Q_1[v \leftarrow Q_1/v] = Q_1$ holds. Since $\mathrm{term}(Q_1) = \mathrm{TRUE}$ by (7), it follows that $\mathrm{term}(Q_1[v \leftarrow Q]) = \mathrm{TRUE}$, i.e., (iii) holds, as claimed.

Case (b): $v < u$. In this case $M[u \leftarrow x]/v = (M/v)[u' \leftarrow x]$, where $vu' = u$. By $M[u \leftarrow x]/v \uparrow\!+\, \bar{\alpha}_\Omega$, the proof of the basis case of $k = 0$ ensures that there exists Q' witnessing $u' \notin I_t(M/v)$, i.e.,

(8) $$Q' \geq M/v \wedge Q'/u' = \Omega \wedge \mathrm{term}(Q') = \mathrm{TRUE}.$$

Obviously, there exists a term M_0 in T such that $M_0 \geq M$. Let $M' = M_0[v \leftarrow Q']$. Then we can prove that M' witnesses $u \notin I_t(M)$, i.e., (i) $M' \geq M$, (ii) $M'/u = \Omega$, and (iii) $\mathrm{term}(M') = \mathrm{TRUE}$. Obviously, (i) holds, since $M' = M_0[v \leftarrow Q'] \geq M[v \leftarrow Q'] \geq M$ by (8), and (ii) holds, since $M'/u = Q'/u' = \Omega$ by (8). Moreover, since $M_0 \in T$ and $\mathrm{term}(Q') = \mathrm{TRUE}$ by (8), (iii) holds, that is , $\mathrm{term}(M') = \mathrm{term}(M_0[v \leftarrow Q']) = \mathrm{TRUE}$. □

We are now ready to prove the main lemma in this section.

LEMMA 5.3. *Let M be an Ω-term, and let $u \in \mathcal{O}(M)$, where $M/u = \Omega$. Then $u \notin I_t(M)$ if and only if there exist $v \in \mathcal{O}(M)$, where $v < u$, and $N \in T_\Omega$ satisfying the following condition*:

(9) $$M[u \leftarrow x]/v \rightarrow_\omega^* N \quad \text{and} \quad N \uparrow\!+\, \alpha_\Omega \quad \text{for some } \alpha \in L_E \text{ where } x \text{ is a variable.}$$

Proof: *if part*. Note that $M[u \leftarrow x]/v = (M/v)[u' \leftarrow x]$, where $u = vu'$. So by Lemma 5.2 there exists $M' \in T_\Omega$ witnessing $u' \notin I_t(M/v)$, i.e.,

(10) $$M' \geq M/v \wedge M'/u' = \Omega \wedge \mathrm{term}(M') = \mathrm{TRUE}.$$

It follows that there exists $Q \in T_\Omega$ such that Q witnesses $u \notin I_t(M)$, because, for example, we can take $M_0[v \leftarrow M']$ as Q, where M_0 is a term in T such that $M_0 \geq M$. (Note that $Q = M_0[v \leftarrow M'] \geq M, Q/u = M'/u' = \Omega$, and term$(Q)$ = TRUE by (10).) Hence $u \notin I_t(M)$.

Proof: *only-if part*. Since $u \notin I_t(M)$, there exists $Q \in T_\Omega$ such that

(11) $$Q \geq M \wedge Q/u = \Omega \wedge \text{ term}(Q) = \text{ TRUE}.$$

So there exists a reduction sequence

$$\gamma : Q = Q_1 \xrightarrow{u_1}_{\text{nv}} Q_2 \xrightarrow{u_2}_{\text{nv}} \cdots \xrightarrow{u_{n-1}}_{\text{nv}} Q_n,$$

where $Q_n \in T$ and u_i is the redex occurrence of $Q_i \rightarrow_{\text{nv}} Q_{i+1}, 1 \leq i < n$. In this reduction sequence γ note that there exists $u_i, 1 \leq i < n$, such that $u_i < u$, since $Q/u = \Omega$ and $Q_n \in T$. Let $j(1 \leq j < n)$ be the smallest number satisfying $u_j < u$. Let $v = u_j$, and let u' satisfy $vu' = u$. Note that

(12) $$Q/v \rightarrow^*_{\text{nv}} Q_j/v \wedge Q_j/v \geq \alpha_\Omega$$

for some $\alpha \in L_E$ and that occurrence u' is safe for this reduction $Q/v \rightarrow^*_{\text{nv}} Q_j/v$, so that $Q_j/vu' = \Omega$. Let $\bar{Q}/v = (Q/v)[u' \leftarrow x]$ and $\bar{Q}_j/v = (Q_j/v)[u' \leftarrow x]$. Then

(13) $$\bar{Q}/v \rightarrow^*_{\text{nv}} \bar{Q}_j/v \quad \text{and} \quad \bar{Q}_j/v \geq \alpha_\Omega,$$

since $\bar{Q}_j/v \geq Q_j/v \geq \alpha_\Omega$ by (12). Note that by (11)

(14) $$\bar{Q}/v = (Q/v)[u' \leftarrow x] \geq (M/v)[u' \leftarrow x] = M[u \leftarrow x]/v.$$

We can now prove that condition (9) holds, i.e., the *only-if* part. Since (13) and (14) hold, by repeatedly invoking Lemma 5.1(b) we have

(15) $$M[u \leftarrow x]/v \rightarrow^*_\omega N_1 \wedge N_1 \leq \bar{Q}_j/v \quad \text{for some } N_1.$$

By (13) and (15), $N_1 \uparrow \alpha_\Omega$ holds. So if $N_1 \neq \Omega$, then $N_1 \uparrow\!+ \alpha_\Omega$ holds, so that condition (9) holds. If $N_1 = \Omega$, then by $M[u \leftarrow x]/v \neq \Omega$ there exists $R \in T_\Omega$ such that $M[u \leftarrow x]/v \rightarrow^*_\omega R \rightarrow_\omega N_1 = \Omega$ and $R \neq \Omega$. Obviously, $R \uparrow\!+ \alpha'_\Omega$ for some $\alpha' \in L_E$. (In this case $\alpha' \rightarrow \beta' \in E$ and $\beta'_\Omega = \Omega$ for some β'.) In either case condition (9) holds. □

The following lemma is a direct consequence of Lemma 5.3.

LEMMA 5.4. *Let M be an Ω-term, where $M \neq \Omega$, and let $u \in \mathcal{O}(M)$ where $M/u = \Omega$. Then $u \notin I_t(M)$ if and only if there exist $v \in \mathcal{O}(M)$, where $v < u$, and $\beta \in R_E$ such that*

$$(M[u \leftarrow x])/v \rightarrow^*_\omega \beta_\Omega,$$

where x is a variable.

Proof. The *only-if* part obviously holds by Lemma 5.3. So consider the *if* part. Let $\gamma : M[u \leftarrow x]/v \rightarrow^*_\omega \beta_\Omega$. Let u' satisfy $vu' = u$. Then, since β_Ω does not contain x, some $v_1 \in \mathcal{O}(M[u \leftarrow x]/v)$ such that $v_1 < u'$ must be an ω-redex occurrence in γ. Thus for such a v_1

$$M[u \leftarrow x]/vv_1 \rightarrow^*_\omega N \quad \text{and} \quad N \uparrow\!+ \bar{\alpha}_\Omega$$

for some $N \in T_\Omega$ and $\bar{\alpha} \in L_E$. Hence by Lemma 5.3 $u \notin I_t(M)$ holds. □

We have shown that the condition of Lemma 5.4 is an "if and only if" condition for ensuring that a given occurrence is no index of term. Now we consider how to check this condition. We will first construct a new set $K(E)$ of rewrite rules from E such that $K(E)$ is a quasi-ground TRS, and we will show a close relationship between $\rightarrow_\omega$ and $\rightarrow_{K(E)}$. Next, using this relation, we will prove that in the condition of Lemma 5.4 $\rightarrow_\omega^*$ can be replaced by $\rightarrow_{K(E)}^*$, that is, for $M \in T_\Omega - \{\Omega\}$ an occurrence $u \in \mathcal{O}(M)$ (where $M/u = \Omega$) is no index of term, i.e., $u \notin I_t(M)$ if and only if there exists $v \in \mathcal{O}(M)$, where $v < u$, and $\beta \in R_E$ such that $M[u \leftarrow x]/v \rightarrow_{K(E)}^* \beta_\Omega$. In other words, we will show that the term I-problem is reducible to the reachability problem for quasi-ground TRSs. We now define the quasi-ground system $K(E)$ as follows.

DEFINITION 5.5. For a set E of rewrite rules we define a new set $K(E)$ of rewrite rules as follows:

$$K(E) = \{\alpha \rightarrow \beta_\Omega \mid \alpha \rightarrow \beta \in E\} \cup \{\Omega \rightarrow N \mid N \in \text{sub}(\alpha_\Omega), \alpha \in L_E\}.$$

Let $\rightarrow_{K(E)}$ be a usual reduction, i.e., $M \rightarrow_{K(E)} N$ if and only if $M/u = \sigma(\alpha)$ and $N = M[u \leftarrow \sigma(\beta)]$ for some $u \in \mathcal{O}(M), \alpha \rightarrow \beta \in K(E)$, and $\sigma : X \rightarrow T_\Omega$. We have assumed that E is left-linear, so that $K(E)$ is a quasi-ground system. We now show a relationship between $\rightarrow_\omega$ and $\rightarrow_{K(E)}$.

LEMMA 5.5. *Let $M \rightarrow_\omega M'$, where $M, M' \in T_\Omega$. Then $M \rightarrow_{K(E)}^* M'$ holds.*

Proof. By $M \rightarrow_\omega M'$ there exist $v \in \mathcal{O}(M)$ and $\alpha \rightarrow \beta \in E$ such that

$$M/v \uparrow\!+ \alpha_\Omega \quad \text{and} \quad M' = M[v \leftarrow \beta_\Omega].$$

By $M/v \uparrow\!+ \alpha_\Omega$ let $Q = M/v \sqcup \alpha_\Omega$. Note that for each $u \in \mathcal{O}_\Omega(M/v)$ (see Definition 3.3) if $u \in \mathcal{O}(\alpha_\Omega)$, then $Q/u = \alpha_\Omega/u$ holds and $Q = M/v[u \leftarrow \alpha_\Omega/u, u \in \mathcal{O}_\Omega(M/v) \cap \mathcal{O}(\alpha_\Omega)]$ by definition of $\sqcup$. Thus Q is obtained from M/v by replacing Ω at each occurrence u in $\mathcal{O}_\Omega(M/v)$ by α_Ω/u, where $u \in \mathcal{O}(\alpha_\Omega)$. Hence we have $M/v \rightarrow_{K(E)}^* Q$ by applying rewrite rules of form $\Omega \rightarrow R$, where $R \in \text{sub}(\alpha_\Omega)$. Furthermore, $Q \rightarrow_{K(E)} \beta_\Omega$ holds by $Q \geq \alpha_\Omega$. Thus $M \rightarrow_{K(E)}^* M[v \leftarrow \beta_\Omega] = M'$ holds. □

LEMMA 5.6. *Let $M \rightarrow_{K(E)} M'$ and $N \leq M$, where $M, M', N \in T_\Omega$. Then there exists $N' \in T_\Omega$ such that $N \rightarrow_\omega^* N'$ and $N' \leq M'$.*

Proof. Let v be the redex occurrence of $M \rightarrow_{K(E)} M'$. If $M/v = \Omega$, then a rule of form $\Omega \rightarrow R$ is applied. In this case $M \leq M'$ holds, so that $N \leq M \leq M'$. Thus this lemma holds if we choose $N' = N$. So consider the case for which $M/v \neq \Omega$, that is, $M/v \geq \alpha_\Omega$ for some $\alpha \in L_E$ and $M' = M[v \leftarrow \beta_\Omega]$, where $\alpha \rightarrow \beta \in E$. For $N \leq M$ if there exists $v' \in \mathcal{O}(N)$ such that $N/v' = \Omega$ and $v' \leq v$, then, obviously, $N \leq M[v \leftarrow \beta_\Omega] = M'$ holds by $N \leq M$. So this lemma holds if we choose $N' = N$.

Consider the case for which there exists no $v' \in \mathcal{O}(N)$ such that $v' \leq v$ and $N/v' = \Omega$. In this case, by $N \leq M$ we have

$$(16) \qquad v \in \mathcal{O}(N) \wedge \text{Occ}(N, v) = \text{Occ}(M, v) \neq \Omega.$$

Since $M/v \geq \alpha_\Omega$ and $M/v \geq N/v$, we have $N/v \uparrow \alpha_\Omega$, so that $N/v \uparrow\!+ \alpha_\Omega$ by (16). Hence $N/v \rightarrow_\omega \beta_\Omega$ by the definition of $\rightarrow_\omega$. So $N \rightarrow_\omega N[v \leftarrow \beta_\Omega]$ holds. By $N[v \leftarrow \beta_\Omega] \leq M[v \leftarrow \beta_\Omega] = M'$ this lemma holds. □

LEMMA 5.7. *Let $M \in T_\Omega - \{\Omega\}$ and $u \in \mathcal{O}(M)$, where $M/u = \Omega$. Then $u \notin I_t(M)$ if and only if there exist $v \in \mathcal{O}(M)$, where $v < u$, and $\beta \in R_E$ such that*

$$(M[u \leftarrow x])/v \rightarrow_{K(E)}^* \beta_\Omega,$$

where x is a variable.

Proof. The *only-if* part obviously holds by Lemmas 5.4 and 5.5. So consider the *if* part. Let $M[u \leftarrow x]/v \to^*_{K(E)} \beta_\Omega$. By Lemma 5.6 we have the following ω-reduction sequence

$$\gamma : M[u \leftarrow x]/v \to^*_\omega N \quad \text{for some } N \leq \beta_\Omega.$$

Let u' satisfy $vu' = u$. Note that N does not contain x, since β_Ω does not contain x. So some $v_1 \in \mathcal{O}(M[u \leftarrow x]/v)$, where $v_1 < u'$, must be a redex occurrence in γ. Thus for such a v_1

$$M[u \leftarrow x]/vv_1 \to^*_\omega Q \to_\omega \bar{\beta}_\Omega \quad \text{and} \quad Q \uparrow\!+ \bar{\alpha}_\Omega$$

for some $Q \in T_\Omega$ and $\bar{\alpha} \to \bar{\beta} \in E$. Hence by Lemma 5.4 we have $u \notin I_t(M)$. □

By Lemma 5.7 the term I-problem is reducible to the reachability problem for quasi-ground systems, which has been shown to be decidable in [4], [15], [16]. Thus we have the following theorem.

THEOREM 5.1. *It is decidable, for an Ω-term M and an occurrence $u \in \mathcal{O}(M)$, whether u is an index of term in M, i.e., $u \in I_t(M)$.*

Next we discuss the time complexity of algorithms for deciding the term I-problem. By Lemma 5.7, to decide whether u is an index of term in M we need the reachability tests for at most $|M| \cdot \|R_E\|$ pairs $(M[u \leftarrow x]/v, \beta_\Omega)$. Let $\mathcal{A}$ be an algorithm that takes as input a quasi-ground system E_1 and two terms M, N and decide whether $M \to^*_{E_1} N$. For example, as $\mathcal{A}$ one can consider an algorithm in [4], [15], [16]. Anyway, it will be natural to consider the input size of $\mathcal{A}$ as size$(E_1) + |M| + |N|$, where size$(E_1) = \sum_{\alpha\to\beta\in E_1} |\alpha| + |\beta|$. Let $T_\mathcal{A}(n)$ be the time needed by algorithm $\mathcal{A}$ under inputs whose size is n. (We consider $T_\mathcal{A}(n)$ as the worst-case time complexity.) Let n = size(E), and let $m = |M|$. Then, for given $u, v \in \mathcal{O}(M)$ and $\beta \in R_E$, whether $M[u \leftarrow x]/v \to^*_{K(E)} \beta_\Omega$ can be checked in the order of time $T_\mathcal{A}(n^2 + m)$, since size$(K(E)) \leq n + n^2$. So the total time required to decide whether u is an index of term in M is bounded by the order of time $mn \cdot T_\mathcal{A}(n^2 + m)$. By the preceding arguments we have the following theorem.

THEOREM 5.2. *Let E be a left-linear system, and let $M \in T_\Omega$. Let n =* size(E), *and let $m = |M|$. Let $\mathcal{A}$ be an algorithm that solves the reachability problem for quasi-ground systems and operates in time $T_\mathcal{A}(k)$ for inputs of size k. Then, using algorithm $\mathcal{A}$, we can decide whether a given occurrence u is an index of term in M in time $mn \cdot T_\mathcal{A}(n^2 + m)$.*

Since reachability for quasi-ground systems can be checked in polynomial time (see [4], [16]), we have the following corollary.

COROLLARY 5.1. *The term I-problem can be decided in polynomial time.*

Note. At each step of the call-by-need reduction sequence, we must find a needed redex. Thus consecutive searches for finding needed redexes are made, and it is important to consider how to reduce the total cost of computing needed redexes, that is, how to use useful information obtained from the search of the previous step to efficiently find a needed redex at each step. In this paper we did not consider this problem, so it will be the next step following the work of this paper.

6. Decidability of NV-sequentiality.

In this section we show that whether a left-linear TRS E is NV-sequential is decidable. For this purpose we need some lemmas and definitions. The following three technical lemmas are used later. They state properties concerning indices of term.

LEMMA 6.1. *Let $M \in T_\Omega$, and let $u, v \in \mathcal{O}(M)$, where $u \mid v$. Then $v \in I_t(M[u \leftarrow \Omega]) \Rightarrow v \in I_t(M)$.*

Proof. By the definition of indices of term, $v \in I_t(M[u \leftarrow \Omega])$ if and only if (i) $M[u \leftarrow \Omega]/v = \Omega$ and (ii) $N \geq M[u \leftarrow \Omega]$ and term$(N) =$ TRUE imply $N/v \neq \Omega$. By $u \mid v$ and (i) we have $M/v = \Omega$. Note that $N \geq M$ implies that $N \geq M[u \leftarrow \Omega]$, so that by (ii) $v \in I_t(M)$ holds □

LEMMA 6.2. *Let $M \in T_\Omega$, and let $u \in \mathcal{O}(M)$. If $I_t(M) = \emptyset$, then either $I_t(M[u \leftarrow \Omega]) = \emptyset$ or $u \in I_t(M[u \leftarrow \Omega])$.*

Proof. Assume that $I_t(M) = \emptyset$ and $u \notin I_t(M[u \leftarrow \Omega])$. Then we can show that $I_t(M[u \leftarrow \Omega]) = \emptyset$. Let $v \in \mathcal{O}(M)$ such that $M/v = \Omega$ and $u \mid v$. By $I_t(M) = \emptyset, v \notin I_t(M)$ holds, so that $v \notin I_t(M[u \leftarrow \Omega])$ holds by Lemma 6.1. Thus for all $u' \in \mathcal{O}(M[u \leftarrow \Omega])$ such that $M[u \leftarrow \Omega]/u' = \Omega$ we have $u' \notin I_t(M[u \leftarrow \Omega])$, so that $I_t(M[u \leftarrow \Omega]) = \emptyset$ holds. □

LEMMA 6.3. *If $uv \in I_t(M)$, then $v \in I_t(M/u)$.*

Proof. To the contrary, we assume that $v \notin I_t(M/u)$. Then by Lemma 5.3 we have

$$\exists w < v \exists Q \in T_\Omega \exists \alpha \in L_E$$

such that

$$(M/u[v \leftarrow x])/w \to_\omega^* Q \uparrow\!+ \alpha_\Omega.$$

By $(M/u[v \leftarrow x])/w = (M[uv \leftarrow x])/uw$ and Lemma 5.3 we have $uv \notin I_t(M)$, a contradiction. □

Next we need the following definition.

DEFINITION 6.1. Let $h_E = \text{Max}\{h(\alpha) | \alpha \in L_E\}$, i.e., h_E is the maximal height of left-hand sides of E. We omit the subscript E where confusion does not occur without it.

The following lemma says that, for a given Ω-term M, if some subterm M/u has an index of term and some Ω-term N' satisfying $N' = M[v \leftarrow \Omega]$ for some $v > u$ also has an index of term, then the existence of an index of term in M is ensured. That is, this lemma shows a kind of transitivity result on the index of term and will be used to prove the main theorem in this section, i.e., to obtain an upper bound of the least size $|M|$ satisfying $I_t(M) = \emptyset$ if such M in Ω-normal form exists. (In [12] a result similar to this lemma is called a partial transitivity result for index propagation.)

LEMMA 6.4. *Let M be an Ω-term, and let $u\delta v \in \mathcal{O}_\Omega(M)$, where $|\delta| \geq h_E$. If $u\delta \in I_t(M[u\delta \leftarrow \Omega])$ and $\delta v \in I_t(M/u)$, then $u\delta v \in I_t(M)$.*

Proof. To the contrary, we assume that $u\delta v \notin I_t(M)$. Then by Lemma 5.3

$$\exists w < u\delta v \exists Q \in T_\Omega \exists \alpha \in L_E$$

such that

(17) $$M[u\delta v \leftarrow x]/w \to_\omega^* Q \uparrow\!+ \alpha_\Omega.$$

Note that $w < u$ must hold by $\delta v \in I_t(M/u)$ and Lemma 5.3 (see Fig. 1). Without loss of generality we can assume that for every occurrence w', where $w \leq w' < u, w'/w$ is safe for the ω-reduction sequence (17). Note that by $\delta v \in I_t(M/u)$, for every occurrence w'', where $u \leq w'' < u\delta v, w''/w$ is also safe for sequence (17). Hence for a term $M[u\delta \leftarrow x]$ we have

$$M[u\delta \leftarrow x]/w \to_\omega^* Q[u\delta/w \leftarrow x].$$

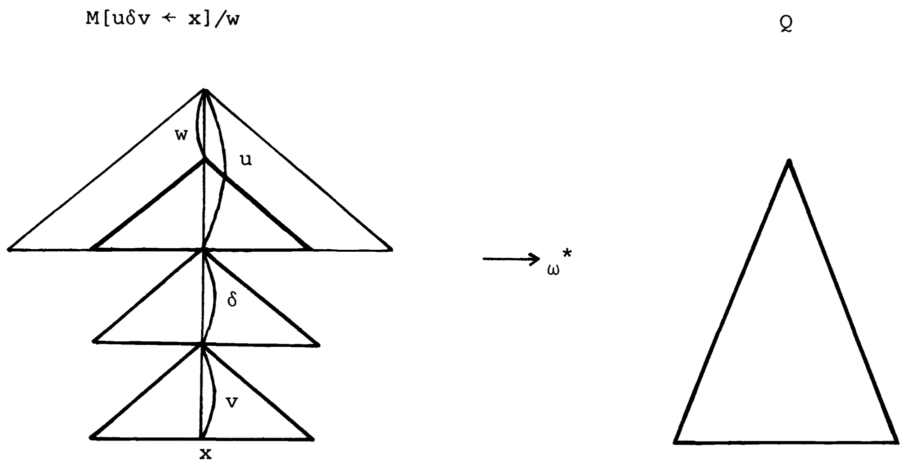


FIG. 1. *ω-reduction sequence* $M[u\delta v \leftarrow x]/w \to_\omega^* Q$.

Moreover, $Q[u\delta/w \leftarrow x] \uparrow\!\!+\ \alpha_\Omega$ holds, since $Q \uparrow\!\!+\ \alpha_\Omega$ by (17) and $|u\delta/w| > h_E$ holds. Hence $u\delta \notin I_t(M[u\delta \leftarrow \Omega])$ by Lemma 5.3, a contradiction. □

We also need the following functions, the use of which will be explained later.

DEFINITION 6.2. We define some functions on Ω-term as follows: For $M \in T_\Omega$ and $n \geq 0$

(i) $\mathcal{O}(M)_n = \{u \in \mathcal{O}(M) \mid |u| = n\}$, i.e., $\mathcal{O}(M)_n$ is the set of occurrences of M with size n.

(ii)

$$\mathrm{Pref}(M)_n = \begin{cases} M[u \leftarrow \Omega, u \in \mathcal{O}(M)_n] & \text{if } \mathcal{O}(M)_n \neq \emptyset, \\ M & \text{otherwise.} \end{cases}$$

We call $\mathrm{Pref}(M)_n$ the Ω-prefix of M with height $\leq n$. Note that $\mathrm{Pref}(M)_n \leq M$.

(iii) $\mathrm{Yield}(M)_h = \{\mathrm{Pref}(N)_h | M \to_\omega^* N\}$, where $h = h_E$ (see Definition 6.1), i.e., $\mathrm{Yield}(M)_h$ is the set of the Ω-prefixes (with height $\leq h$) of Ω-terms reachable from M under $\to_\omega$.

(iv) $\mathrm{Env}(M,u)_{2h} = \{\mathrm{Pref}(N)_{2h} \mid \exists\gamma : M[u \leftarrow \Omega] \to_\omega^* N$ such that $u \in \mathcal{O}(M)$ is safe for $\gamma\}$, where $h = h_E$, i.e., $\mathrm{Env}(M,u)_{2h}$ is the set of the Ω-prefixes (with height $\leq 2h$) of Ω-terms reachable from $M[u \leftarrow \Omega]$ by ω-reduction sequences for which occurrence $u \in \mathcal{O}(M)$ is safe. $\mathrm{Env}(M,u)_{2h}$ is undefined if $u \notin \mathcal{O}(M)$.

Example. Let $M = f(c, f(\Omega, x))$, where $f \in F_2, c \in F_0, x \in X$. Then $\mathcal{O}(M)_1 = \{1,2\}, \mathcal{O}(M)_2 = \{21,22\}$, and $\mathrm{Pref}(M)_1 = f(\Omega,\Omega)$, $\mathrm{Pref}(M)_2 = f(c, f(\Omega,\Omega))$.

Note. We can show that Yield and Env are computable functions, i.e., we can give algorithms to compute $\mathrm{Yield}(M)_h$ and $\mathrm{Env}(M,u)_h$ for a given Ω-term M and $u \in \mathcal{O}(M)$. The proof is omitted because Theorem 6.1 does not need the proof of existence of such algorithms, but simply upper bounds of $\|\mathrm{Yield}(M)_h\|$ and $\|\mathrm{Env}(M,u)_h\|$ (which are obvious).

Now we explain how to use the function Env. For this purpose we need the following definition.

DEFINITION 6.3. Two Ω-term M and N have the same environment concerning an occurrence δ (where $\delta \in \mathcal{O}_\Omega(M) \cap \mathcal{O}_\Omega(N)$) if $\mathrm{Env}(M,\delta)_{2h} = \mathrm{Env}(N,\delta)_{2h}$, where $h = h_E$.

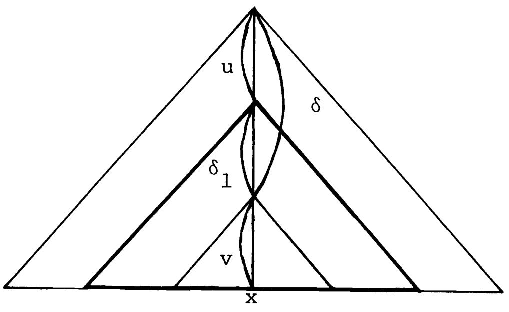


FIG. 2. $M[\delta \leftarrow Q[v \leftarrow x]]/u$.

The following lemma says that if M and N have the same environment concerning δ, then for any Ω-term Q if $M[\delta \leftarrow Q]$ has an index δv of term for some $v \in \mathcal{O}(Q)$, then $N[\delta \leftarrow Q]$ has also δv as an index of term. Thus replacement of $M[\delta \leftarrow Q]$ by $N[\delta \leftarrow Q]$ is index preserving in the above sense. This property will be used to prove the main theorem in this section, i.e., to obtain an upper bound of the least size $|M|$ satisfying $I_t(M) = \emptyset$ if such M in Ω-normal form exists.

LEMMA 6.5. *Let $M, N \in T_\Omega$ and $\delta \in \mathcal{O}_\Omega(M) \cap \mathcal{O}_\Omega(N)$, where $|\delta| = h$. Let M and N have the same environment concerning δ. Then for any $Q \in T_\Omega$ and $v \in \mathcal{O}_\Omega(Q)$*

$$\delta v \in I_t(M[\delta \leftarrow Q]) \textit{ if and only if } \delta v \in I_t(N[\delta \leftarrow Q]).$$

Proof: *if part*. To the contrary, we assume that $\delta v \notin I_t(M[\delta \leftarrow Q])$. By Lemma 5.3

$$\exists u < \delta v \exists P \in T_\Omega \exists \alpha \in L_E$$

such that

(18) $$M[\delta \leftarrow Q[v \leftarrow x]]/u \to_\omega^* P \uparrow + \alpha_\Omega.$$

If $u \geq \delta$, then $N[\delta \leftarrow Q[v \leftarrow x]]/u \to_\omega^* P \uparrow + \alpha_\Omega$, so that $\delta v \notin I_t(N[\delta \leftarrow Q])$ by Lemma 5.3, a contradiction.

So consider the case $u < \delta$. Let δ_1 satisfy $u\delta_1 = \delta$ (see Fig. 2). Without loss of generality we can assume that any $w < \delta_1$ is safe for the ω-reduction sequence (18), so that we can divide (18) into two subsequences, one of which is the subsequence with ω-redex occurrences $\geq \delta_1$ and the other of which has ω-redex occurrences disjoint from δ_1 :

(19) $$M[\delta \leftarrow Q[v \leftarrow x]]/u\delta_1 (= Q[v \leftarrow x]) \to_\omega^* P/\delta_1,$$

(20) $$M[\delta \leftarrow \Omega]/u \to_\omega^* P[\delta_1 \leftarrow \Omega].$$

Since M and N have the same environment concerning δ, by (20) we have

(21) $$N[\delta \leftarrow \Omega] \to_\omega^* P'$$

for some $P' \in T_\Omega$, where

$$\text{Pref}(P'/u)_h = \text{Pref}(P[\delta_1 \leftarrow \Omega])_h \tag{22}$$

and δ is safe for this ω-reduction sequence (21), so that $P'/\delta = \Omega$ holds. Thus by (21) we have

$$N[\delta \leftarrow \Omega]/u \to_\omega^* P'/u. \tag{23}$$

Now we combine the ω-reduction sequences (23) and (19):

$$N[\delta \leftarrow Q[v \leftarrow x]]/u \to_\omega^* (P'/u)[\delta_1 \leftarrow P/\delta_1]. \tag{24}$$

Note that $\text{Pref}((P'/u)[\delta_1 \leftarrow P/\delta_1])_h = \text{Pref}(P)_h$ holds by (22), so that

$$(P'/u)[\delta_1 \leftarrow P/\delta_1] \uparrow\!+ \alpha_\Omega$$

holds by (18). Therefore, $\delta v \notin I_t(N[\delta \leftarrow Q])$ by (24) and Lemma 5.3, which is a contradiction. □

Proof: *only-if part.* This proof is the same as the above except that M and N are interchanged. □

Now we give the last definitions needed in this section.

DEFINITION 6.4. For $n \geq 0$ we define $T_\Omega(n)$ and $\text{lg}(n)$ as follows: $T_\Omega(n) = \{M \in T_\Omega \mid h(M) \leq n\}$, i.e., $T_\Omega(n)$ is the set of Ω-terms of height $\leq n$. $\text{lg}(n) = \text{Max}\{|M| \mid M \in T_\Omega \wedge h(M) \leq n\}$, i.e., $\text{lg}(n)$ is the maximal size of Ω-terms of height $\leq n$.

DEFINITION 6.5. Let $l_0, l_1, \cdots, l_5$ be constants defined as follows: Let $l_0 = \text{Max}\{a(f) \mid f \in F\}$ (i.e., l_0 is the maximal arity of F), $l_1 = l_0^h, l_2 = \|2^{T_\Omega(2h)}\|$ (i.e., l_2 is the cardinality of subsets of $T_\Omega(2h)$), $l_3 = \|2^{T_\Omega(h)}\|, l_4 = \|T_\Omega(h)\|$, and $l_5 = \text{Max}\{|M| \mid M \in \text{NF}_E \wedge h(M) \leq h\}$, where $h = h_E$.

We are now ready to show the decidability of NV-sequentiality for left-linear TRSs. The following theorem says that to decide whether a left-linear TRS E is NV-sequential, we need to check, for only a finite number of Ω-normal forms, whether there exist indices of term. In other words, this theorem gives an upper bound of the least size $|M|$ satisfying $I_t(M) = \emptyset$ if such an Ω-term M in Ω-normal form exists.

THEOREM 6.1. *Let $L = 1 + h + l_5 + l_1 \cdot l_2 \cdot l_3 \cdot l_4$. Then a left-linear* TRS *E is* NV*-sequential if and only if for all Ω-normal form M in T_Ω such that $|M| \leq lg(L)$ and $M \notin T, I_t(M) \neq \emptyset$ holds.*

Proof: *only-if part.* If M is in Ω-normal form and $M \notin T$, then $\text{term}(M) = \text{FALSE}$, so that $I_t(M) \neq \emptyset$ by NV-sequentiality of E.

Proof: *if part.* To the contrary, we assume that there exists an Ω-normal form M such that $|M| > lg(L), M \notin T$, and $I_t(M) = \emptyset$. Here we assume that M is such an Ω-term with the least size $|M|$. By $|M| > lg(L)$ Definition 6.4 of $lg(L)$ ensures $h(M) > L$, so that $\exists v \in \mathcal{O}(M)$, such that $|v| > L$. Let

$$v = v_1 v_2, \quad \text{where } |v_2| = l_5 + 1. \tag{25}$$

Note that $|v_1| > h + l_1 \cdot l_2 \cdot l_3 \cdot l_4$.

Our goal is to show that there exists an Ω-normal form M' such that $|M'| < |M|$, $M' \notin T$, and $I_t(M') = \emptyset$. Thus this contradicts the minimality of the size of M. The

crucial point in this argument is how to construct such an Ω-term M'. We will show that $M' = M[u_0 \leftarrow M/u_1]$ for some $u_0, u_1 \in \mathcal{O}(M)$ satisfies the above required conditions:
(i) $M' \notin T$,
(ii) $|M'| < |M|$,
(iii) M' is in Ω-normal form,
(iv) $I_t(M') = \emptyset$.

The following assertion shows that if we choose u_1 so that $u_1 \leq v_1$ may hold for v_1 in (25), then $M/u_1 \notin T$, so that $M' = M[u_0 \leftarrow M/u_1]$ satisfies condition (i).

ASSERTION 6.1. *M/v_1 contains Ω.*

Proof. To the contrary, we assume that $M/v_1 \in T$. Note that M/v_1 is in normal form. By (25), $v_2 \in \mathcal{O}(M/v_1)$ and $|v_2| > l_5 = \mathrm{Max}\{|M| \mid M \in \mathrm{NF}_E \wedge h(M) \leq h_E\}$, so that we have $h(M/v_1) > h_E$. Let

$$N = (M/v_1)[u \leftarrow x, u \in \mathcal{O}(M/v_1)_{h_E}],$$

where $x \in X$. Then clearly $N \in \mathrm{NF}_E$ by $M/v_1 \in \mathrm{NF}_E$. Note that

$$(26) \qquad |N| < |M/v_1| \quad \text{and} \quad \mathrm{Pref}(N)_{h_E} = \mathrm{Pref}(M/v_1)_{h_E}.$$

Let

$$M_1 = M[v_1 \leftarrow N].$$

Then $|M_1| < |M|$ holds by (26). Note that $M_1 \notin T$ by $M \notin T$ and the assumption $M/v_1 \in T$, and M_1 is in Ω-normal form, since M is in Ω-normal form and replacement of M/v_1 by N does not produce any redex by (26). Moreover, we can show that $I_t(M_1) = \emptyset$, so that this contradicts the minimality of the size of M.

We now show that $I_t(M_1) = \emptyset$. By $I_t(M) = \emptyset$, for every $u \in \mathcal{O}_\Omega(M)$, we have $u \notin I_t(M)$, so that by Lemma 5.3

$$\exists u' < u \exists Q \in T_\Omega \exists \alpha \in L_E$$

such that

$$(27) \qquad M[u \leftarrow x]/u' \to_\omega^* Q \uparrow\!+ \alpha_\Omega.$$

Note that $u|v_1$ holds by $M/v_1 \in T$ and $M/u = \Omega$. There are two cases: (i) $u'|v_1$ and (ii) $u' < v_1$. In case (i) obviously $u \notin I_t(M_1)$ by Lemma 5.3, so consider case (ii). Let δ satisfy $u'\delta = v_1$. Without loss of generality we can assume that every occurrence δ' satisfying $\delta' < \delta$ and $u'\delta' < u$ is safe for the ω-reduction sequence (27) (see Fig. 3). We want to show that

$$(28) \qquad M_1[u \leftarrow x]/u' \quad (= M[u \leftarrow x, v_1 \leftarrow N]/u') \to_\omega^* Q' \uparrow\!+ \alpha_\Omega$$

for some $Q' \in T_\Omega$, so that $u \notin I_t(M_1)$. If any $\delta' < \delta$ is safe for the ω-reduction sequence (27), then $M/v_1 \to_\omega^* Q/\delta$ by the definition of safe, and since $M/v_1 \in \mathrm{NF}_E$ by the assumption, we have

$$M/v_1 = Q/\delta.$$

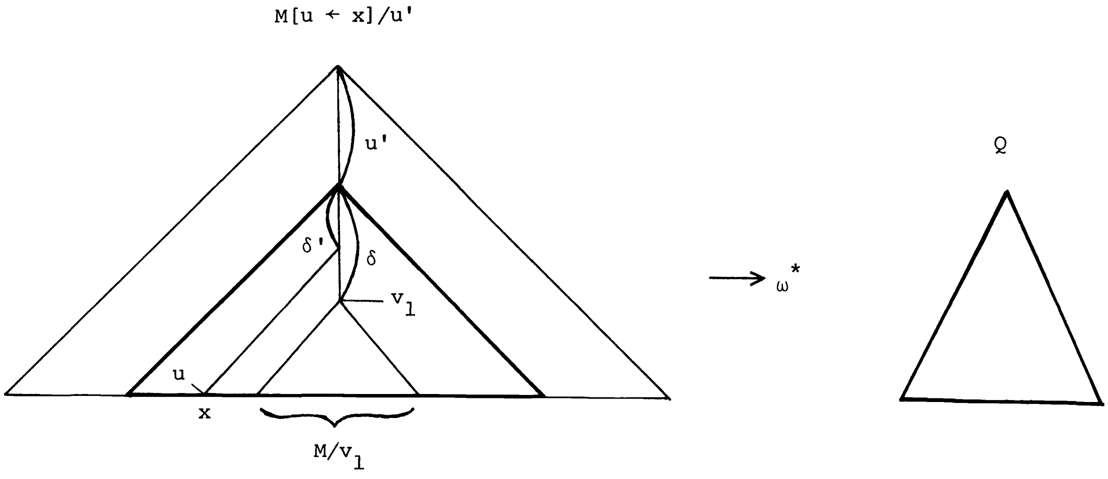


FIG. 3. *ω-reduction sequence* $M[u \leftarrow x]/u' \to_\omega^* Q$.

By replacing M/v_1 with N, we have the following sequence from (27):

$$M[u \leftarrow x, v_1 \leftarrow N]/u' \to_\omega^* Q[\delta \leftarrow N].$$

(Note that $v_1/u' = \delta$.) Then $Q[\delta \leftarrow N] \uparrow\!+\, a_\Omega$ holds, since $\mathrm{Pref}(N)_{h_E} = \mathrm{Pref}(M/v_1)_{h_E}$ by (26) and $Q \uparrow\!+\, a_\Omega$ by (27), where $Q/\delta = M/v_1$. Hence we have the above ω-reduction sequence (28), where $Q' = Q[\delta \leftarrow N]$. Otherwise, i.e., if some u'' satisfying that $u'' < \delta$ and $u'u''|u$ is an ω-redex occurrence in (27), then $\mathrm{Pref}(M/v_1)_{h_E} = \mathrm{Pref}(N)_{h_E}$ also ensures the existence of the above ω-reduction sequence (28). (In this case $Q' = Q$ holds.) Hence $u \notin I_t(M_1)$. Thus $I_t(M_1) = \emptyset$, as claimed. □

Now we explain how to choose occurrences u_0 and u_1 in $\mathcal{O}(M)$ to construct $M' = M[u_0 \leftarrow M/u_1]$. We choose u_1 satisfying $u_1 \le v_1$, so that $M' \notin T$ by Assertion 6.1.

For each occurrence u such that $u \le v_1$ and $|u| \le |v_1| - h$ we define tuple(u) as follows:

$$\mathrm{tuple}(u) = (\delta, \mathrm{env}, \mathrm{yield}, \mathrm{pref}) \in \mathbb{N}^h \times 2^{T_\Omega(2h)} \times 2^{T_\Omega(h)} \times T_\Omega(h),$$

where δ satisfies $|\delta| = h$ and $u\delta \le v_1$, env $= \mathrm{Env}(M/u, \delta)_{2h}$, yield $= \mathrm{Yield}(M/u\delta)_h$, pref $= \mathrm{Pref}(M/u\delta)_h$, and here $h = h_E$. Note that the number of different tuples is bounded by $l_1 \cdot l_2 \cdot l_3 \cdot l_4$ (see Definition 6.5).

By $|v_1| > h + l_1 \cdot l_2 \cdot l_3 \cdot l_4$ there exist

(29) $u, u' \in \mathcal{O}(M)$ such that $u < u' < v_1, |u'| \le |v_1| - h,$ and $\mathrm{tuple}(u) = \mathrm{tuple}(u')$.

Let

(30) $\mathrm{tuple}(u) = (\delta, \mathrm{env}, \mathrm{yield}, \mathrm{pref}) = \mathrm{tuple}(u'), u_0 = u\delta,$ and $u_1 = u'\delta$.

Note that $u'\delta \le v_1$, so that $M' = M[u_0 \leftarrow M/u_1]$ satisfies condition (i), $M' \notin T$, by Assertion 6.1. Our goal is to show the remaining three conditions.

Condition (ii), $|M'| < |M|$, holds, since by (29) $u < u'$, so that $|M/u_0| > |M/u_1|$ follows.

Condition (iii), that M' is in Ω-normal form, also holds. Since M is in Ω-normal form, M/u_1 is also in Ω-normal form. Moreover, since $\mathrm{Pref}(M/u_0)_{h_E} = \mathrm{Pref}(M/u_1)_{h_E} = \mathrm{pref}$ holds by (30), $M[u_0 \leftarrow M/u_1](= M')$ has no redex. Thus it remains to show condition (iv).

Proof of (iv). $I_t(M') = \emptyset$. We will show that any occurrence v in $\mathcal{O}_\Omega(M')$ is not an index of term so that $I_t(M') = \emptyset$.

Let $v \in \mathcal{O}_\Omega(M')$. Then there are two cases: (a) $v|u_0$ and (b) $u_0 < v$.

Case (a). $v|u_0$. In this case $v \in \mathcal{O}_\Omega(M)$ holds by $M' = M[u_0 \leftarrow M/u_1]$. Since $v \notin I_t(M)$, Lemma 5.3 ensures that

$$\exists w < v \exists Q \in T_\Omega \exists \alpha \in L_E \text{ such that } M[v \leftarrow x]/w \to_\omega^* Q \uparrow\!+ \alpha_\Omega. \tag{31}$$

There are two subcases: (a1) $w|u_0$ and (a2) $w < u_0$.

Case (a1). $w|u_0$. In this case $M[v \leftarrow x]/w$ is a subterm of $M'[v \leftarrow x]$, so that $v \notin I_t(M')$ by Lemma 5.3.

Case (a2). $w < u_0$. This proof is a slight generalization of that of Assertion 6.1. Let δ_0 satisfy $w\delta_0 = u_0$. Without loss of generality, we can assume that every occurrence δ' satisfying $\delta' < \delta_0$ and $w\delta' < v$ is safe for (31). We want to show that

$$M'[v \leftarrow x]/w \to_\omega^* Q' \uparrow\!+ \alpha_\Omega \tag{32}$$

for some $Q' \in T_\Omega$, so that $v \notin I_t(M')$. If any $\delta' < \delta_0$ is safe for (31), then

$$M/u_0 \to_\omega^* Q/\delta_0$$

holds. Since $\mathrm{Yield}(M/u_0)_{h_E} = \mathrm{Yield}(M/u_1)_{h_E} = \mathrm{yield}$ by (30), we have

$$M/u_1 \to_\omega^* P \quad \text{and} \quad \mathrm{Pref}(P)_{h_E} = \mathrm{Pref}(Q/\delta_0)_{h_E} \tag{33}$$

for some $P \in T_\Omega$. Hence by (33) and (31) we have

$$M[v \leftarrow x, u_0 \leftarrow M/u_1]/w \to_\omega^* Q[\delta_0 \leftarrow P].$$

(Note that $u_0/w = \delta_0$.) Then $Q[\delta_0 \leftarrow P] \uparrow + \alpha_\Omega$ holds, since $Q \uparrow + \alpha_\Omega$ by (31) and $\mathrm{Pref}(P)_{h_E} = \mathrm{Pref}(Q/\delta_0)_{h_E}$ by (33). Hence we have the sequence (32), where $Q' = Q[\delta_0 \leftarrow P]$. Otherwise, i.e., if some δ' satisfying $\delta' < \delta_0$ and $w\delta'|v$ is an ω-redex occurrence in (31), then $\mathrm{Yield}(M/u_0)_{h_E} = \mathrm{Yield}(M/u_1)_{h_E}$ (by (30)) also ensures the existence of the above ω-reduction sequence (32). (In this case $Q' = Q$ holds.) Hence $v \notin I_t(M')$, as claimed.

Case (b). $u_0 < v$. Let v' satisfy $v = u_0 v'$. Note that

$$v' \in \mathcal{O}_\Omega(M/u_1) \tag{34}$$

by $M'/u_0 = M/u_1$. Thus $u_1 v' \in \mathcal{O}_\Omega(M)$ holds. We can show the following assertion that $\delta v' \notin I_t(M/u')$. Note that $u_1 = u'\delta$ by (30). By using this result it will be shown that $v \notin I_t(M')$.

ASSERTION 6.2. $\delta v' \notin I_t(M/u')$.

Proof. To the contrary, we assume that $\delta v' \in I_t(M/u')$. Then we can show that

$$u'\delta \notin I_t(M[u'\delta \leftarrow \Omega]), \tag{35}$$

because if $u'\delta \in I_t(M[u'\delta \leftarrow \Omega])$ and $\delta v' \in I_t(M/u')$, then $I_t(M) \neq \emptyset$ would hold by Lemma 6.4, a contradiction. However, since Lemma 6.2 says that if $I_t(M) = \emptyset$, then either $I_t(M[u'\delta \leftarrow \Omega]) = \emptyset$ or $u'\delta \in I_t(M[u'\delta \leftarrow \Omega])$, and (35) ensures that

$$I_t(M[u'\delta \leftarrow \Omega]) = \emptyset. \tag{36}$$

Since $|M[u'\delta \leftarrow \Omega]| < |M|$ and $M[u'\delta \leftarrow \Omega]$ is in Ω-normal form, (36) contradicts the minimality of size of M. Thus Assertion 6.2 holds. □

Using Assertion 6.2 $\delta v' \notin I_t(M/u')$, we can show that

$$\delta v' \notin I_t(M'/u). \tag{37}$$

It follows that $u\delta v'(= v) \notin I_t(M')$ by Lemma 6.3, as claimed.

Thus it remains only to show (37). We first note that

$$\mathrm{Env}(M/u, \delta)_{2h} = \mathrm{Env}(M/u', \delta)_{2h} = \text{ env} \tag{38}$$

holds by (30). By Definition 6.3 (iv) of Env,

$$\mathrm{Env}(M/u, \delta)_{2h} = \mathrm{Env}(M/u[\delta \leftarrow \Omega], \delta)_{2h},$$

$$\mathrm{Env}(M/u', \delta)_{2h} = \mathrm{Env}(M/u'[\delta \leftarrow \Omega], \delta)_{2h}.$$

Thus $\mathrm{Env}(M/u[\delta \leftarrow \Omega], \delta)_{2h} = \mathrm{Env}(M/u'[\delta \leftarrow \Omega], \delta)_{2h}$ holds by (38), i.e., $M/u[\delta \leftarrow \Omega]$ and $M/u'[\delta \leftarrow \Omega]$ have the same environment concerning δ. Hence by Lemma 6.5 we have

$$\delta v'' \in I_t(M/u[\delta \leftarrow Q]) \quad \text{if and only if } \delta v'' \in I_t(M/u'[\delta \leftarrow Q]) \tag{39}$$

for any $Q \in T_\Omega$ and $v'' \in \mathcal{O}_\Omega(Q)$. Let $Q = M/u'\delta(= M/u_1)$, and let $v'' = v'$. Note that $v' \in \mathcal{O}_\Omega(M/u_1)$ by (34). Then (39) is

$$\delta v' \in I_t(M/u[\delta \leftarrow M/u_1]) \quad \text{if and only if } \delta v' \in I_t(M/u'[\delta \leftarrow M/u_1]). \tag{40}$$

Since $M/u'[\delta \leftarrow M/u_1] = M/u'$ and $\delta v' \notin I_t(M/u')$ by Assertion 6.2, the right-hand side of (40) is false, so that the left-hand side is false: $\delta v' \notin I_t(M/u[\delta \leftarrow M/u_1])$, i.e., $\delta v' \notin I_t(M'/u)$ by $M/u[\delta \leftarrow M/u_1] = M[u\delta \leftarrow M/u_1]/u = M'/u$. Thus (37) holds, as claimed. □

By the above arguments we have shown that any v in $\mathcal{O}_\Omega(M')$ is not an index of term, so that $I_t(M') = \emptyset$. □

Hence M' satisfies all the conditions (i)–(iv), so that we have a contradiction. Thus this theorem holds. □

Note. In the proof of Theorem 6.1 we have chosen u and u' satisfying tuple$(u) = (\delta$, env, yield, pref$) =$ tuple(u'). The sameness of the first, second, and third components (i.e., δ, env, and yield) was used in the proof of (iv), and that of the fourth component pref was used in the proof of (iii). Note that the use of the same δ's was necessary to use Lemma 6.5.

Note. Our concern was mainly to give a simplified proof of the decidability of NV-sequentiality. It remains open whether there exists an efficient algorithm to decide NV-sequentiality.

By Theorem 5.1 it is decidable whether $I_t(M) = \emptyset$ for Ω-normal form $M \in T_\Omega$. Thus by Theorem 6.1 we have the following theorem.

THEOREM 6.2. *It is decidable whether a left-linear* TRS E *is* NV-*sequential.*

Acknowledgments. The author is grateful to the referees for their critical reading of the earlier versions of this paper and their many subsequent helpful comments. He is also grateful to M. Broy, H. Hussman, P. Padawitz, and M. Wirsing for their helpful comments, and to N. Honda, T. Fukumura, and the members of the theoretical computer science group Jodankai in the Tokai area for their earnest discussions and advice.

REFERENCES

[1] H. P. BARENDREGT, *Functional programming and lambda calculus*, in Handbook of Theoretical Computer Science, Vol. B, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 321–363.

[2] ———, *The Lambda Calculus, Its Syntax and Semantics*, 2nd ed., North-Holland, Amsterdam, 1984.

[3] J. A. BERGSTRA AND J. W. KLOP, *Conditional rewrite rules: confluence and termination*, J. Comput. System Sci., 32 (1986), pp. 323–362.

[4] M. DAUCHET, S. TISON, T. HEUILLARD, AND P. LESCANNE, *Decidability of the confluence of finite ground term rewriting systems and of other related term rewriting systems*, Inform. and Comput., 88 (1990), pp. 187–201.

[5] C. M. HOFFMAN AND M. J. O'DONNELL, *An interpreter generator using tree pattern matching*, in Proc. 6th ACM Symposium on the Principles of Programming Languages, 1979, pp. 169–179.

[6] ———, *Implementation of an interpreter for abstract equations*, in Proc. 11th ACM Symposium on the Principles of Programming Languages, 1984, pp. 111–121.

[7] G. HUET, *Confluent reductions: abstract properties and applications to term rewriting systems*, J. Assoc. Comput. Mach., 27 (1980), pp. 797–821.

[8] G. HUET AND J.-J. LEVY, *Call by need computations in non-ambiguous linear term rewriting systems*, Rapport de Recherche No. 359, Institut National de Recherche en Informatique et en Automatique, Le Chesney, France, 1979; *Computations in orthogonal rewriting systems* I *and* II, in Computational Logic: Essays in Honor of Alan Robinson, J.-L. Lassez and G. Plotkin, eds., MIT Press, Cambridge, MA, 1991, pp. 395–443.

[9] G. HUET AND C. OPPEN, *Equations and rewrite rules: a survey*, in Formal Language Theory: Perspectives and Open Problems, R. V. Book, ed., Academic Press, New York, 1980, pp. 349–393.

[10] G. KAHN AND D. B. MACQUEEN, *Coroutines and networks of parallel processes*, in International Federation of Information Processing Societies 77, B. Gilichrist, ed., North-Holland, Amsterdam, 1977, pp. 993–998.

[11] G. KAHN AND G. PLOTKIN, *Domaines concrets*, Rapport Laboria No. 336, Institut de Recherche en Informatique et en Automatique, Le Chesney, France, 1978.

[12] J. W. KLOP AND A. MIDDELDORP, *Sequentiality in orthogonal term rewriting systems*, J. Symbolic Comput., 12 (1991), pp. 161–195.

[13] D. KNUTH AND P. BENDIX, *Simple word problems in universal algebras*, in Computational Problems in Abstract Algebra, J. Leech, ed., Pergamon Press, Elmsford, NY, 1970, pp. 263–297.

[14] M. J. O'DONNELL, *Computing in systems described by equations*, Lecture Notes in Computer Science 58, Springer-Verlag, Berlin, New York, 1977.

[15] M. OYAMAGUCHI, *The reachability problem for quasi-ground term rewriting systems*, J. Inform. Process., 9 (1986), pp. 232–236.

[16] ———, *On the word problem for right-ground term-rewriting systems*, Trans. IEICE Japan E, 73 (1990), pp. 718–723.

[17] ———, *Sufficient sequentiality: a decidable condition for call-by-need computations in term rewriting systems*, Tech. Report, Mie University, Tsu-shi, Japan, 1986.

[18] J. E. STOY, *Denotational Semantics, the Scott–Strachey Approach to Programming Languages*, MIT Press, Cambridge, MA, 1977.

[19] Y. SUGIYAMA, I. SUZUKI, K. TANIGUCHI, AND T. KASAMI, *Efficient execution in a class of term rewriting systems*, Trans. IECE Japan, J65-D (1982), pp. 858–865.

[20] S. THATTE, *A refinement of strong sequentiality for term rewriting with constructors*, Inform. and Comput., 72 (1987), pp. 46–65.

[21] D. A. TURNER, *A new implementation technique for applicative languages*, Software Practice and Experience, 9 (1979), pp. 31–49.

[22] J. VUILLEMIN, *Correct and optimal implementation of recursion in a simple programming language*, J. Comput. System Sci. 9 (1974), pp. 332–354.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 136–146, February 1993

ASPACE(o(log log n)) IS REGULAR*

KAZUO IWAMA†

Abstract. One of the common results of resource bounded Turing machines is the $\log\log n$ lower bound for the space usage of deterministic and nondeterministic Turing machines that accept nonregular languages. In this paper this result is extended to alternating Turing machines: It is proved that if $f(n) = o(\log\log n)$, then $f(n)$-space-bounded (off-line) alternating Turing machines can accept only regular sets. The problem has been open for a decade.

Key words. space lower bounds, alternating Turing machines

AMS(MOS) subject classifications. 68Q05, 68Q15, 68Q68

1. Introduction. One of the common results regarding resource-bounded Turing machines (TMs) [5] is that there is a nontrivial, tight lower bound, $\log\log n$, for the space usage of both deterministic TMs (DTMs) and nondeterministic TMs (NTMs) that accept nonregular languages. In this paper we extend this result to alternating TMs (ATMs); we prove that if $f(n) = o(\log\log n)$, then $f(n)$-space-bounded (off-line) ATMs can accept only regular sets. The problem was first considered by Sudborough in [11], and the $\log\log n$ lower bound was proved under the *strong definition* of space bounds (see below). In [1] the $\log\log\log n$ lower bound is proved under the *standard definition*, and the $\log\log n$ bound for *on-line* ATMs is also proved. The $\log\log n$ bound for *off-line* ATMs was once claimed to be proved [3], but the claim was later corrected [4]. Thus the problem was still open.

There are two distinct definitions of space complexity measurement for nondeterministic (and alternating) space-bounded TMs: (i) All computations must satisfy the space bound (*strong definition*). (ii) It is enough that at least one accepting computation on each acceptable string satisfies the bound (*standard definition*). For small space bounds, such as $\log\log n$, the distinction is important because we cannot use the common technique of marking off the work tape in advance and then finishing the computation as rejection if the head goes outside the bound. Definition (ii) is more reasonable according to the principle of nondeterminism (existence of a single good computation is enough) and is clearly more standard (used in most papers, e.g., [2], [7], [9]).

It should be noted that there are a number of facts that might mislead us into feeling that the lower bound that we are currently interested in can differ between NTMs and ATMs:

(1) There exists a (regular) language for which m states by deterministic finite automata (DFAs), $\log m$ states by nondeterministic finite automata (NFAs) and $\log\log m$ states by alternating finite automata (AFAs) are necessary and sufficient [2]. There also exists a (nonregular) language for which $\log n$ space by DTMs and $\log\log n$ space (standard definition) by NTMs are necessary and sufficient (see §2). Can the latter statement be extended to encompass $\log\log\log n$ space by ATMs by the analogy of finite automata?

(2) Not surprisingly, $\log\log n$ space-bounded (standard definition) ATMs are more powerful than the same space-bounded NTMs; this is demonstrated by using a rather simple language in the Appendix.

(3) Using a language similar to that of fact (2), we can also show that $\log\log n$ space-bounded (standard definition) off-line ATMs are more powerful than their on-line coun-

*Received by the editors August 14, 1991; accepted for publication (in revised form) January 8, 1992.

†Department of Computer Science and Communication Engineering, Kyushu University, Hakozaki, Fukuoka 812, Japan (iwama@csce.kyushu-u.ac.jp).

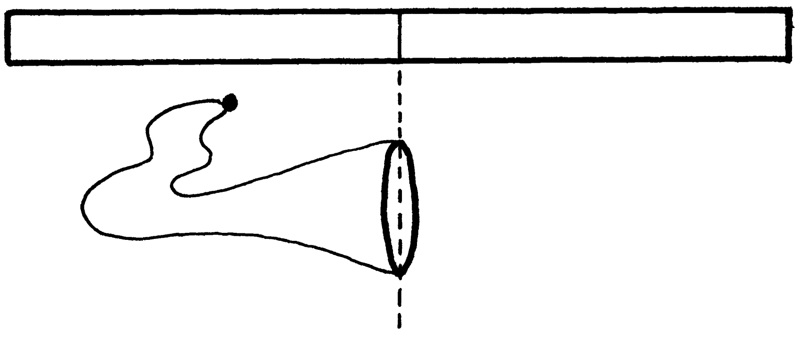


FIG. 1. *Computation subtrees.*

terparts (see Appendix). Thus we cannot rule out the possible reduction of the known $\log\log n$ lower bound for on-line ATMs.

(4) There does exist a complexity measure whose lower bound for NTMs is different from that for ATMs. In [6] it is shown that the (tight) lower bound of the reverse (or cross) complexity of single-read–write-tape TMs is $\log\log n$ for NTMs but $\log\log\log n$ for ATMs of even a single alternation and becomes as small as $\log^* n$ for general ATMs.

In this paper TMs are those machines having a read-only input tape and a read–write work tape. They are said to be *on-line* if the input head can move only from left to right (and can stay there) and *off-line* if it can move in both directions. We will call a triple (w, h, s) of the nonblank portion w of the work tape, the work-tape head position h, and the state s of the finite control a *global state*. A *configuration* consists of the input string, the input head position, and the global state. A *computation* (*tree*) P for an input x is a tree of the following structure: (i) The root is associated with the initial configuration denoted always by I_{init}. (ii) If a vertex v is associated with an existential configuration σ (i.e., its state is existential), then v has exactly one child that is one of the possible successors of σ. (iii) If a vertex v is a universal configuration, then v has the children that are the set of all possible successors of σ. The computation P is said to be *k-space bounded* if at any vertex of P at most k work-tape cells are used. P is said to be *accepting* if it is finite and all of its leaves are associated with accepting configurations. (We assume that TMs never write the blank symbol on the work tape. Then, since the length of the nonblank portion increases monotonically, "at any vertex of P" in the above definition can be replaced by "at any leaf of P" if P is an accepting computation.) An ATM M is called *$f(n)$-space bounded* if for any input string of length n that is accepted by M there is an $f(n)$-space-bounded accepting computation. Besides the (complete) computation tree described above, we often consider a *computation subtree*; its root is associated with an arbitrary (not necessarily initial) configuration, and all of its paths end (are cut forcibly) by some condition, e.g., when the input head crosses some boundary of the tape (see Fig. 1). All TMs described in this paper accept their inputs when the input heads are at the right end.

2. Basic tools. This section demonstrates two fundamental techniques that are important in proving the main theorem. Before that proof, we first take a look at how the strong and standard definitions are different in our current situation.

PROPOSITION 1 [11]. *If an* ATM *uses an unbounded number of work-tape cells under the strong definition of space usage, then it must use at least* $c\log\log n$ *cells infinitely often for some* $c > 0$.

Proof. Let M be an ATM. We can construct the NTM M' from M by replacing each universal state of M by an existential state with the same set of choices. Then, if M has a computation path Q (not the whole tree defined previously but a single path from the root to some leaf) that uses k work-tape cells, then M' has (the same) computation path Q that uses k work-tape cells for the same input. This concludes the proof since the statement is known to be true for NTMs [10]. □

Now let us return to the standard definition. We can no longer claim this proposition. First of all, we do not have to count the space usage if the above computation path Q is not accepting. Even if it is accepting (i.e., it ends with an accepting configuration), any computation *tree* of M including Q may not be accepting. In other words, "the same input" above, which must be accepted by the NTM M', is not necessarily accepted by the ATM M, and we again do not have to consider such an input when measuring the space usage of M. Furthermore, if there is an accepting computation tree of M including Q, we still cannot rule out the existence of a better computation tree for the same input.

We shall next review the convenient tool to prove small resource bounds, generally called *cut-and-paste*, by using the following well-known bounds for on-line TMs.

PROPOSITION 2. *Let* $f_1(n) = o(\log n)$, *i.e.*, $\sup(f_1(n)/\log n) = 0$, *and let* $f_2(n) = o(\log\log n)$. *Then any* $f_1(n)$*-space-bounded on-line* DTM *or* $f_2(n)$*-space-bounded on-line* NTM *can accept only a regular set.*

Proof. We shall show that if a $g_1(n)$-space-bounded DTM M accepts a nonregular language, then $g_1(n) \geq c \log n$ for an infinite number of n's and for some fixed constant c determined by M. Clearly, $g_1(n)$ is not bounded by any constant. Then one can see that there exists an arbitrary large integer k for which there is a string v such that (i) v is accepted by M, and (ii) for all strings u that are accepted by M and are shorter than v, there is a computation that uses less than k space. Intuitively, v is the shortest string that needs k space.

Recall that we are now discussing *on-line* TMs. For strings x and y let $t_{x;y}$ denote M's global state (w, h, s) that M, on the input xy, is in when its input head crosses (only once) the boundary between x and y (";" indicates the boundary). Now select an arbitrary pair of v and k mentioned above. If v is written as $v = xyz$ $(y \neq \varepsilon)$, then $t_{x;yz}$ and $t_{xy;z}$ must be different. (The reasoning is as follows: Suppose that $t_{x;yz} = t_{xy;z}$. Then one can see that (i) xz is also accepted and (ii) M also needs k space to accept xz since both xyz and xz lead M to the same accepting configuration, which determines the space usage. That contradicts the assumption that xyz is the shortest string that needs k space.) Note that if the computation is k-space bounded, $|w| \leq k$ at any boundary. Thus the number of all different triples (w, h, s) is bounded by c_1^k for a constant c_1. In order that $t_{x;yz} \neq t_{xy;z}$ for any two boundaries, $c_1^k \geq |xyz| = n$ or $k \geq c_2 \log n$ for a constant c_2.

The same argument may also be applied when M is an NTM by replacing $t_{x;y}$ with $T_{x;y}$ that is the set of all global states $t = (w, h, s)$ such that M can reach t at the boundary. Now suppose that $T_{x;yz} = T_{xy;z}$. Then we can claim that

(*) if xyz is accepted with k, but not less space, then
xz is also accepted with k, but not less, space.

(Note that (*) is equivalent to the statement that xyz is accepted with k space if and only if xz is accepted with k space. Again, recall that the space usage of a computation path is determined by that of its final configuration. If $t_{\text{init}} \xrightarrow{x} t_1 \xrightarrow{y} t_2 \xrightarrow{z} t_3$ is an accepting computation path with k space, then, since t_2 that appears in $T_{xy;z}$ on the above path is also in $T_{x;yz}$, $t_{\text{init}} \xrightarrow{x} t_2 \xrightarrow{z} t_3$ is also such a path. Note that t_1 may differ from t_2.

Conversely, suppose that $t_{\text{init}} \xrightarrow{x} t_5 \xrightarrow{z} t_6$ is a path with k' space, which means that t_5 is in $T_{x;z}$. Clearly, $T_{x;z}$ coincides with $T_{x;yz}$, and, since $T_{x;yz} = T_{xy;z}$ by the assumption, there must be t_4 that constructs the path $t_{\text{init}} \xrightarrow{x} t_4 \xrightarrow{y} t_5 \xrightarrow{z} t_6$, which clearly uses the same space. That is enough to claim the statement (*).) A crucial point is that we can get the same conclusion if the assumption $T_{x;yz} = T_{xy;z}$ is relaxed to $T^k_{x;yz} = T^k_{xy;z}$, where $T^k_{x;y}$ is the subset of $T_{x;y}$ such that $|w| \leq k$. Thus we can conclude that $T^k_{x;yz}$ should differ from $T^k_{xy;z}$ for any $y \neq \epsilon$. Now a simple evaluation of the size of the power set is enough to achieve the goal. □

It turns out that these lower bounds are tight: Let $bin(i)$ denote the string over 0 and 1 whose reverse is the binary representation of the integer i. For example, $bin(6) = 011$. Now let

$$BIN = \{\sharp bin(0)\sharp bin(1)\sharp \cdots \sharp bin(n)\sharp \mid n \geq 0\},$$

and let $\overline{BIN} = \{0,1,\sharp\}^* - BIN$. Then it is not difficult to see [5] that $\overline{BIN}$ is recognized by a $(\log n)$-space-bounded ($(\log\log n)$-space-bounded, respectively) on-line DTM (NTM, respectively).

Now we must determine how we can apply the cut-and-paste method to ATMs. If ATMs are *on-line*, we have the following answer.

PROPOSITION 3 [1]. *Let $f(n) = o(\log\log n)$. Then any $f(n)$-space-bounded on-line* ATM *M can recognize a regular set.*

Proof. We could simply extend the above proof for NTMs by replacing a computation path with a set of computation paths. The idea is that at each boundary the machine M is universally in a set S_1 of global states (corresponding to one computation tree) or is universally in set S_2 (another computation tree), and so on. Thus this extension would force us to enumerate sets of sets of global states, or we would obtain only a $\log\log\log n$ lower bound.

The following proof is considerably modified from [1] in order to prepare ourselves for the main theorem. It should be noted that $T_{x;y}$, introduced in Proposition 2, depends on x but not at all on y. To attain the same goal as before, we can develop a similar construct that does not depend on x but does depend on y. Namely, we define $S_{x;y}$ as the set of global states t, at the boundary between x and y, that is led to acceptance by y. Then it is not difficult to see that $S_{x;yz} = S_{xy;z}$ implies that xyz is accepted if and only if xz is also. A difficulty is that the claim (*) on the space usage in the proof of Proposition 2 no longer holds for the following reason: Suppose that there is an accepting computation path $t_{\text{init}} \xrightarrow{x} t_1 \xrightarrow{y} t_2 \xrightarrow{z} t_3$. Since $S_{x;yz} = S_{xy;z}$, there must be t_2 in $S_{x;yz}$. However, there may not be a computation (sub)path $t_{\text{init}} \xrightarrow{x} t_2$. (Note that $S_{x;yz}$ requires only that the machine can go to an accepting configuration *if* we assume that it is in some global state in $S_{x;yz}$ at the boundary between x and yz.) There should exist the accepting path $t_{\text{init}} \xrightarrow{x} t_1 \xrightarrow{z} t_4$, but it may need less space than before.

With this problem in mind, we define $S^k_{x;y}$ as follows: Let (t,j) be a pair of a global state and an integer. Then $S^k_{x;y}$ contains all (t,j)'s such that (i) $j \leq k$ and (ii) if M is in t at the boundary, then t leads to acceptance while using j space but does not do so while using $j-1$ space or less (i.e., there is an accepting computation subtree of the described space bound whose root is associated with the global state t and the input head being on the leftmost symbol of y). This introduction of *pairs* of a global state and space, unlike the conventional *only global states*, is important, especially for proving the main theorem.

Let P be a computation subtree of M on input xy such that its root is I_{init} and all paths end when the input head crosses the boundary between x and y. We also define

$SEC_{x;y}(P)$ as the set of all global states that M, following P, is in when it crosses the boundary. A key fact (whose proof is easy and is omitted) is that there is an accepting computation tree of k-space bound for M on input xy if and only if there is this kind of computation subtree P such that

$$SEC_{x;y}(P) \subseteq \{t \mid (t,j) \in S^k_{x;y}\}.$$

Now suppose that the ATM M accepts input xyz and uses k, but not less than k, space and that $S^k_{x;yz} = S^k_{xy;z}$. Then we can conclude by the following reasoning that (i) xz is also accepted while using k space and (ii) that it is not accepted while using less than k space: The assumption means that there is an accepting k-space-bounded computation tree for xyz. By this and the preceding fact there is a computation subtree P_0 whose root is I_{init} and all of whose paths end at the boundary between x and yz such that

$$SEC_{x;yz}(P_0) \subseteq \{t \mid (t,j) \in S^k_{x;yz}\}.$$

Note that $S^k_{x;yz} = S^k_{xy;z}$ means $S^k_{x;yz} = S^k_{x;z}$ (because both $S^k_{xy;z}$ and $S^k_{x;z}$ depend on only z), and therefore the computation subtree P_0 also satisfies

$$SEC_{x;z}(P_0) \subseteq \{t \mid (t,j) \in S^k_{x;z}\}.$$

Again by the fact ("if" part), there is an accepting computation tree of k-space bound for xz. Thus statement (i) is correct. As for (ii), we can similarly show that if xz is accepted and uses less than k space, then xyz is also.

We are now ready to carry out the same enumeration as that of Proposition 2. The number of all different (t,j)'s such that $t = (w,h,s)$ and $|w| \leq j \leq k$ is a bit larger than before, but it is still bounded by c^k for some constant c. □

3. Main result. It should be noted that the weaker $\log\log\log n$ lower bound proved in [1] can also be proved by modifying the transition matrix in [5] from $r \times r$ to $2^r \times 2^r$.

THEOREM 1. *Let $f(n) = o(\log\log n)$. Then any $f(n)$ space-bounded off-line* ATM *M can recognize a regular set.*

Proof. With the boundary between strings x and y we associate $S^k_{x;y}$ as before. This time, according to the alternation from on-line to off-line, $S^k_{x;y}$ is not a simple set but is an array such as

$$S^k_{x;y} = (S^k_{x;y}(1), S^k_{x;y}(2), \cdots, S^k_{x;y}(p)),$$

where the value p will be given later. We need a few more notations: Let P be a computation subtree whose root is associated with a global state t and an input head position τ. In what follows we often focus our attention to some boundary, say, the boundary between x and y of the input xy. If τ is on the rightmost symbol of x (leftmost symbol of y, respectively), then we say that P has *root* $(t,x;y,L)$, where L stands for the left of the boundary (respectively, $(t,x;y,R)$, where R stands for the right of the boundary). $CRS_{x;y}(P)$ denotes how many times M's input head, following P, crosses the boundary. (The number of crosses can differ according to each path of P. We take the largest value as $CRS_{x;y}(P)$.) $SP(P)$ denotes the space P uses.

Now we define $S^k_{x;y}(i)$. Suppose that i is odd. Then $S^k_{x;y}(i)$ contains all the pairs (t,j) of a global state t and an integer $j \leq k$ satisfying the following conditions: (i) There is an accepting computation subtree P (see Fig. 2) such that P has root $(t,x;y,R)$, $CRS_{x;y}(P) = i-1$, and $SP(P) = j$. (ii) There is no accepting computation subtree P such that P has the same root $(t,x;y,R)$ and the same $CRS_{x;y}(P) = i-1$ but for which

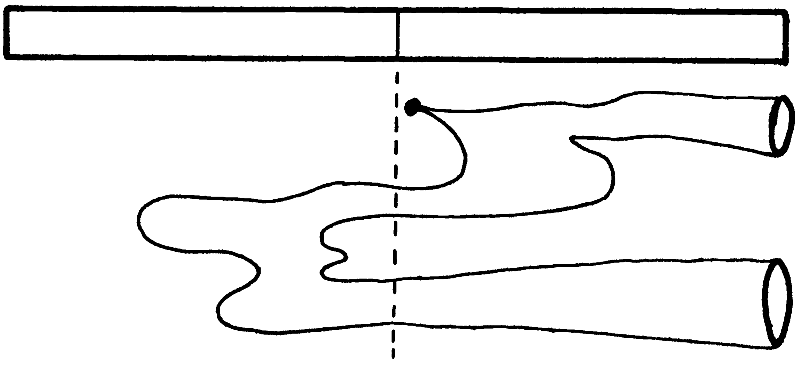


FIG. 2. *Accepting computation subtrees P to construct $S^k_{x;y}(3)$.*

$SP(P) < j$. (iii) If there is a pair (t, j') in $S^k_{x;y}(i')$ for some odd $i' < i$, then $j' > j$. If i is even, then P has root $(t, x; y, L)$ instead of $(t, x; y, R)$ and "odd" above should be replaced by "even." Intuitively speaking, the fact that (t, j) exists in $S^k_{x;y}(i)$ means that the global state t can lead to acceptance while using j space and $i - 1$ crosses *most economically*: If t can lead to acceptance while using j space and $i - 3$ (or fewer) crosses or while using $j - 1$ (or less) space and i crosses, then (t, j) does not appear in $S^k_{x;y}(i)$. Let us observe more about this. Suppose, for example, that $(a, 8)$, $(b, 3)$, and $(c, 5)$ are in $S^k_{x;y}(3)$ for global states a, b, and c. Then we know that M needs 8 space to go from global state a to acceptance. However, we cannot rule out the possibility that M needs, for example, only 5 space to do the same if we allow M to cross the boundary more: There may be a computation subtree Q such that it has root $(a, x; y, R)$ and all paths end when M crosses the boundary not from right to left (the first cross) but from left to right (the second cross) and such that $SEC_{x;y}(Q) = \{b, c\}$. If that is the case, then $(a, 5)$ should be in $S^k_{x;y}(5)$.

Thus $S^k_{x;y}$ seems to depend on both sides of the boundary, both x and y of the input xy. However, this is not desirable for attempting to apply the cut-and-paste. One can see that the proof of Proposition 3 fully depends on the fact that $S^k_{x;y}$ is determined by only the right side of the boundary. It is for this reason that the next claim is important.

CLAIM 1. *If i is odd, then $S^k_{x;y}(i)$ is determined by the right-side string y of the boundary and by $S^k_{x;y}(2), S^k_{x;y}(4), \cdots, S^k_{x;y}(i-1)$. If i is even, then $S^k_{x;y}(i)$ is determined by the left-side string x and by $S^k_{x;y}(1), S^k_{x;y}(3), \cdots, S^k_{x;y}(i-1)$.*

Proof. By the definition, $S^k_{x;y}(1)$ contains all (t_1, j_1) such that the global state t_1, which M is in when crossing the boundary from left to right, leads to acceptance while using j_1 (not less) space and never coming back to the boundary. (Note that it does not matter whether M can actually reach the boundary in t_1 from I_{init}.) Thus $S^k_{x;y}(1)$ clearly depends only on y.

Next it is shown that $S^k_{x;y}(2)$ can be constructed from x and $S^k_{x;y}(1)$. For each global state $t_2 = (w, h, s)$ ($|w| \leq k$) we construct, if there are any, all computation subtrees P_2 such that (i) P_2 has root $(t_2, x; y, L)$ and every path ends when it crosses the boundary from left to right for the first time (see Fig. 3) and (ii) $SEC_{x;y}(P_2) \subseteq \{t \mid (t, j) \in S^k_{x;y}(1)\}$, where $SEC_{x;y}(P_2)$ denotes, as before, the set of all global states M is in when it crosses the boundary. For such a subtree as P_2 we also define $SP(P_2)$ as

$$SP(P_2) = \max\{j \mid \exists t \text{ s.t. } t \in SEC_{x;y}(P_2) \text{ and } (t, j) \in S^k_{x;y}(1)\}.$$

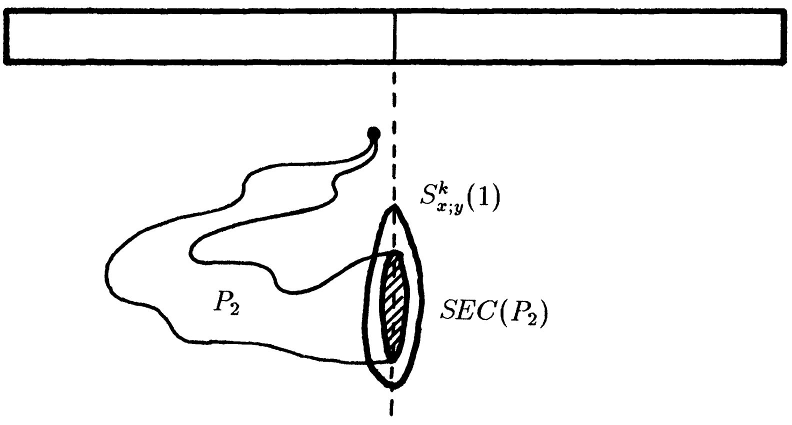


FIG. 3. *Computation subtree P_2.*

Note that there are, in general, many such P_2's for a single t_2. We select from among them one such that $SP(P_2)$ is minimum ($= j_2$) and (t_2, j_2) is included in $S^k_{x;y}(2)$. Note that we used only x and $S^k_{x;y}(1)$ for this construction.

Now suppose that we have constructed $S^k_{x;y}(1)$ through $S^k_{x;y}(i)$ for an even i. To obtain $S^k_{x;y}(i+1)$, we first let

$$\begin{aligned} MINS^k_{x;y}(i) = \{(t,j) \mid\ & (t,j) \in S^k_{x;y}(2) \cup S^k_{x;y}(4) \cup \cdots \cup S^k_{x;y}(i) \text{ and} \\ & (t,j') \notin S^k_{x;y}(2) \cup S^k_{x;y}(4) \cup \cdots \cup S^k_{x;y}(i) \text{ if } j' < j\}. \end{aligned}$$

Thus the fact that (t, j) is in $MINS^k_{x;y}(i)$ means that the global state t at the left side of the boundary (on the rightmost symbol of x) can lead to acceptance with j (but not less) space if it can cross the boundary $i-1$ *or fewer* times. Now, as before, for each global state t_{i+1} we construct all computation subtrees P_{i+1} such that (i) P_{i+1} has root $(t_{i+1}, x; y, R)$ and every path ends in acceptance directly (without coming back to the boundary) or ends when it crosses the boundary from right to left for the first time (Fig. 4) and (ii) $SEC_{x;y}(P_{i+1}) \subseteq \{t \mid (t,j) \in MINS^k_{x;y}(i)\}$. (It should be noted that $SEC_{x;y}(P_{i+1})$ includes several global states, each of which may need different crosses before ending with an accepting configuration. That is why we introduced $MINS^k_{x;y}(i)$.) $SP(P_{i+1})$ is defined as

$$\begin{aligned} SP(P_{i+1}) \;=\; & \max\{j \mid \text{(i) } \exists t \text{ s.t. } t \in SEC_{x;y}(P_{i+1}) \text{ and } (t,j) \in MINS^k_{x;y}(i) \\ & \text{or (ii) } \exists \text{ path from the root directly to acceptance while using } j \text{ space}\}. \end{aligned}$$

Among all the subtrees thus constructed, we select a subtree P_{i+1} such that $SP(P_{i+1})$ is minimum ($= j_{i+1}$). Thus obtained, (t_{i+1}, j_{i+1}) is not included in $S^k_{x;y}(i+1)$ unconditionally, however. It is included in $S^k_{x;y}(i+1)$ if (t,j), such that $t = t_{i+1}$ and $j \le j_{i+1}$, is not in $MINS^k_{x;y}(i-1)$. $MINS^k_{x;y}(i-1)$ is defined in the same way as $MINS^k_{x;y}(i)$, but $S^k_{x;y}(2) \cup S^k_{x;y}(4) \cup \cdots \cup S^k_{x;y}(i)$ is replaced by $S^k_{x;y}(1) \cup S^k_{x;y}(3) \cup \cdots \cup S^k_{x;y}(i-1)$. (Note that we do not violate the dependency of the construction. $S^k_{x;y}(i-1)$, for example, was determined by y and $S^k_{x;y}(2), \cdots, S^k_{x;y}(i-2)$.) Intuitively speaking, (t_{i+1}, j_{i+1}) is added to $S^k_{x;y}(i+1)$ if either (i) t_{i+1} absolutely needs at least i crosses before reaching accepting configurations regardless of its space usage or (ii) although it is already in, say, $S^k_{x;y}(i-3)$, we can save space if we allow it more crosses.

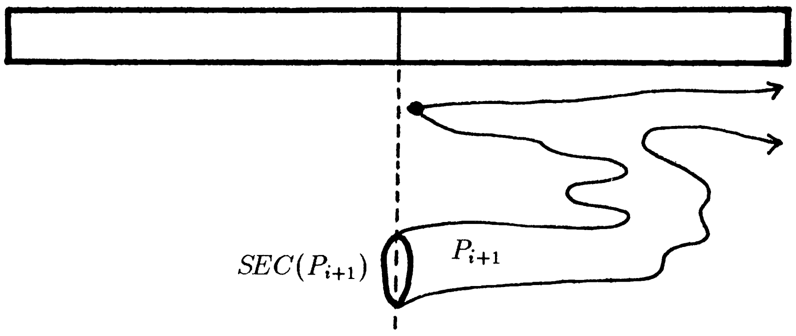


FIG. 4. *Computation subtree P_{i+1}.*

The argument is almost the same for an odd i and is therefore omitted. It may also be omitted in the formal justification of the construction. □

CLAIM 2. *If k is fixed, then:* (i) *there exists an integer i such that $S^k_{x;y}(i) = \phi$, and* (ii) *if $S^k_{x;y}(i) = \phi$, then $S^k_{x;y}(i+1) = S^k_{x;y}(i+2) = \cdots = \phi$.*

Proof. Part (i) is almost obvious by the restriction that all (t, j) in $S^k_{x;y}(i)$ satisfy $j \leq k$ and by definition (iii) of $S^k_{x;y}(i)$. (ii) If $S^k_{x;y}(i) = \phi$, then by the definition $MINS^k_{x;y}(i) = MINS^k_{x;y}(i-2)$. One can see that if some (t, j) were in $S^k_{x;y}(i+1)$, then they would also be in $S^k_{x;y}(i-1)$, which contradicts definition (iii) of $S^k_{x;y}(i)$ again. □

Now we define $S^k_{x;y}$. Fix some input string xy that is accepted by M and integer k. Then by Claim 2 the sequence $S^k_{x;y}(1), S^k_{x;y}(2), \cdots$ becomes empty at some moment and remains empty after that. Let $S^k_{x;y}(p)$ be the last member of the sequence that is not empty. Then $S^k_{x;y}$ is defined as $(S^k_{x;y}(1), S^k_{x;y}(2), \cdots, S^k_{x;y}(p))$.

CLAIM 3. *Let xyz be an input string. Then $S^k_{x;yz} = S^k_{xy;z}$ implies $S^k_{x;yz} = S^k_{xy;z} = S^k_{x;z}$.*

Proof. By Claim 1 both $S^k_{xy;z}(1)$ and $S^k_{x;z}(1)$ depend only on z, and therefore they must be the same. Then

$$S^k_{xy;z}(1) = S^k_{x;yz}(1) = S^k_{x;z}(1)$$

by the assumption. This and Claim 1 imply that $S^k_{x;yz}(2) = S^k_{x;z}(2)$ and therefore that

$$S^k_{xy;z}(2) = S^k_{x;yz}(2) = S^k_{x;z}(2).$$

We can continue this argument step by step until we get to

$$S^k_{xy;z}(p+1) = S^k_{x;yz}(p+1) = S^k_{x;z}(p+1) = \phi. \qquad □$$

CLAIM 4. *Suppose that $S^k_{x;yz} = S^k_{xy;z}$. Then if* ATM *M accepts input xyz using k, but not $k-1$ or less, space, then it also accepts xz using k, but not $k-1$ or less, space.*

Proof. The proof is very similar to the last portion of the proof of Proposition 3 (see Fig. 5). Let

$$\begin{aligned} MINS^k_{x;y} = \{(t,j) \mid\ & (t,j) \in S^k_{x;y}(i) \text{ for some odd } i \leq p \text{ and} \\ & (t,j') \notin S^k_{x;y}(i) \text{ for any odd } i \leq p \text{ if } j' < j\}. \end{aligned}$$

The definition means that if (t, j) is in $MINS^k_{x;y}$, then the global state t at the right side of the boundary can lead to acceptance with j ($\leq k$), but not with $j-1$ or less, space.

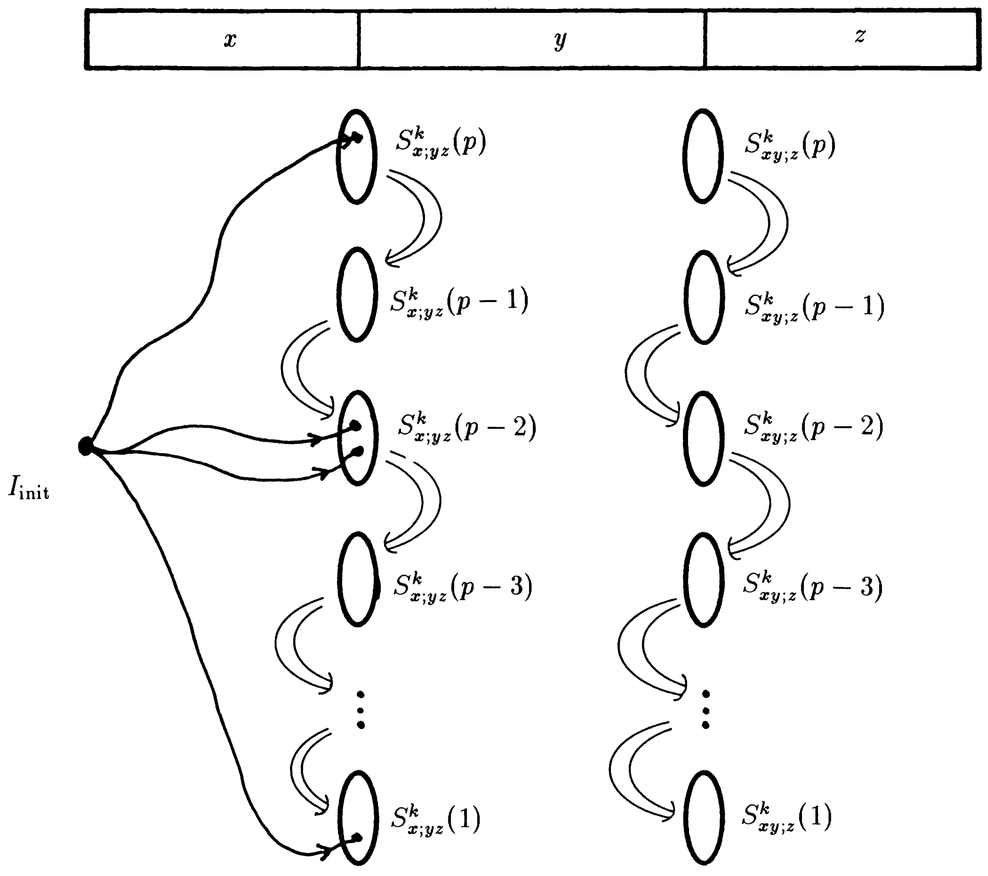


FIG. 5. *Cut-and-paste in off-line ATMs.*

Now suppose that M accepts xyz using k space. Then one can see that there is a computation subtree P_0 (having root I_{init} and for which all paths end when the input head crosses the boundary for the first time) such that

$$SEC_{x;yz}(P_0) \subseteq \{t \mid (t,j) \in MINS^k_{x;yz}\}.$$

We next consider the input xz. Since $S^k_{x;yz} = S^k_{xy;z}$, $S^k_{x;yz} = S^k_{x;z}$ by Claim 3. Then it is clear from the definition that $MINS^k_{x;yz} = MINS^k_{x;z}$. Hence the computation subtree P_0 introduced above satisfies the condition that

$$SEC_{x;z}(P_0) \subseteq \{t \mid (t,j) \in MINS^k_{x;z}\},$$

which says that xz is also accepted while using k space.

Similarly, we can show that if xz is accepted while using k' space, then xyz is also accepted while using k' space under the condition that $S^k_{x;yz} = S^k_{xy;z}$. That is enough to prove the claim. □

Now we are ready to carry out the enumeration. Suppose that $v = xyz$ is the shortest string accepted by M while using k space. Then by Claim 4 $S^k_{x;yz}$ and $S^k_{xy;z}$ must be different for any $y \neq \epsilon$. Recall that $S^k_{x;yz} = (S^k_{x;yz}(1), S^k_{x;yz}(2), \cdots, S^k_{x;yz}(p))$. Each

$S^k_{x;yz}(i)$ consists of pairs like (t, j) of a global state and an integer $\leq k$. The number of all different such pairs is bounded by c^k for a constant c, and therefore the number of all different $S^k_{x;yz}(i)$'s is bounded by 2^{c^k}. A key fact is that our construction of $S^k_{x;yz}(i)$'s guarantees that $S^k_{x;yz}(i_1)$ and $S^k_{x;yz}(i_2)$ are disjoint if i_1 and i_2 are both odd (or even). Since $S^k_{x;yz}(i)$ is a subset of the (at most) c^k different elements, it follows that

$$p/2 \leq c^k.$$

Thus the number of different $S^k_{x;yz}$ is bounded by

$$(2^{c^k})^p \leq (2^{c^k})^{2c^k} = 2^{2c^{2k}} \leq 2^{d^k}$$

for some constant d, which is enough to claim the theorem. □

Appendix. Let *EXBIN* be the language that contains strings of the form

$$\sharp u_0 \sharp v_0 \sharp u_1 \sharp v_1 \sharp \cdots \sharp u_i \sharp v_i \sharp \cdots \sharp u_n \sharp v_n \sharp\sharp u \sharp v \sharp$$

such that (i) $n \geq 0$ and $v_i = bin(i)$ (see §2 for $bin(i)$), (ii) u_i is any string over $\{0, 1\}$ and $|u_i| = |v_i|$, and (iii) for some j, $u = u_j$ and $v = v_j$. As for Theorem 2 below, a different language that is based on the prime-number theory and plays the same role as *EXBIN* is found in [3]. However, the current language would give us a more intuitive image of the difference of abilities between small space-bounded NTMs and ATMs as well as between on-line ATMs and off-line ATMs (Theorem 3).

THEOREM 2. (i) *EXBIN needs* $\log n$ *nondeterministic space (off-line).* (ii) $\log\log n$ *alternating on-line space is enough for the same language.*

Proof. (i) Intuitively, $\log\log n$ space-bounded NTMs can check conditions (i) and (ii) but not condition (iii). In what follows, every string satisfies all the conditions above except for condition (iii). Suppose that there is a k-space-bounded off-line NTM M that recognizes *EXBIN*. We focus our attention on the boundary between $\sharp u_0 \sharp bin(0) \sharp \cdots \sharp u_n \sharp bin(n) \sharp$ and $\sharp u \sharp v \sharp$ of each string. The left-side portion is determined by $u_0, u_1, \cdots, u_n$, which we denote by $left(u_0 u_1 \cdots u_n)$. Suppose that M, if it crosses that boundary in global state t from right to left (i.e., enters $left(u_0 u_1 \cdots u_n)$ from right), can come back to the same boundary (exits $left(u_0 u_1 \cdots u_n)$) in global state s. Then let $S_t(u_0 \cdots u_n)$ denote the set of all such global states s. (Note that more than one such s may exist since M is nondeterministic.) Also let

$$S(u_0 \cdots u_n) = (S_0(u_0 \cdots u_n), S_{t_1}(u_0 \cdots u_n), \cdots, S_{t_p}(u_0 \cdots u_n)),$$

where $S_0(u_0 \cdots u_n)$ is the set of all the global states s in which M can get to the boundary for the first time from I_{init}, and t_1 through t_p are all different global states in some fixed order. Note that p is bounded by c^k for some constant c.

Now suppose that $u_0 u_1 \cdots u_n \neq u'_0 u'_1 \cdots u'_n$. Then $S(u_0 u_1 \cdots u_n)$ must be different from $S(u'_0 u'_1 \cdots u'_n)$. (The reason is as follows: Suppose that $u_i \neq u'_i$. Consider two strings

$$\begin{aligned} z_1 &= left(u_0 \cdots u_i \cdots u_n) \sharp u_i \sharp bin(i) \sharp, \\ z_2 &= left(u'_0 \cdots u'_i \cdots u'_n) \sharp u_i \sharp bin(i) \sharp \end{aligned}$$

Since z_1 is in *EXBIN*, M accepts it. Then one can see that if $S(u_0 \cdots u_n) = S(u'_0 \cdots u'_n)$, then M also accepts z_2, a contradiction.) Note that the number of different $u_0 u_1 \cdots u_n$ is at least $2^{N/d}$ for some constant d and for the length N of the input. On the other hand, the number of different $S(u_0 \cdots u_n)$ is bounded by

$$(2^{c^k})^{c^k} = 2^{c^{2k}}.$$

Then to satisfy $S(u_0 \cdots u_n) \neq S(u'_0 \cdots u'_n)$ for every two strings $u_0 \cdots u_n$ and $u'_0 \cdots u'_n$, we need the condition $2^{c^{2k}} \geq 2^{N/d}$ or

$$k \geq c_1 \log N$$

for some constant c_1.

(ii) It is not difficult to see that condition (iii) can now be checked by using $\log \log n$ space. The details are omitted. □

THEOREM 3. *Let $EXBIN^R = \{x^R \mid x \in EXBIN\}$. Then* (i) *there is a* $\log \log n$ *space-bounded off-line ATM that recognizes $EXBIN^R$, but* (ii) *at least* $\log n$ *is necessary (and sufficient) to recognize the same language by on-line* ATMs.

Proof. Part (i) is obvious by Theorem 2. (ii) It is now difficult for $\log \log n$ space-bounded ATMs to check condition (iii). Again we look at the boundary between $y = \sharp v^R \sharp u^R \sharp$ and $x = \sharp bin(n)^R \sharp u_n^R \sharp \cdots \sharp bin(0)^R \sharp u_0^R$. Now recall Proposition 3. Let S be the set of all global states at the boundary between y and x that lead to acceptance. By the same line of observation as for Theorem 2 one can see that at least $2^{N/d}$ different S's, each of which corresponds to a different x above, are necessary. Suppose that a k-space-bounded on-line ATM M recognizes $EXBIN^R$. Then

$$2^{c^k} \geq 2^{N/d}$$

should be met, which implies $k \geq c_1 \log N$ for some constant c_1. □

Remark. Recall that all the results in this appendix assume the standard definition. However, they also hold under the strong definition except for the sufficient part in (ii) of Theorem 3. Under the strong definition $\log n$ is probably not sufficient since, intuitively, the on-line ATM should hold the $\sharp v^R \sharp u^R \sharp$ portion of the input in its work tape *before* it knows whether the following portion is correct (i.e., whether it is exponentially longer than $\sharp v^R \sharp u^R \sharp$).

REFERENCES

[1] M. ALBERTS, *Space complexity of alternating Turing machines*, in Lecture Notes in Computer Science, 199, Springer-Verlag, Berlin, New York, 1985, pp. 1–7.

[2] A. CHANDRA, D. KOZEN, AND L. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[3] J. CHANG, O. IBARRA, B. RAVIKUMAR, AND L. BERMAN, *Some observations concerning alternating Turing machines using small space*, Inform. Process. Lett., 25 (1987), pp. 1–9.

[4] ———, *Erratum*, Inform. Process. Lett., 27 (1988), p. 53.

[5] J. HOPCROFT AND J. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.

[6] K. IWAMA, *Low-level tradeoffs between cross and alternation*, Tech. Report, Kyoto Sangyo University, Kyoto, Japan, 1986.

[7] R. KANNAN, *Towards separating nondeterminism from determinism*, Math. Systems Theory, 17 (1984), pp. 29–45.

[8] R. LADNER, R. LIPTON, AND L. STOCKMEYER, *Alternating pushdown and stack automata*, SIAM J. Comput., 13 (1984), pp. 135–155.

[9] W. RUZZO, *Tree-size bounded alternation*, J. Comput. System Sci., 21 (1980), pp. 218–235.

[10] R. STEARNS, J. HARTMANIS, AND P. LEWIS, *Hierarchies of memory limited computations*, in Proc. IEEE Symposium on Switching Circuit Theory and Logical Design, 1965, pp. 191–202.

[11] I. SUDBOROUGH, *Efficient algorithms for path system problems and applications to alternating and time–space complexity classes*, in Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 62–73.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 147–156, February 1993

THE COMPLEXITY OF MALIGN MEASURES*

PETER BRO MILTERSEN†

Abstract. This paper analyzes the concept of *malignness*, which is the property of probability ensembles making the average case running time equal to the worst case running time for a class of algorithms. The author derives lower and upper bounds on the complexity of malign ensembles, which are tight for exponential time algorithms, and which show that no polynomial time computable malign ensemble exists for the class of polynomial time algorithms. Furthermore, it is shown that for no class of superlinear algorithms a polynomial time samplable malign ensemble exists, unless every language in P has an expected polynomial time constructor.

Key words. average case complexity, malignness

AMS(MOS) subject classifications. 68Q15, 68Q30

1. Introduction. The average case time complexity of specific algorithms has for a number of years been an active area of research, often showing significant improvement over the worst case complexity when specific distributions of the inputs were assumed. Li and Vitányi [4] studied the Solomonoff–Levin measure $\mathbf{m}$ and found that when the inputs to any algorithm are distributed according to this measure, it holds that the algorithm's average case complexity is of the same order of magnitude as its worst case complexity. More precisely,

$$\exists c > 0 : \sum_{x\in\Sigma^n} \frac{\mathbf{m}(x)}{\sum_{y\in\Sigma^n}\mathbf{m}(y)} T_A(x) \geq c \max_{x\in\Sigma^n} T_A(x).$$

In this paper, we use the term *malign* for such a measure.

The result seems significant for the following reason: Li and Vitányi argue in [5] that the Solomonoff–Levin measure should be used as the a priori probability in Bayesian reasoning because, in a certain sense, it lies close to any recursively enumerable measure (it dominates them multiplicatively). Similarly, they argue in [4] that if inputs are given from a natural source, the result implies that with high probability the average case time will be close to the worst case time, so that no improvement is possible. This would make worst case complexity the significant way to measure the cost of computation, and average case considerations with respect to, e.g., the uniform distribution much less relevant.

However, since the Solomonoff–Levin measure is not even recursive, the result has no immediate implications for the complexity theory of average case complexity, founded by Levin in [3] and extended in [2] and [1]. In Levin's approach to average case complexity, the distribution function of the input measure is required to be polynomial time computable. Thus, it is natural to ask if some kind of time bounded version of Li and Vitányi's result can be derived.

In this paper, we analyze from a complexity-theoretic perspective the property of malignness. We restate Li and Vitányi's result and give a simple proof. It seems that the surprising property of malignness is dependent upon an exponential time pattern, which in our view makes the above interpretations less obvious. We present a number of

*Received by the editors November 28, 1990; accepted for publication (in revised form) November 21, 1991. A preliminary version of this paper appeared in the Proceedings of the 6th Structure in Complexity Theory Conference, Chicago, Illinois, June 1991. The work was partially supported by the ESPRIT II Basic Research Actions Program of the European Community under contract 3075 (project ALCOM).

†Computer Science Department, Aarhus University, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.

results that support this intuition. Our results pose a limit on the results achievable in the average case direction by the time bounded versions of the Solomonoff–Levin measure, which are also discussed in [4]. In general, they suggest that if inputs are produced by a polynomial time adversary, phenomena like the above will not arise.

2. Notation.

• We consider the fixed binary alphabet $\Sigma = \{0,1\}$. Σ^* is the set of binary strings with the usual ordering, first by length and then lexicographically, and Σ^n is the set of strings of length n. By $x-1$ we denote the string preceding x, and by $x+1$ the string following x. The empty string is denoted Λ. The length of the string x is denoted by $|x|$. $\langle\cdot,\cdot\rangle$ denotes a polynomial time bijective pairing function $\Sigma^* \times \Sigma^* \to \Sigma^*$ with polynomial time projections.

• A *measure* μ on a finite or countable set M is a function from M to the real numbers, with $\mu(x) \geq 0$ for all x. Given a subset $A \subseteq M$, we define

$$\mu(A) = \sum_{x \in A} \mu(x).$$

A *probability measure* is a measure with $\sum_{x\in M} \mu(x) = 1$. Given a measure μ on Σ^* or Σ^n, we denote by μ^* its distribution function $\mu^*(x) = \mu(\{y | y \leq x\})$. If $\mu(\Sigma^n) \neq 0$, the nth derived measure of μ is the probability measure on Σ^n defined by

$$\mu_n(x) = \frac{\mu(x)}{\mu(\Sigma^n)}.$$

• A function $f : \Sigma^* \to \boldsymbol{R}$ is called *polynomial time computable* if there exists a polynomial time Turing machine transducer, which on input $\langle x, 1^i\rangle$ produces the binary notation of a number t with

$$|f(x) - t| \leq 2^{-i}.$$

In general, if the machine runs in time at most $g(|x|, i)$, we say that f is computable in time $g(|x|, i)$.

• Our model for *algorithms* is, for concreteness, multitape Turing machines. The results are easily transferred to other models. An algorithm takes one input x, and it always halts. Algorithms cannot be effectively enumerated of course, but the Turing machines are assumed enumerated $A_1, A_2, \ldots$ such that simulation, including stepcounting, etc. of n steps of A_i on input x can be performed in time polynomial in i,$|x|$ and n.

• Given an algorithm A, we define $T_A(x), x \in \Sigma^*$ to be its running time on the binary string x. Given a function $f : \boldsymbol{N} \to \boldsymbol{N}$, we define

$$ALG(f) = \{A | T_A(x) \leq f(|x|) \text{ for all } x\text{, except a finite number}\}.$$

$$T_A^w(n) = \max_{|x|=n} T_A(x)$$

is the algorithm's worst case running time. We denote by $w_A(n)$ the lexicographically least string in Σ^n with $T(w_A(n)) = T_A^w(n)$.

$$T_A^a(\mu, n) = \sum_{|x|=n} \mu_n(x) T_A(x)$$

is the algorithm's average case running time with respect to the measure μ.

3. Malign measures. In this section malignness is defined and Li and Vitányi's result on the Solomonoff–Levin measure is presented. We give a direct proof and skip the conceptual developments of [4]. We consider the class of Turing machines, where each machine has three tapes:

- A binary input tape, infinite in one direction, with the restriction that the head can only move in this direction. Thus, the input tape is one-way, one-way infinite;
- A two-way, infinite, work tape;
- A one-way, one-way infinite binary output tape.

The input tape and the output tape are started with their heads on the first square, and a machine must be able to determine by itself when it has read its input. Now consider an acceptable enumeration $M_1, M_2, \ldots$ of the above class of Turing machines, and let U be a machine universal for the class, i.e., U will on input 1^i0t, where t is an infinite tape, halt if and only if the machine M_i on input tape t halts, and U will in that case output whatever M_i outputs.

We next consider the input tape of U filled with the results of an infinite sequence of coin tosses. The Solomonoff–Levin measure $\mathbf{m}(x)$ of a string $x \in \Sigma^n$ is then defined as the probability that U halts, outputting x. Since U of course has a nonzero probability of not halting, we have that

$$\sum_{x\in\Sigma^*} \mathbf{m}(x) < 1.$$

The Solomonoff–Levin measure was first defined rigorously (for continuous sample spaces) in [8]. Intuitively, it gives a large amount of measure to strings with lots of pattern, since these have short programs that have a high probability of appearing. Actually, it is closely tied to self-delimiting Kolmogorov complexity, since $\mathbf{m}(x) = \Theta(2^{-K(x)})$, where $K(x)$ is the self-delimiting Kolmogorov complexity of x, but we do not need this result here (see [8] for a proof, and [5] and [6] for general discussions of the properties of $\mathbf{m}$).

DEFINITION 3.1. A measure μ is malign for an algorithm A if and only if there exists a $c > 0$ such that for all sufficiently large n,

$$T_A^a(\mu, n) \geq cT_A^w(n).$$

It is malign for a class of algorithms $\mathcal{A}$ if it is malign for each $A \in \mathcal{A}$.
The following is the main result from [4] on average case complexity.

THEOREM 3.1 (Li and Vitányi). *The Solomonoff–Levin measure* $\mathbf{m}$ *is malign for all algorithms.*

Proof. Consider the following Turing machine M, of the above kind.

```
Read the prefix $1^i0$, $i \geq 0$ from the tape.
Simulate $U$ on the rest of the tape.
if $U$ halts then
        $n := |U\text{'s output}|$
        simulate $A_i$ on all inputs of length $n$, finding the lexicographically
            least worst case output, $w_{A_i}(n)$ (This may not halt.)
        output $w_{A_i}(n)$
fi
```

Assume $M = M_k$ in the above enumeration, and assume that i is the index of an algorithm, i.e., that A_i halts on all inputs. If U is started with the tape 1^k01^i0t, where U, started on t, outputs a string of length n, U will output $w_{A_i}(n)$. The events of reading 1^k01^i0 and reading t are independent. But this means that for all n,

$$\mathbf{m}(w_{A_i}(n)) \geq 2^{-k-i-2}\mathbf{m}(\Sigma^n).$$

But then

$$T^a_{A_i}(\mathbf{m}, n) \geq \frac{\mathbf{m}(w_{A_i}(n))}{\mathbf{m}(\Sigma^n)} T^w_{A_i}(n) \geq 2^{-k-i-2} T^w_{A_i}(n). \qquad \square$$

Observe that the proof uses that the worst case input to A_i of length n can be described in the following way: "The worst case input to A_i of length n." Thus $w_{A_i}(n)$ has a short description, i.e., lots of pattern. The problem with the pattern "The worst case input to A_i of length n" is that it may take at least exponential time to get from the description to the result. Thus, the pattern is computationally difficult. The main object of this paper is to establish that this is the way it has to be. In general it does not hold that inputs that are slow to process have an easy pattern. To make this more precise, we want to answer questions of the following kind: Suppose μ is malign for a class of algorithms $\mathcal{A}$. What complexity does μ have?

If the complexity of a measure is defined in a suitable way, it is our belief that natural input sources are unlikely to have exponential complexity. Indeed, we would not expect the source to have resources that are exponentially larger than the resources of the algorithm to which it supplies inputs. This motivates searching for lower bounds on the complexity of malign ensembles. As in the theory of Average-*NP* [3], [2], [1], it seems natural to take the computational complexity of the distribution function μ^* as the complexity of μ. However, if we put no further restrictions on μ, this does not seem to be the correct approach, as the following theorem shows.

THEOREM 3.2. *For each general recursive function f there exists a measure μ that is malign for $ALG(f)$ and whose distribution function μ^* is polynomial time computable.*

Proof. The idea of the proof is to let the jth digit of $\mu_n(x)$ be 1 if and only if x is the lexicographically least worst case input of A_j. In order to remain within the time bounds, we let $\mu(\Sigma^n)$ vanish rapidly. Define

$$T(x, i, t) = \min(T_{A_i}(x), t).$$

$$w(n, i, t) = \min\{y \in \Sigma^n | \forall x \in \Sigma^n : T(y, i, t) \geq T(x, i, t)\},$$

i.e., $w(n, i, t)$ is the lexicographically least worst case input of A_i of size n, when A_i is restricted to run for at most t time steps. Let g be a time constructible function with $g(n) \geq f(n)$ for all n. Define

$$v(n, i) = w(n, i, g(n)).$$

Observe that $v(n, i)$ can be computed in time $t(n, i) = q(2^n p(n, i, g(n)))$, where p and q are polynomials, by simulating A_i on all inputs of length n. Put

$$u(n) = t(n, n).$$

By choosing p and q appropriately (sufficiently large), u can be made time constructible. We can without loss of generality assume that $u(n + 1) \geq u(n) + n$. Now define

$$b(x,i) = \begin{cases} 0 & \text{if } i \le u(|x|), \\ 0 & \text{if } u(|x|) < i \le u(|x|) + |x| \quad \text{and} \quad v(|x|, i - u(|x|)) \ne x, \\ 1 & \text{if } u(|x|) < i \le u(|x|) + |x| \quad \text{and} \quad v(|x|, i - u(|x|)) = x, \\ 0 & \text{if } u(|x|) + |x| < i, \end{cases}$$

$$\mu(x) = \sum_{i=1}^{\infty} b(x,i) 2^{-i}.$$

Since $v(n,j) \in \Sigma^n$, we have

$$\mu(\Sigma^n) = \sum_{i=u(n)+1}^{u(n)+n} 2^{-i} < 2^{-u(n)}.$$

Fix an algorithm $A_j \in ALG(f)$. We have that $v(n,j) = w_{A_j}(n)$ for sufficiently large n. But then

$$b(w_{A_j}(n), u(n) + j) = 1 \quad \text{for } n \ge j$$

and, therefore,

$$\mu(w_{A_j}(n)) \ge 2^{-u(n)-j},$$

$$\mu_n(w_{A_j}(n)) = \frac{\mu(w_{A_j}(n))}{\mu(\Sigma^n)} > 2^{-j},$$

$$T^a_{A_j}(\mu, n) \ge \mu_n(w_{A_j}(n)) T_{A_j}(w_{A_j}(n)) > 2^{-j} T^w_{A_j}(n).$$

Thus μ is malign for $ALG(f)$. It remains to show that μ^* is polynomial time computable.

$$\mu^*(x) = \sum_{j<|x|} \mu(\Sigma^j) + \mu(\{y | |y| = |x| \wedge y \le x\}).$$

The ith binary digit of $\sum_{j<|x|} \mu(\Sigma^j)$ is 1 if and only if there exists an $m \in \{0, \ldots, |x|-1\}$ so that $u(m) < i \le u(m)+m$. This can be decided in time polynomial in $|x|$ and i, since u is time constructible. For the second term, observe that the ith binary digit of $\mu(\{y||y| = |x| \wedge y \le x\})$ is 1 if and only if $u(|x|) < i \le u(|x|) + |x|$ and $v(|x|, i - u(|x|)) \le x$. The inequality $u(|x|) < i \le u(|x|) + |x|$ can be checked in polynomial time and if it is found to hold, the calculation of $v(|x|, i - u(|x|))$ can be done in time

$$t(|x|, i - u(|x|)) \le t(|x|, |x|) = u(|x|) < i. \qquad \square$$

Of course, the theorem (and in particular its proof) suggests that the complexity of the measure μ itself is not particularly relevant when we are interested in properties such as being malign for a class. This is no big surprise, since we only use the derived probability measures μ_n. The time bound is achieved by letting $\mu(\Sigma^n)$ vanish very rapidly, so that the nonzero digits of $\mu(x)$ appear so late that we have a sufficient amount of time to compute them. The fact that μ is not necessarily normalized, i.e., that $\mu(\Sigma^n)$ may not be 1, thus seems to be giving us an unreasonable advantage in the computation of μ^*. If we instead look at the time required to compute the normalized $\mu^*_{|x|}(x)$, no such trick will work. It is, therefore, still reasonable to conjecture that it requires time exponential in x to compute this number, since the computation of a digit seems to require running an algorithm on all inputs of size $|x|$. Consequently, from now on we require that the measures we consider are normalized.

4. Malign ensembles.

DEFINITION 4.1. A probability ensemble (or merely ensemble) is a function $\mu : \Sigma^* \rightarrow [0,1]$, with $\mu(\Sigma^n) = 1$ for all n.

Thus, if μ is an ensemble, $\mu(x) = \mu_{|x|}(x)$ for all x.

DEFINITION 4.2. An ensemble μ is called polynomial time computable if and only if its family of distribution functions $x \rightarrow \mu^*_{|x|}(x)$ is polynomial time computable. We denote by PE the class of polynomial time computable ensembles. In general, we say that an ensemble is computable in time $f(|x|, i)$ if $\mu^*_{|x|}$ is computable in time $f(|x|, i)$, in the sense of §2.
This definition relativizes in the obvious way.

We can, through the use of essentially the same technique as in the proofs of Theorems 3.1 and 3.2, construct malign ensembles and hence provide upper bounds for the time required to compute malign ensembles for certain classes of algorithms.

THEOREM 4.1. *There exists a polynomial p so that for any time constructible function f there exists an ensemble μ, computable in time $p(2^{|x|}f(|x|), i)$ so that μ is malign for $ALG(f)$.*

Proof. Define

$$T(x, i) = \min(T_{A_i}(x), f(|x|)).$$

By our assumptions on the enumeration A_i, $T(x, i)$ can be computed in time $p_1(f(|x|), i)$, where p_1 is a polynomial, which, by the time constructibility of f, does not depend upon f. Define

$$w(n, i) = \min\{x \in \Sigma^n | \forall y \in \Sigma^n : T(y, i) \leq T(x, i)\}.$$

The function w can be computed in time $p_2(2^n, p_1(f(n), i))$, i.e., in time $p_3(2^n f(n), i)$.

$$b(x, i) = \begin{cases} 1 & \text{if } w(|x|, i) = x, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mu_{|x|}(x) = \sum_{i=0}^{\infty} b(x, i) 2^{-i}.$$

μ is a probability ensemble. It can be computed in the required time because the ith binary digit of $\mu^*_{|x|}(x)$ is 1 if and only if $w(|x|, i) \leq x$. It only remains to show that it is malign for $ALG(f)$. But for this we observe that if A_j is such an algorithm, $T(x, j) = T_{A_j}(x)$ for sufficiently large $|x|$, and for these x, $b(x, j) = 1$ if x is the lexicographically least worst case input of size $|x|$ for A_j. But then

$$T^a_{A_j}(\mu, n) \geq 2^{-j} T^w_{A_j}(n)$$

for sufficiently large n. □

Of course, if $f \in 2^{\Omega(n)}$, the factor $2^{|x|}$ can be omitted from the stated time, i.e., for classes of exponential time algorithms, we can compute the ensemble almost as fast as the algorithms run.

COROLLARY 4.1. *There exists an ensemble μ, computable in time $p(2^{|x|}, i)$, where p is a polynomial, so that μ is malign for the class of polynomial time algorithms.*

Theorem 4.1 and the corollary reflect our intuition from the proof of Theorem 3.1: Malignness can be obtained if we are willing to use exponential time. By using the same technique, we can provide a recursive measure that is malign for classes of algorithms with some recursive upper bound on their running time. It won't provide us with a recursive measure for the class of all algorithms, and, as is proved below, no such thing exists. We now turn to a negative result, complementing Theorem 4.1.

THEOREM 4.2. *There is an $\epsilon > 0$ and a polynomial p, such that for all nondecreasing time constructible functions f with $f \in \Omega(p)$, there is no ensemble μ, malign for $ALG(f)$ and computable in time $f(|x|)^\epsilon h(i)$, where h is any function.*

Proof. The idea of the proof is the following: Given an ensemble μ, we can find a string y with low μ-value by binary search. We can then construct an algorithm A for which μ is not malign by making sure A runs for a long time on the input y.

Given a nondecreasing, time constructible function g, $g \in \Omega(n)$ and any function h, consider an ensemble μ, computable in time $g(|x|)h(i)$. We may assume that h is recursive, since $h(i)$ otherwise can be replaced with $\max_n T_\mu(n, i)$, which is recursive. We may furthermore assume that h is time constructible, strictly increasing, and tends to infinity, since any general recursive function is dominated by such a function. Define

$$\tilde{h}(n) = \min(\max(1, \max\{j | h(j) < \log(n)\}), n).$$

By h's time constructibility, $\tilde{h}$ can be computed in polynomial time. Furthermore, the polynomial time bound does not depend upon h. Consider the following algorithm, B:

```
input $x$
$y := \Lambda$
for $i := 1$ to $\tilde{h}(|x|)$ do
    $v_1 :=$ a $2^{-i}$-approximation to $\mu_{|x|}\{z | z < y0^{|x|-i+1}\}$
    $v_2 :=$ a $2^{-i}$-approximation to $\mu^*_{|x|}(y01^{|x|-i})$
    $v_3 :=$ a $2^{-i}$-approximation to $\mu^*_{|x|}(y1^{|x|-i+1})$
    if $v_2 - v_1 \leq v_3 - v_2$ then
        $y := y0$
    else
        $y := y1$
    fi
od
$y := y0^{|x|-|y|}$
if $x = y$ then
    idle for $q(g(|x|)\log(|x|))^2$ time steps ($q$ being specified below)
fi
```

The function q should be a polynomial such that $q(g(|x|)\log(|x|))$ is an upper bound for the running time of the algorithm when $x \neq y$. Observe that q can be picked independently of μ, g, and h. We may assume that q is of the form $q(t) = t^\alpha$. Put $\epsilon = \frac{1}{3\alpha}$. Putting $g(n) = f(n)^\epsilon$, we have that B halts on almost all inputs in time $q(\log(|x|)f(|x|)^\epsilon)^2 = \log(|x|)^{2\alpha} f(|x|)^{\frac{2}{3}}$, which is less than $f(|x|)$ for almost all $|x|$. Thus, $B \in ALG(f)$. By an easy induction, the invariant

$$\mu_{|x|}\{yz | z \in \Sigma^{|x|-i}\} \leq \left(\frac{3}{4}\right)^{i-3}$$

holds at the end of the ith cycle of the for loop, so the y found by the for loop has

$$\mu_{|x|}(y) \leq \left(\frac{3}{4}\right)^{\tilde{h}(|x|)-3}$$

We then have

$$T_B^a(n) \le \left(\frac{3}{4}\right)^{\tilde{h}(n)-3} T_B^w(n) + \sqrt{T_B^w(n)}.$$

But this is smaller than $cT_B^w(n)$, for any $c > 0$, for sufficiently large n, i.e., μ is not malign for B, which was to be proved. □

For exponential time algorithms the lower bound matches the upper bound of Theorem 4.1 within a polynomial. The theorem implies that no recursive measure is malign for the class of all algorithms and thus that the Solomonoff–Levin measure is not recursive (of course, this can be proven in more elementary ways, see [6]). For polynomial time algorithms, we have the following.

COROLLARY 4.2. *No ensemble $\mu \in PE$ is malign for the class of polynomial time algorithms.*

5. Malign ensembles for classes of fast algorithms. Corollary 4.2 still leaves something to be desired. After all, most algorithms we are likely to run will be in, e.g., $ALG(n^4)$. It still seems that exponential time is required to compute malign ensembles for such classes, even if our lower bound is much smaller. However, we are not likely to be able to prove these intuitions correct because the following theorem tells us that in order to show that superpolynomial time is necessary, we would have to prove $P \neq NP$. We are reusing the technique from the previous constructions.

THEOREM 5.1. *For all k, an ensemble $\mu \in PE^{\Sigma_2^p}$ exists, which is malign for $ALG(n^k)$.*

Proof. Put $f(n) = n^k$ in the proof of Theorem 4.1. This makes $T(x, i)$ polynomial time computable. Observe that the ith binary digit of $\mu^*_{|x|}(x)$ is 1 if and only if

$$\exists y \in \Sigma^n \, \forall z \in \Sigma^n \, [y \le x \wedge T(z, i) \le T(y, i)],$$

and this is a Σ_2^p-problem. □

It thus seems that we will have to concentrate on merely making the existence of such a PE-ensemble unlikely, instead of trying to prove that it does not exist. We will indeed do this, by deriving from it a complexity-theoretic equality that seems unreasonable, although it is a lot weaker than $P = NP$. For this, we need a result on sampling.

DEFINITION 5.1. An ensemble μ is polynomial time samplable if a polynomial time probabilistic Turing machine exists, which on input $\langle 1^n, 1^i \rangle$ produces a string of length n, $M(n, i)$ such that for all $x \in \Sigma^n$,

$$|\Pr(M(n, i) = x) - \mu_n(x)| \le 2^{-i}.$$

Thus, the polynomial time samplable ensembles are those ensembles that can be approximated as the output distributions of polynomial time probabilistic algorithms. A similar definition and an analogy to the following theorem can be found in [1]. Note that we cannot demand that the strings are produced with the exact probability, since it is easy to see that no probabilistic machine with a worst case time bound can produce a string with an irrational probability. Thus, demanding exact sampling would make the following theorem false.

THEOREM 5.2. *Every ensemble μ in PE is polynomial time samplable.*

The proof is essentially equal to the one in [1] and is, therefore, omitted. We might note that the converse result holds if and only if $P = PP$.

DEFINITION 5.2. Given a language L. An expected polynomial time constructor for L is a probabilistic algorithm that on input 1^n produces an $x \in L \cap \Sigma^n$ in expected polynomial time if one exists and otherwise fails to halt.

This is a natural generalization of the deterministic constructors defined and studied by Sanchis and Fulk in [7]. A useful equivalence is the following.

LEMMA 5.1. *Every $L \in P$ has an expected polynomial time constructor if and only if every $L \in DTIME(n)$ has one.*

Proof. Suppose every $L \in DTIME(n)$ has a constructor. Let J be a language in P. There is a constant c, such that $J \in DTIME(n^c)$. Define $\tilde{J} = \{x10^{|x|^c} | x \in J\}$. It is easy to see that $\tilde{J} \in DTIME(n)$, so, by assumption, $\tilde{J}$ has a constructor, $\tilde{C}$. By running $\tilde{C}$ on input 1^{n+1+n^c} and extracting the first n digits of the output if $\tilde{C}$ halts, we get a string in $L \cap \Sigma^n$. □

THEOREM 5.3. *Suppose a polynomial time samplable ensemble μ is malign for $ALG(f)$, where f is a time constructible function with $n \in o(f)$. Then every $L \in P$ has an expected polynomial time constructor.*

Proof. By Lemma 5.1, we need only to show that every $L \in DTIME(n)$ has a constructor. Consider the following algorithm A_L:

```
Input $x$
Decide if $x \in L$.
If it is not, halt immediately.
If it is, wait $f(n)/2$ time-steps before halting.
```

A_L is an $ALG(f)$-algorithm, so there exists a c so that $T_A^a(\mu, n) \geq cT_A^w(n)$. We will show that for sufficiently large n,

$$L \cap \Sigma^n \neq \emptyset \Rightarrow \mu_n(L \cap \Sigma^n) \geq \frac{c}{2}.$$

Assume not, i.e., $\mu_n(L \cap \Sigma^n) < c/2$. Then

$$\sum_{x \in L \cap \Sigma^n} \mu_n(x) T_A(x) < \frac{c}{2} T_A^w(n).$$

We also have that for sufficiently large n:

$$\sum_{x \in \Sigma^n \setminus L} \mu_n(x) T_A(x) \leq n.$$

But then

$$\frac{c}{2} T_A^w(n) + n \geq T_A^a(\mu, n) \geq cT_A^w(n).$$

If $L \cap \Sigma^n \neq \emptyset$, the algorithm will run for $f(n)/2$ time steps for some value of x. Therefore,

$$n \geq \frac{c}{2} T_A^w(n) \geq \frac{cf(n)}{4},$$

which is a contradiction for sufficiently large n. Let M be a polynomial time sampler for μ. By the definition of sampling, if $L \cap \Sigma^n \neq \emptyset$, then $\Pr(M(n, \lceil -\log \frac{c}{4} \rceil + n) \in L \cap \Sigma^n) \geq (c/2) - (c/4) = c/4$. Thus, running M several times on input $< 1^n, 1^{\lceil -\log \frac{c}{4} \rceil + n} >$ until an element of L is produced, is a construction of such an element in expected polynomial time. □

The following theorem (an analog to Proposition 4.1 in [7]) makes expected polynomial time constructors for all languages in P unlikely.

DEFINITION 5.3. Let RE be the class of languages L for which there exist a probabilistic Turing machine, running in time $2^{c|x|}$ on input x, rejects if $x \notin L$, and accepts with probability at least $\frac{1}{2}$ if $x \in L$.

Thus, RE is for $E = \cup_{c\geq 0} DTIME(2^{cn})$ what RP is for P. Hence, RE should be considered a rather small extension of E, and $RE = NE$, where $NE = \cup_{c\geq 0} NTIME(2^{cn})$ must be considered only slightly more plausible than $E = NE$. Actually, by standard arguments $RE = NE$ if and only if there are no tally languages in $NP - RP$.

THEOREM 5.4. *If every $L \in P$ has an expected polynomial time constructor, then $RE = NE$.*

Proof. Let L be a language in NE, and let M be a nondeterministic machine, running in time 2^{cn} and recognizing L. We can represent the nondeterministic computations of M on input x as binary strings of length at most $2^{c|x|}$, where the bits represent the nondeterministic choices of M. Denote by x_n^i the lexicographically ith string of size n. Define $f : \Sigma^* \to \mathbf{N}$ by $f(x_n^i) = 2^{cn} + i$. The function f is clearly injective, provided $c \geq 1$, which we may assume. Now consider

$$\tilde{L} = \{y \mid \exists x : |y| = f(x) \text{ and } y \text{ codes an accepting computation of } M \text{on } x\}.$$

Clearly $\tilde{L} \in P$ and has, therefore, by assumption, an expected polynomial time constructor, C. Let $p(n)$ be an upper bound on C's expected running time on inputs of size n. If we simulate C on $1^{f(x)}$ for $2p(f(x))$ time steps and accept if an element has been produced by then and reject otherwise, we have an RE-acceptor for L. □

COROLLARY 5.1. *If an ensemble $\mu \in PE$ is malign for $ALG(f)$, where f is time constructible and $n \in o(f)$, then $RE = NE$.*

Thus, if we assume $RE \neq NE$, we get a superpolynomial lower bound on ensembles malign for this class of algorithms. By making the stronger assumption that there are tally sets in NP, which cannot be recognized in subexponential randomized time, the same technique gives an exponential lower bound, essentially matching the upper bound of Theorem 4.1.

Acknowledgments. Thanks to Joan Boyar for her helpful supervision on the thesis on which this paper is based and to Sven Skyum for useful comments and suggestions. Lane A. Hemachandra and Paul M.B. Vitányi pointed out some errors in the original manuscript.

REFERENCES

[1] S. BEN-DAVID, B. CHOR, O. GOLDREICH, AND M. LUBY, *On the theory of average case complexity*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, May 1989, pp. 204–216.

[2] Y. GUREVICH, *Complete and incomplete randomized NP problems*, in Proc. 28th Annual Symposium on Foundations of Computer Science, Los Angeles, CA, October 1987, pp. 111–117.

[3] L.A. LEVIN, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.

[4] M. LI AND P. M. B. VITÁNYI, *A theory of learning simple concepts under simple distributions and average case complexity for the universal distribution*, in Proc. 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, October 1989, pp. 34–39.

[5] ———, *Inductive reasoning and Kolmogorov complexity*, in Proc. 4th Annual Structure in Complexity Theory Conference, June 1989, pp. 165–185.

[6] M. LI AND P. M. B. VITÁNYI, *Kolmogorov complexity and its applications*, in Handbook of Theoretical Computer Science, Jan van Leeuwen, ed., Elsevier Science Publishers B. V., Amsterdam, 1990, pp. 187–254.

[7] L. A. SANCHIS AND M. A. FULK, *On the efficient generation of language instances*, SIAM J. Comput., 19 (1990), pp. 281–296.

[8] A. K. ZVONKIN AND L. A. LEVIN, *The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms*, Russian Math. Surveys, 25 (1970), pp. 83–124.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 157–174, February 1993

SCAN-FIRST SEARCH AND SPARSE CERTIFICATES: AN IMPROVED PARALLEL ALGORITHM FOR k-VERTEX CONNECTIVITY[*]

JOSEPH CHERIYAN[†], MING-YANG KAO[‡], AND RAMAKRISHNA THURIMELLA[§]

Abstract. Given a graph $G = (V, E)$, a *certificate* of k-vertex connectivity is an edge subset $E' \subset E$ such that the subgraph (V, E') is k-vertex connected if and only if G is k-vertex connected. Let n and m denote the number of vertices and edges. A certificate is called *sparse* if it contains $O(kn)$ edges.

For undirected graphs, this paper introduces a graph search called the *scan-first search*, and shows that a certificate with at most $k(n-1)$ edges can be computed by executing scan-first search k times in sequence on subgraphs of G. For each of the parallel, distributed, and sequential models of computation, the complexity of scan-first search matches the best complexity of any graph search on that model. In particular, the parallel scan-first search runs in $O(\log n)$ time using $C(n, m)$ processors on a CRCW PRAM, where $C(n, m)$ is the number of processors needed to find a spanning tree in each connected component in $O(\log n)$ time, and the parallel certificate algorithm runs in $O(k \log n)$ time using $C(n, m)$ processors. The parallel certificate algorithm can be employed to test the k-vertex connectivity of an undirected graph in $O(k^2 \log n)$ time using $knC(n, kn)$ processors on a CRCW PRAM. For all combinations of n, m, and $k > 3$, both the running time and the number of processors either improve on or match those of all known deterministic parallel algorithms.

This paper also obtains an online algorithm for computing an undirected graph certificate with at most $2kn$ edges, and a sequential algorithm for computing a directed graph certificate with at most $2k^2n$ edges.

Key words. parallel algorithms, PRAM, vertex connectivity, graphs

AMS(MOS) subject classifications. 05C40, 68Q22, 90B12

1. Introduction.

Graph connectivity is one of the most fundamental properties in graph theory [4]. Given a positive integer k, an undirected (or directed) graph $G = (V, E)$ with at least $k+1$ vertices is called *k-vertex connected* if the deletion of any $k-1$ vertices leaves the graph connected (respectively, strongly connected). A *certificate* for the k-vertex connectivity of G is a subset E' of E such that the subgraph (V, E') is k-vertex connected if and only if G is k-vertex connected. Let $n = |V|$ and $m = |E|$. Note that $kn/2$ is a trivial lower bound on the number of edges in a certificate for a k-vertex connected graph. For general k, it is not obvious that there is any upper bound strictly less than m on the number of edges in a certificate for k-vertex connectivity; however, nonconstructive results of Mader imply an upper bound of $O(kn)$ for undirected graph certificates [5]. We call a certificate for k-vertex connectivity *sparse* if it has $O(kn)$ edges. For instance, a spanning tree is a sparse certificate for the 1-vertex connectivity of a connected undirected graph.

Sparse certificates have applications in diverse areas of computer science. For example, they can be used for message-efficient fault-tolerant protocols in distributed computing [20], [21]. Also, they are useful for improving existing graph k-vertex connectivity algorithms. The k-vertex connectivity of a graph can be tested in two stages. Stage 1

[*]Received by the editors June 29, 1990; accepted for publication (in revised form) December 5, 1991.

[†]Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada N2L-3G1. This author's work was done at Cornell University, Ithaca, New York, and at the University of the Saarland, Saarbrücken, Germany. His research was supported in part by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research, through National Science Foundation grant DMS-8920550, and by the ESPRIT II Basic Research Actions Program of the EC under contract 3075 (project Algorithms and Complexity).

[‡]Department of Computer Science, Duke University, Durham, North Carolina 27706. The research of this author was supported in part by National Science Foundation grant CCR-9101385.

[§]Department of Mathematics and Computer Science, University of Denver, Denver, Colorado 80208. This work was done while this author was with the Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, Maryland 20742.

computes a sparse certificate of the input graph. Stage 2 applies a given k-vertex connectivity algorithm to the certificate obtained in Stage 1. Because a certificate preserves the k-vertex connectivity of the input graph, Stage 1 ensures the correctness of the test. Stage 2 can *potentially* improve the complexity of the test by reducing the size of the input to the given k-vertex connectivity algorithm.

For sequential computing, Nagamochi and Ibaraki have presented an algorithm for sparse undirected graph certificates that runs in $O(m + n)$ time [27]. For testing the k-vertex connectivity of undirected graphs, this gives sequential running times of $O(k^2n^2)$ for $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ for $k > \sqrt{n}$ when their sparse-certificate algorithm is combined with Galil's k-vertex connectivity algorithm [17]. Independently, another sequential algorithm for k-vertex connectivity with the same complexity for $k \leq \sqrt{n}$ was reported in preliminary versions of this paper [9], [8], the stated bound being achieved in the latter [8]. It is now clear that the work of Nagamochi and Ibaraki [27] was the earlier result, although unknown to us at the time of our work.

We present an algorithm for finding sparse certificates for undirected graphs, using a new graph search procedure called the *scan-first search*. This search procedure has surprisingly efficient implementations in the parallel, distributed and sequential models of computation. Consequently, our certificate algorithm also has efficient implementations in all these three models. Below we list the complexity of scan-first search on the three models. Note that the complexity of scan-first search on each model matches the best complexity of any graph search on that model.

• **Parallel computing.** Scan-first search runs in $O(\log n)$ time using $C(n,m)$ processors on a CRCW PRAM, where $C(n,m)$ is the number of processors used to compute a spanning tree in each connected component in $O(\log n)$ time. For deterministic algorithms, Cole and Vishkin have shown [11] that $C(n,m) = O((m+n)\alpha(n,m)/\log n)$, where $\alpha(n,m)$ is the inverse of Ackerman's function and has an extremely slow growth rate [12]. For randomized algorithms, Gazit has shown [18] that $C(n,m)$ achieves the optimal bound of $\Theta((m+n)/\log n)$.

• **Distributed computing.** Scan-first search runs in $O(d\log^3 n)$ time using $O(m + n\log^3 n)$ messages, where d is the diameter of the input graph.

• **Sequential computing.** Scan-first search runs in $O(m+n)$ time.

The advantage of scan-first search over the two most well-known graph search procedures is easy to see. Depth-first search runs in optimal linear time on the sequential model [28] but has no known parallel implementation that is efficient [1], [2]. Breadth-first search runs efficiently on the sequential and the distributed models [14], [3]. However, currently the best parallel implementation is no more efficient than matrix multiplication [19].

For undirected graphs, we show that a sparse certificate can be computed by executing scan-first search k times in sequence on subgraphs of G; moreover, the resulting certificate has at most $k(n-1)$ edges, only a factor of two away from the trivial lower bound (see Theorem 2.4). Combining this result with the above implementations of scan-first search shows that the complexity of our sparse-certificate algorithm on the three models is as follows: $O(k\log n)$ time using $C(n,m)$ processors on the parallel model, $O(kn\log^3 n)$ time using $O(k(m+n\log^3 n))$ messages on the distributed model, and $O(k(m+n))$ time on the sequential model.

Consider the problem of testing an undirected graph for k-vertex connectivity. Using the method sketched above, our certificate algorithm can be used as a preprocessing step, both with Khuller and Schieber's parallel algorithm [24], and with Even's sequential algorithm [13].This gives a parallel running time of $O(k^2\log n)$ using $knC(n, \min\{kn, m\})$

processors, and sequential running times of $O(k^2n^2)$ when $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ when $k > \sqrt{n}$.

For general k, there are no previous parallel or distributed algorithms for sparse certificates. The best previous parallel algorithm for (undirected) k-vertex connectivity, due to Khuller and Schieber [24], runs in $O(k^2 \log n)$ time and uses $knC(n, m)$ processors. For $m > kn$, our algorithm uses fewer processors than their algorithm. For dense graphs and for $k = O(1)$, our algorithm runs in $O(\log n)$ time and has a time-processor product that is within an alpha-factor of the trivial lower bound of $\Omega(n^2)$.

We also obtain the following results. For *undirected* graphs, we give an *online* algorithm that computes a certificate with at most $2kn$ edges. This algorithm is parallelized to run probabilistically in $O(\log^2 n)$ time using $mP(n, m)$ processors, where $P(n, m)$ is the number of processors needed to find a maximum matching in $O(\log^2 n)$ time with high probability. Currently, the best value known for $P(n, m)$ is $O(mn^{3.38})$ [26]. For *directed* graphs, we show that a certificate with at most $2k^2n$ edges can be computed in $O(km \max\{n, k\sqrt{n}\})$ sequential time.

The above discussion has highlighted the results of this paper. The following sections proceed to detail those results. Section 2 discusses scan-first search and the algorithm for computing undirected graph certificates in the three models. Section 3 presents the online algorithm for undirected graph certificates and its parallelization. Section 4 describes the sequential algorithm for directed graph certificates. Section 5 concludes the paper with open problems.

2. Scan-first search and sparse undirected graph certificates.

The main result of this section is that a sparse certificate for undirected k-vertex connectivity can be found by iteratively performing k scan-first searches on subgraphs of the input graph.

Section 2.1 defines scan-first search and discusses how to perform the search efficiently in the parallel, distributed, and sequential models of computation. Section 2.2 states the main certificate theorem based on scan-first search and discusses its algorithmic implications. Sections 2.3 and 2.4 prove the main certificate theorem, and §2.4 gives a generalization of the theorem.

2.1. Scan-first search.

Given a connected undirected graph and a specified vertex, a *scan-first search* in the graph starting from the specified vertex is a systematic way of marking the vertices. The main marking step is called *scan*: to scan a marked vertex means to mark all previously unmarked neighbors of that vertex. At the beginning of the search, only the specified starting vertex is marked. Then, the search iteratively scans a marked and unscanned vertex until all vertices are scanned.

A scan-first search in a connected undirected graph produces a spanning tree defined as follows. At the beginning of the search, the tree is empty. Then, for each vertex x in the graph, when x is scanned, all the edges between x and its previously unmarked neighbors are added to the tree; the edges between x and its previously marked neighbors are not added to the tree.

For an undirected graph that may or may not be connected, a scan-first search can be performed by applying the above search procedure to each connected component (as well as to each isolated vertex). The search produces a spanning forest with a spanning tree in each connected component.

Notice that scan-first search is more general than sequential breadth-first search [14]. In other words, all sequential breadth-first search trees are scan-first search trees, but some scan-first search trees are not breadth-first search trees. For example, let C be an undirected graph consisting of a five-vertex cycle x_1, x_2, x_3, x_4, x_5. Let e_i denote the edge $\{x_i, x_{i+1}\}$. Then, $C - \{e_2\}$, $C - \{e_3\}$, $C - \{e_4\}$ are the scan-first search trees of C rooted at x_1. However, $C - \{e_3\}$ is the only breadth-first search tree rooted at x_1.

```
Subroutine PARALLEL SCAN-FIRST SEARCH
Input: a connected undirected graph $G = (V, E)$ and a vertex $r$.
Output: a scan-first search spanning tree $T$ of $G$ rooted at $r$.
begin
    Find a spanning tree $T'$ of $G$ rooted at $r$;
    Assign a preorder numbering to the vertices in $T'$;
    For each vertex $v \in T'$ with $v \neq r$, let $b(v)$ denote the neighbor of $v$ in $G$ with
    the smallest preorder number;
    Let $T$ be the tree formed by the edges $\{v, b(v)\}$ for all $v \neq r$;
end.
```

FIG. 1. *Computing a scan-first search spanning tree in parallel.*

THEOREM 2.1. *For an undirected graph with n vertices and m edges, a scan-first search spanning forest can be found in $O(\log n)$ time using $C(n, m)$ processors on a* CRCW PRAM, *where $C(n, m)$ is the number of processors used to compute a spanning tree in each connected component in $O(\log n)$ time.*

Proof. Let G be the input graph. Without loss of generality, assume that G is connected. Let r be a given starting vertex in G. To prove the theorem, Fig. 1 describes an algorithm for finding a scan-first search spanning tree T of G rooted at the vertex r.

T is a scan-first search spanning tree of G rooted at r for the following reasons. T corresponds to a scan-first search in G starting at r with the preorder of T' being the scanning order. T is a spanning tree because every vertex except r has a neighbor with a smaller preorder number in T'.

T can be found in $O(\log n)$ time using $C(n, m)$ processors on a CRCW PRAM as follows. By the definition of $C(n, m)$, T' can be found in $O(\log n)$ time using $C(n, m)$ processors. By standard parallel algorithmic techniques [23], the preorder numbers and the neighbors $b(v)$ in Figure 1 can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors. Because $C(n, m) = \Omega((n+m)/\log n)$, the total complexity is $O(\log n)$ time using $C(n, m)$ processors. □

To find a scan-first search tree on the distributed model, we use the best distributed breadth-first search algorithm currently known, that of Awerbuch and Peleg [3].

THEOREM 2.2. *For an undirected graph with diameter d, n vertices, and m edges, a scan-first search spanning forest can be found in $O(d \log^3 n)$ distributed time using $O(m + n \log^3 n)$ messages.*

Proof. The distributed breadth-first search algorithm of Awerbuch and Peleg [3] runs in $O(d \log^3 n)$ distributed time using $O(m + n \log^3 n)$ messages. We first execute their algorithm, and then modify the resulting breadth-first search forest to give a scan-first search forest within the same complexity bounds. □

THEOREM 2.3. *For an undirected graph with n vertices and m edges, a scan-first search spanning forest can be found in $O(n + m)$ sequential time.*

Proof. The proof is easy. □

2.2. The main certificate theorem and its algorithmic implications. The next theorem shows that sparse certificates for the k-vertex connectivity of undirected graphs can be computed efficiently.

THEOREM 2.4 (The main certificate theorem). *Let $G = (V, E)$ be an undirected graph and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, \ldots, k$, let E_i be the edge set of a scan-first search forest in the graph $G_{i-1} = (V, E - (E_1 \cup \cdots \cup$

E_{i-1})). Then $E_1 \cup \cdots \cup E_k$ is a certificate for the k-vertex connectivity of G, and this certificate has at most $k(n-1)$ edges.

Theorem 2.4 has algorithmic consequences for the parallel, distributed and sequential models of computation.

THEOREM 2.5. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(k \log n)$ time using $C(n,m)$ processors on a* CRCW PRAM.

Proof. The proof is obtained by Theorems 2.4 and 2.1. □

The next theorem improves on the best previous parallel algorithms for testing the k-vertex connectivity of undirected graphs, namely, the algorithm in Khuller and Schieber [24] and the one in the preliminary version of this paper [10].

THEOREM 2.6. *For an undirected graph with n vertices and m edges, the k-vertex connectivity can be tested in $O(k^2 \log n)$ time using $knC(n,kn)$ processors on a* CRCW PRAM.

Proof. The k-vertex connectivity is tested in two steps. Step 1: Compute a sparse certificate for the k-vertex connectivity of the input graph via Theorem 2.5. Step 2: Apply the k-vertex connectivity algorithm of Khuller and Schieber [24] to the certificate found in the first step. Step 1 runs in $O(k \log n)$ time using $C(n,m)$ processors. Step 2 runs in $O(k^2 \log n)$ time using $knC(n,kn)$ processors. Because $knC(n,kn) \geq C(n,m)$, the total complexity is as stated. □

The main certificate theorem also gives an efficient algorithm on the distributed model of computation. For general k, the first distributed algorithm for undirected sparse certificates was presented in the preliminary version of this paper [10]. For $k=2$, Itai and Rodeh have previously given a distributed algorithm for undirected sparse certificates [21].

THEOREM 2.7. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(kn \log^3 n)$ distributed time using $O(k(m+n\log^3 n))$ messages.*

Proof. This proof follows from Theorems 2.4 and 2.2. For $i > 1$, notice that the diameter of G_{i-1} may increase to $\Omega(n)$. □

For sequential computation, linear-time algorithms for sparse certificates for the 2-vertex connectivity and the 3-vertex connectivity of undirected graphs have been given by Itai and Rodeh [21] and Cheriyan and Maheshwari [7], respectively. For general k, Nagamochi and Ibaraki have recently given a linear-time algorithm for sparse certificates for undirected graphs [27]. The main certificate theorem gives a slower sequential algorithm for general k.

THEOREM 2.8. *For an undirected graph with n vertices and m edges, a sparse certificate for k-vertex connectivity with at most $k(n-1)$ edges can be found in $O(k(m+n))$ sequential time.*

Proof. The proof is obtained by Theorems 2.4 and 2.3. □

For testing the k-vertex connectivity of undirected graphs, sequential running times of $O(k^2n^2)$ for $k \leq \sqrt{n}$ and of $O(k^3n^{1.5})$ for $k > \sqrt{n}$ have been reported in Nagamochi and Ibaraki [27]. The main certificate theorem gives the same sequential complexity, when combined with Even's k-vertex connectivity algorithm [13].

THEOREM 2.9. *The k-vertex connectivity of an n-vertex undirected graph can be tested in $O(k^2n^2)$ sequential time for $k \leq \sqrt{n}$ and in $O(k^3n^{1.5})$ sequential time for $k > \sqrt{n}$.*

Proof. First use Theorem 2.8 to find a sparse certificate. Then run the k-vertex connectivity algorithm of Even [13], [14]. □

COROLLARY 2.10. *For $k = O(1)$, the k-vertex connectivity of an n-vertex undirected graph can be tested in $O(n^2)$ sequential time.*

Proof. The proof follows from Theorem 2.9. □

2.3. The proof of the main certificate theorem. The main certificate theorem states that a certificate with at most $k(n-1)$ edges can be computed by successively finding the edge set E_1 of a scan-first search spanning forest of $G_0 = G$, the edge set E_2 of a scan-first search spanning forest of $G_1 = (V, E - E_1), \ldots$, the edge set E_k of a scan-first search spanning forest of $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$, and taking their union $E_1 \cup \cdots \cup E_{k-1}$.

For $i = 1, \ldots, k$, let F_i denote the spanning forest computed by the ith scan-first search (i.e., F_i has edge set E_i), and let H_i denote the subgraph $(V, (E_1 \cup \cdots \cup E_i))$. See Fig. 2 for an example illustrating the definitions of G_i, F_i and H_i. Clearly, the theorem holds if H_k is k-vertex connected. To prove the theorem by contradiction, suppose that H_k is not k-vertex connected, and that G is k-vertex connected. Then there is a subset S of at most $k-1$ vertices such that $H_k - S$ is disconnected, by Menger's theorem. The next lemma shows that at least one tree of the last scan-first search forest F_k must contain vertices of two or more connected components of $H_k - S$.

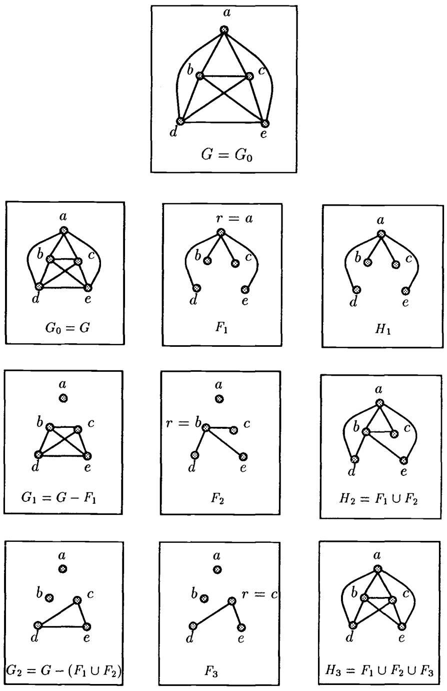


FIG. 2. *An example illustrating the definitions of G_i, F_i, and H_i when G is K_5.*

LEMMA 2.11. *Suppose that H_k is not k-vertex connected, and that G is k-vertex connected. Then the following two statements hold:*

1. *There is a subset $S \subset V$ with $|S| < k$ such that $H_k - S$ is disconnected;*
2. *F_k contains a simple tree path P_k whose two endpoints are in different connected components of $H_k - S$.*

Proof. Focus on the second statement. By the k-vertex connectivity of G, the graph $G - S$ obtained by deleting S from G is connected because S contains $k-1$ or fewer vertices. Because $H_k - S$ is disconnected and $G - S$ is connected, there exists an edge

e in G whose endpoints are in two different connected components of $H_k - S$. The edge e is not in H_k, and so is not in $E_1 \cup \cdots \cup E_{k-1}$. Hence the edge e is in $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$, and so the two endpoints of e are in the same connected component of G_{k-1}. Because F_k is a scan-first search forest in G_{k-1}, the forest F_k has a spanning tree for the connected component in G_{k-1} that contains e. Therefore, F_k contains a simple tree path P_k between the two endpoints of e. This shows that the second statement holds. □

To finish the proof of Theorem 2.4, the following discussion proceeds to show that the path P_k of Lemma 2.11 cannot exist, yielding the desired contradiction.

A few definitions are in order. Let ω denote the size of S. The proof of Theorem 2.4 makes crucial use of the following indexing scheme of S. Let $s_1, \ldots, s_\omega$ denote the vertices of S such that s_i is the first vertex in $S - \{s_1, \ldots, s_{i-1}\}$ that is scanned by the ith search. Because $\omega < k$ by the first statement of Lemma 2.11, this indexing scheme is well defined and establishes a one-to-one onto correspondence between the vertices in S and the forests $F_1, \ldots, F_\omega$.

For each s_i in S, the *home component* of s_i is defined as follows. In the forest F_i, let r be the root of the tree that contains s_i. There are three cases.

Case 1. If $r \notin S$, then the home component of s_i is the connected component in $H_k - S$ that contains r.

Case 2. If $r \in S$ and $r \neq s_i$, then the home component of s_i is the home component of r.

Case 3. If $r = s_i$, then s_i has no home component.

Figure 3 illustrates the definition of home components. The next lemma shows that the definition is consistent.

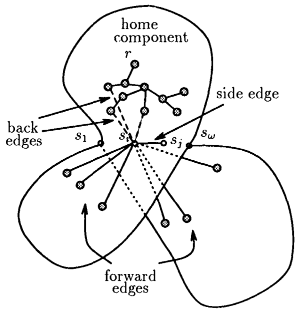


FIG. 3. *Illustration of the definitions of home components and of forward, back, and side edges (of vertex s_i).*

LEMMA 2.12. *For each $s_i \in S$, if s_i satisfies Case 1 or Case 2 of the above definition, then the home component of s_i is a connected component of $H_k - S$. If s_i satisfies Case 3, then s_i has no home component.*

Proof. The lemma is obviously true if s_i satisfies Case 1 or Case 3 of the definition. For Case 2, the lemma is shown by induction on the indices of S.

Induction Hypothesis: For all $j < i$, if s_j satisfies Case 2 of the above definition, then the home component of s_j is a connected component of $H_k - S$.

Induction Step: The goal is to show that s_i has a home component. Because s_i satisfies Case 2, the root of the tree in F_i that contains s_i is some $s_h \in S$.

CLAIM 1. $h < i$.

Proof. Note that $i \neq h$ because $s_h \neq s_i$ by the definition of Case 2. To prove this claim by contradiction, assume that $h > i$. Then, $s_h \in S - \{s_1, \ldots, s_{i-1}\}$. Moreover, because s_i is a descendant of s_h in F_i, the ith search scans the vertex s_h before s_i. Thus, the vertex s_i should not have been indexed i. This is a contradiction. Consequently, h must be smaller than i. This finishes the proof of Claim 1. □

CLAIM 2. *No tree of F_h has s_h as its root.*

Proof. To prove this claim by contradiction, assume that s_h is a root in F_h. Then the edges in G that are adjacent to s_h and are not in $F_1, \ldots, F_{h-1}$ would all be included in F_h. Consequently, s_h would be an isolated vertex in G_h. Because $i > h$ by Claim 1, s_h would be an isolated vertex in F_i. This contradicts the fact that s_i is a descendant of s_h in F_i. Therefore, s_h is not a root in F_h. This finishes the proof of Claim 2.

To show that s_i has a home component, by the definition of Case 2, it suffices to prove that s_h has a home component. If Case 1 holds for s_h, then by definition, s_h has a home component. If Case 2 holds for s_h, then by Claim 1 and the induction hypothesis, s_h has a home component. Case 3 cannot hold for s_h by Claim 2. This finishes the proof of the lemma. □

To describe the key properties of scan-first search, more definitions are needed. For each vertex $s \in S$ if s has a home component, let $hcc(s)$ denote the home component of s; if s has no home component, let $hcc(s)$ denote s itself. Moreover, for each $v \in V - S$, let $hcc(v)$ denote the connected component in $H_k - S$ that contains v.

Given a forest F_i, a *jump* of F_i is a simple tree path $Q = v_1, \ldots, v_q$ of F_i with $hcc(v_1) \neq hcc(v_q)$.

For each vertex $s \in S$, the edges incident with s in G are classified into three types as follows. The *back* edges of s are those between s and its home component. The *side* edges of s are those between s and $S - \{s\}$. The rest of the edges incident with s are the *forward* edges. Note that if s does not have a home component, then it has no back edges. Refer to Fig. 5 for an illustration of these definitions.

The next lemma shows a key property of the first scan-first search, and the following lemma, which is the critical one for the proof of the main certificate theorem, generalizes the key property to the first i scan-first searches for all $i \in \{1, 2, \ldots, \omega\}$.

LEMMA 2.13. *The following two statements are true.*

1. *Every jump of F_1 contains at least one vertex of S.*
2. *The scan-first search forest F_1 contains all the forward edges of s_1.*

Proof. To prove the first statement by contradiction, assume that F_1 has a jump $Q = v_1, \ldots, v_q$ which contains no vertices from S. Then, $v_1, v_q \in V - S$. By the definition of hcc, $hcc(v_1)$ and $hcc(v_q)$ are connected components in $H_k - S$. By the definition of a jump, $hcc(v_1) \neq hcc(v_q)$. In sum, Q is a path in H_k that connects two different connected components of $H_k - S$. Therefore, Q must contain a vertex from S, contradicting the assumption of the proof. This finishes the proof of the first statement.

To prove the second statement, note that F_1 is a tree because $G_0 = G$ is connected. Let r be the root of F_1. Then there are two cases: either $r \in S$ or $r \notin S$.

Case 1. $r \in S$. By the definition of the indexing scheme of S, $r = s_1$. When the first search scans s_1, none of the neighbors of s_1 in G have been marked. Consequently, F_1 includes all edges incident with s_1 in G, and hence the lemma holds for this case.

Case 2. $r \notin S$. Refer to Fig. 4(ii) for an illustration of this case. By definition, s_1 has a home component and $hcc(r) = hcc(s_1)$. To prove this case by contradiction, assume that there is a forward edge $e = \{s_1, x\} \notin F_1$. This can happen only if the first search marks x before it scans s_1. Focus on the tree path, say, Q of F_1 from r to x. Because e is a forward edge, $hcc(s_1) \neq hcc(x)$. Therefore, $hcc(r) \neq hcc(x)$ and the path Q is a jump of F_1. By the first statement of this lemma, Q contains a vertex $s \in S$. Because s is an ancestor of x in F_1, the first search scans s before or when it marks x. Hence the first search scans s before it scans s_1. However, this contradicts the definition of the indexing scheme of S, because s_1 should have received a higher index. This finishes the proof of case (2) of the second statement, and completes the proof of the lemma. □

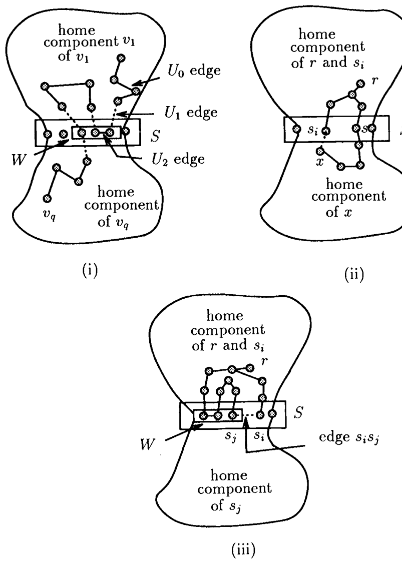


FIG. 4. *Proof of the main certificate theorem.* (i) *Proof of Claim* 3 (*in Lemma* 2.14) *and Lemma* 2.15. (ii) *Proof of Lemma* 2.13 *and Claim* 4 (*in Lemma* 2.14). (iii) *Proof of Claim* 5 (*in Lemma* 2.14).

LEMMA 2.14. *For each $i \in \{1, 2, \ldots, \omega\}$, the following statements are true.*

1. *Every jump of F_i contains at least one vertex in $S - \{s_1, \ldots, s_{i-1}\}$.*
2. *$F_1, \ldots, F_i$ contain the following edges of G :*
   (a) *All forward edges of s_i, and*
   (b) *All side edges $\{s_i, s_j\}$ with $i > j$ and $hcc(s_i) \neq hcc(s_j)$.*

Proof. This lemma is proved by induction on i as follows. The base step follows from Lemma 2.13; note that Statement 2(b) holds vacuously for $i = 1$. The induction hypothesis is that the lemma holds for $i = t$. The induction step shows that the lemma holds for $i = t + 1$. This is proved by the following three claims.

CLAIM 3. *Statement* 1 *holds for $i = t + 1$.*

To prove the above claim by contradiction, assume that F_{t+1} has a jump $Q = v_1, \ldots, v_q$ which contains no vertices from $S - \{s_1, \ldots, s_t\}$. Refer to Fig. 4(i) for an illustration. Let W be the set of vertices that are in both Q and S. Let U_0 be the set of edges in Q that each have two endpoints in $H_k - S$. Let U_1 be the set of edges in Q that each have one endpoint in W and the other endpoint in $H_k - S$. Let U_2 be the set of edges in Q that each have two endpoints in W. Observe that for all edges $\{x, y\} \in U_0$, $hcc(x) = hcc(y)$. Next, from the assumption of the proof, $W \subseteq \{s_1, \ldots, s_t\}$. Therefore, from statement 2(a) of the induction hypothesis, the edges in U_1 cannot be forward edges and hence for all edges $\{x, y\} \in U_1$, $hcc(x) = hcc(y)$. Furthermore, because the edges in U_2 are side edges, from statement 2(b) of the induction hypothesis, for all edges $\{x, y\} \in U_2$, $hcc(x) = hcc(y)$. Hence, for all vertices $x, y \in Q$, $hcc(x) = hcc(y)$. In particular, $hcc(v_1) = hcc(v_q)$, contradicting the assumption that Q is a jump. This finishes the proof of Claim 3.

CLAIM 4. *Statement* 2(a) *holds for* $i = t + 1$.

To prove the above claim, let T be the tree of F_{t+1} that contains s_{t+1} and let r be the root of T. Then there are two cases: (1) $r \neq s_{t+1}$ or (2) $r = s_{t+1}$.

Case 1. $r \neq s_{t+1}$. Refer to Fig. 4(ii) for an illustration of this case. By definition, s_{t+1} has a home component and $hcc(r) = hcc(s_{t+1})$. To prove this case by contradiction, assume that there is a forward edge $e = \{s_{t+1}, x\} \notin F_{t+1}$. This can happen only if the $(t + 1)$th search marks x before it scans s_{t+1}. From the existence of e, the tree T contains x. So T contains a tree path Q from r to x. Because e is a forward edge, $hcc(s_{t+1}) \neq hcc(x)$. Therefore, $hcc(r) \neq hcc(x)$, and Q is a jump of F_{t+1}. By Claim 3, Q contains a vertex $s \in S - \{s_1, \ldots, s_t\}$. Notice that $x \neq s$ because $x \in H_k - S$. Therefore, s is an ancestor of x in T. Consequently, the $(t + 1)$th search scans s before or when it marks x. Thus, the $(t + 1)$th search scans s before it scans s_{t+1}. Therefore, s_{t+1} should not have been indexed by $t+1$, contradicting the indexing scheme of S. This finishes the proof of Case 1 of Claim 4.

Case 2. $r = s_{t+1}$. Notice that when the $(t + 1)$th search scans s_{t+1}, none of the neighbors of s_{t+1} in G_t have been marked. So F_{t+1} includes all edges incident with s_{t+1} in G_t. Consequently, $F_1, \ldots, F_{t+1}$ include all edges incident with s_{t+1} in G and statement 2(a) holds for Case 2. This finishes the proof of Case 2 of Claim 4 and the proof of Claim 4.

CLAIM 5. *Statement* 2(b) *holds for* $i = t + 1$.

To prove the above claim, let T be the tree of F_{t+1} that contains s_{t+1}, and let r be the root of T. Then there are two cases: (1) $r \neq s_{t+1}$ or (2) $r = s_{t+1}$.

Case 1. $r \neq s_{t+1}$. Refer to Fig. 4(iii) for an illustration of this case. By definition, s_{t+1} has a home component and $hcc(r) = hcc(s_{t+1})$. To prove this case by contradiction, assume that there is a side edge $e = \{s_{t+1}, s_j\} \notin F_{t+1}$ such that $t+1 > j$ and $hcc(s_{t+1}) \neq hcc(s_j)$. This can happen only if the $(t+1)$th search marks s_j before it scans s_{t+1}. From the existence of e, the tree T contains s_j. So T contains a tree path Q from r to s_j. Notice that $hcc(r) = hcc(s_{t+1}) \neq hcc(s_j)$. Therefore, Q is a jump of F_{t+1}. Next, let W be the set of vertices that are in both S and Q. Notice that $s_j \in W$. Also, s_j is a proper descendant of all vertices in $W - \{s_j\}$, and the $(t + 1)$th search scans $W - \{s_j\}$ before it marks s_j. In sum, the $(t + 1)$th search scans $W - \{s_j\}$ before it scans s_{t+1}. Then, by the indexing scheme of S, $W - \{s_j\} \subseteq \{s_1, \ldots, s_t\}$. By the assumption of the proof, $s_j \in \{s_1, \ldots, s_t\}$. Hence, $W \subseteq \{s_1, \ldots, s_t\}$; by Claim 3, this contradicts the assumption that Q is a jump of F_{t+1}. This finishes the proof of Case 1 of Claim 5.

Case 2. $r = s_{t+1}$. This case is exactly the same as case (2) of Claim 4. This finishes the proof of case (2) of Claim 5, the proof of Claim 5, and the proof of the induction step. □

To complete the proof of the main certificate theorem, recall our assumption that H_k is not k-vertex connected, while G is k-vertex connected. Then Lemma 2.11 shows that F_k must contain a path P_k whose endpoints are in two different connected components of $H_k - S$, where S is a subset of V with $|S| < k$ whose deletion disconnects H_k. The next lemma shows that the path P_k cannot exist, using the same argument as in the proof of Claim 3 in Lemma 2.14, and thus completes the proof of Theorem 2.4.

LEMMA 2.15. *The path* P_k *of the second statement of Lemma* 2.11 *cannot exist.*

Proof. To prove the lemma by contradiction, assume that P_k exists. Let $P_k = v_1, \ldots, v_q$. Refer to Fig. 4(i) for an illustration. Because the two endpoints of P_k are in two different connected components of $H_k - S$, $hcc(v_1) \neq hcc(v_q)$ and the path P_k is a jump of F_k. Let W be the set of vertices that are in both P_k and S. Let U_0 be the set of edges in P_k that each have two endpoints in $H_k - S$. Let U_1 be the set of edges in P_k that each have one endpoint in W and the other endpoint in $H_k - S$. Let U_2 be the set of edges in P_k that each have two endpoints in W. Observe that for all edges $\{x, y\} \in U_0$, $hcc(x) = hcc(y)$. Next, because $W \subseteq S = \{s_1, \ldots, s_\omega\}$, by statement 2(a) of Lemma 2.14, the edges in U_1 cannot be forward edges and hence for all edges $\{x, y\} \in U_1, hcc(x) = hcc(y)$. Furthermore, because the edges in U_2 are side edges, from statement 2(b) of Lemma 2.14, for all edges $\{x, y\} \in U_2$, $hcc(x) = hcc(y)$. In sum, for all vertices $x, y \in P_k$, $hcc(x) = hcc(y)$. In particular, $hcc(v_1) = hcc(v_q)$, contradicting the assumption that P_k is a jump. This finishes the proof of Lemma 2.15. □

2.4. A generalization of the main certificate theorem. This section gives a generalization of the main certificate theorem, and discusses its applications.

For two distinct vertices x and y in G, the *local connectivity* of x and y, denoted $\kappa(x, y)$, is the maximum number of internally vertex-disjoint paths between x and y in G. A *certificate of local connectivity* k for G is a subset E' of E such that for every two distinct vertices x and y, $\kappa'(x, y) \geq \min\{k, \kappa(x, y)\}$, where $\kappa'(x, y)$ denotes $\kappa(x, y)$ for the subgraph (V, E'). A certificate of local connectivity k is said to be sparse if it has $O(kn)$ edges.

THEOREM 2.16 (The generalized certificate theorem). *Let $G = (V, E)$ be an undirected graph, and let n denote the number of vertices. Let k be a positive integer. For $i = 1, 2, \ldots, k$, let E_i be the edge set of a scan-first search forest in the graph $G_{i-1} = (V, E - (E_1 \cup \cdots \cup E_{i-1}))$. Then $E_1 \cup \cdots \cup E_k$ is a certificate of local connectivity k for G, and this certificate has at most $k(n-1)$ edges.*

Proof. The proof is essentially the same as that of the main certificate theorem. Let H_k be the subgraph $(V, (E_1 \cup \cdots \cup E_k))$, and let $\kappa_k(x, y)$ denote $\kappa(x, y)$ for H_k. To prove the theorem by contradiction, assume that $\kappa_k(u, w) < \min\{k, \kappa(u, w)\}$ for some two vertices $u, w \in V$ with $u \neq w$.

Although Lemma 2.11 does not apply now because it supposes that G is k-vertex connected, we first show that the two statements in Lemma 2.11 hold under the assumption that $\kappa_k(u, w) < \min\{k, \kappa(u, w)\}$ for some two vertices $u, w \in V$ with $u \neq w$. By Menger's theorem, there exist two vertices $u', w' \in V$ and a set $S \subseteq V - \{u', w'\}$ such that (1) $|S| = \kappa_k(u, w)$, (2) u' and w' are in different connected components of $H_k - S$, and (3) u' and w' are in the same connected component of $G - S$. By properties (1) and (2), there is a subset $S \subset V$ with $|S| < k$ such that $H_k - S$ is disconnected. By property (3), there exists an edge e in $G - S$ whose endpoints are in two different connected components of $H_k - S$. Clearly, the edge e is not in H_k, and hence e is in $G_{k-1} = (V, E - (E_1 \cup \cdots \cup E_{k-1}))$. So the two endpoints of e are in the same connected component of G_{k-1}. Because F_k is a scan-first search forest in G_{k-1}, the forest F_k has a spanning tree for the connected component in G_{k-1} that contains e. Therefore, F_k contains a simple tree path P_k whose two endpoints are in different connected components of $H_k - S$. This finishes the proof of the two statements in Lemma 2.11.

With the two statements in Lemma 2.11 proven, we can complete the proof of this theorem by using Lemmas 2.13, 2.14, and 2.15 to show that the above path P_k cannot exist. □

The next corollary is useful for computing the k-separators of a graph. For a positive integer k, a *k-separator* of G is a set S of k vertices such that $G - S$ has more connected components than G.

COROLLARY 2.17. *For a positive integer $k < n$ and for all $i \in \{1, \ldots, k-1\}$, G and $H_k = (V, (E_1 \cup \cdots \cup E_k))$ have the same i-separators.*

Proof. The proof is straightforward by Theorem 2.16. □

Some of the recent algorithmic research on k-connected graphs has focussed on highly efficient parallel algorithms for finding k-vertex connected components and k-separators for small k, namely, $k = 2$, 3, and 4. Theorem 2.16 and Corollary 2.17, when combined with Theorem 2.5, yield immediate improvements to several of these algorithms.

An undirected graph is said to be *bridge-connected* if for each edge the deletion of that edge leaves the graph connected. For a bridge-connected graph, Fussell and Thurimella [16] have given an algorithm for finding an open ear decomposition for each biconnected component on an $O(\sqrt{n}/\log n \times \sqrt{n}/\log n)$ mesh of trees architecture in $O(\log^3 n)$ time. We improve the running time to $O(\log^2 n)$ by first finding a sparse certificate of local connectivity 2 for the input graph, and then running their algorithm on the subgraph induced by the certificate. The main bottleneck of the original algorithm [16] is to compute, for a given spanning tree T of the input graph, the nearest common ancestor in T of all nontree edges. Note that by executing their algorithm on a sparse certificate, the worst-case number of nontree edges decreases from $\Omega(n^2)$ to $O(n)$.

THEOREM 2.18. *Let $G = (V, E)$ be a bridge-connected graph, and let n denote the number of vertices. An open ear decomposition for each biconnected component of G can be found on an $O(\sqrt{n}/\log n \times \sqrt{n}/\log n)$ mesh of trees architecture in $O(\log^2 n)$ time.*

Fussell, Ramachandran, and Thurimella [15] have given an algorithm for computing the triconnected components of an undirected graph in $O(\log n)$ time with a time-processor product of $O((m+n)\log\log n)$. We obtain the following improvement by first finding a sparse certificate of local connectivity 3 for the input graph, and then running their algorithm on the subgraph induced by the certificate.

THEOREM 2.19. *Let $G = (V, E)$ be an undirected biconnected graph, and let n and m denote the number of vertices and edges. The 3-vertex connected components of G can be found on an* ARBITRARY-CRCW PRAM *in $O(\log n)$ time with a time-processor product of $O(m\alpha(n, m) + n \log\log n)$.*

For an undirected triconnected graph, Kanevsky and Ramachandran [22] have given an algorithm for finding a compact representation of all the 3-separators. Their algorithm runs in $O(\log^2 n)$ time with a time-processor product of $O(n^2 \log^2 n)$. We obtain the following improvement in two steps. First, find a sparse certificate E' of local connectivity 4 for the input graph $G = (V, E)$. Then, for each vertex v, use the algorithm of Fussell, Ramachandran, and Thurimella [15] to find all the 2-separators of $(V, E') - \{v\}$.

THEOREM 2.20. *Let $G = (V, E)$ be an undirected triconnected graph, and let n denote the number of vertices. An $O(n^2)$ representation for all the 3-separators of G can be found on an* ARBITRARY-CRCW PRAM *in $O(\log n)$ time with a time-processor product of $O(n^2 \log\log n)$.*

The algorithm of Kanevsky and Ramachandran [22] also tests the input graph for 4-vertex connectivity in $O(\log^2 n)$ time using $O(n^2)$ processors. Note that for testing 4-vertex connectivity, our Theorem 2.6 gives a running time of $O(\log n)$ using $4nC(n, 4n) = O(n^2\alpha(n, 4n)/\log n)$ processors.

3. An online algorithm for sparse undirected graph certificates and its parallelization.

We first present a sequential online algorithm for finding sparse certificates for the k-vertex connectivity of undirected graphs. Then we parallelize the algorithm to obtain a randomized NC algorithm with a complexity independent of k.

Let $G = (V, E)$ denote the input undirected graph to our online certificate algorithm. Let n and m denote the numbers of vertices and edges of G. G is given one edge at a time; the input order $e_1, \ldots, e_m$ of the edges is arbitrary. Upon receiving an edge e_i, the online certificate algorithm must decide whether to include e_i in the final certificate $F \subseteq E$. Once an edge is included, it cannot be deleted from F at a later step; similarly, if an edge is not included, it cannot be added to F later. Initially, F is empty. Our online algorithm incrementally updates F by examining each input edge $\{v, w\}$ and including $\{v, w\}$ in F if and only if the subgraph induced by the current F has at most $k - 1$ vertex-disjoint paths between v and w.

The detailed description of our online certificate algorithm is given in Fig. 5. For an example, refer to Fig. 6. The next lemma shows that the F output by the online certificate algorithm is indeed a certificate of k-vertex connectivity.

```
Subroutine ONLINE CERTIFICATE
Input: the edges of an undirected graph $G = (V, E)$ given one at a time in an arbitrary
order $e_1, \ldots, e_m$.
Output: a sparse certificate $F \subseteq E$ for the $k$-vertex connectivity of $G$.
begin
    $F := \emptyset$;
    for $i := 1$ to $m$ do
        begin
            Let $v$ and $w$ denote the endpoints of $e_i$;
            Let $k_i$ denote the maximum number of vertex-disjoint paths between $v$
            and $w$ in $(V, F)$;
            if $k_i < k$ then $F := F \cup \{e_i\}$ else $F$ remains unchanged;
        end
end.
```

FIG. 5. *Computing a sparse certificate online.*

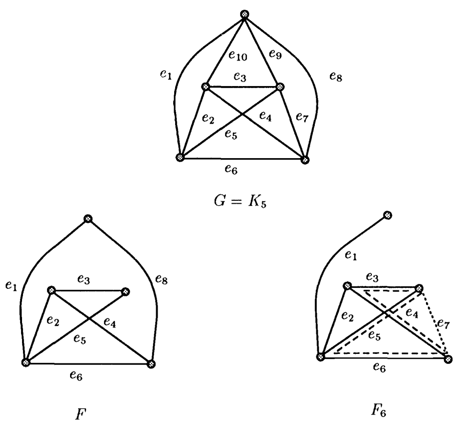


FIG. 6. *Using the online algorithm to find a certificate for 2-vertex connectivity. The edge ordering is* $e_1, e_2, \ldots, e_{10}$. *F does not contain e_7, for example, because in $F_6 = \{e_1, e_2, \ldots, e_6\}$ there are two vertex-disjoint paths between the endpoints of e_7.*

LEMMA 3.1. *If $G = (V, E)$ is k-vertex connected, then the final subgraph (V, F) is also k-vertex connected.*

Proof. To prove the lemma by contradiction, suppose that G is k-vertex connected but (V, F) is not k-vertex connected. Then, there is a set S of less than k vertices such that $(V, F) - S$ is disconnected. Let I be a connected component of $(V, F) - S$. Since $G - S$ is connected, G has an edge $\{v, w\}$ with $v \in I$ and $w \in V - (I \cup S)$. Because (V, F) has at most $|S|$ vertex-disjoint paths between v and w and because $|S| < k$, the online certificate algorithm would have added the edge $\{v, w\}$ to F when it examined $\{v, w\}$. This contradicts the assumption of the proof. Thus the lemma is true. □

Next we prove that the final F has at most $2kn$ edges by combining the following lemma with a theorem due to Mader.

LEMMA 3.2. *The final subgraph (V, F) does not contain any subgraph that is $(k+1)$-vertex connected.*

Proof. To prove the lemma by contradiction, suppose that (V, F) contains a $(k+1)$-vertex connected subgraph J. Let $e_i = \{v, w\}$ be the edge with the largest index among all the edges in J. In other words, e_i is the last edge added to J by the online certificate algorithm. Then $J - \{e_i\}$ has k vertex-disjoint paths between v and w because $J - \{e_i\}$ is k-vertex connected. Therefore, when the algorithm examined e_i, it would not have added e_i to F. This contradicts the assumption of the proof and finishes the proof the lemma. □

THEOREM 3.3 (Mader [5]). *For an integer $k \geq 1$, if an undirected graph has at least $2k - 1$ vertices and at least $(2k-1)(n-k)+1$ edges, where n denotes the number of vertices, then it contains a $(k+1)$-vertex connected subgraph.*

LEMMA 3.4. *The final F has at most $2kn$ edges.*

Proof. If $n \geq 2k - 1$, then the lemma follows from Lemma 3.2 and Theorem 3.3. If $n < 2k - 1$, then F contains at most $n(n-1)/2$ edges, which is less than $2kn$. □

The next theorem summarizes the discussion of the online certificate algorithm.

THEOREM 3.5. *Let G be an n-vertex undirected graph. Assume that the edges of G are given one at a time. Then a certificate for the k-vertex connectivity of G with at most $2kn$ edges can be computed on line in $O(k^2n)$ time per input edge.*

Proof. The correctness of the online certificate algorithm follows from Lemmas 3.1 and 3.4. As for the running time, note that for each edge e_i, the algorithm attempts to find k vertex-disjoint paths between the endpoints of e_i in (V, F). This takes time proportional to k times the size of (V, F) [14]. Therefore, the running time for examining one edge is $O(k^2n)$. □

A fast parallel version of the online certificate algorithm can be obtained by a parallel greedy method as follows. Let $E_0 = \emptyset$. For $i = 1, \ldots, m$, let E_i denote the edge set $\{e_1, e_2, \ldots, e_i\}$. For $i = 1, \ldots, m$, test in parallel whether the graph (V, E_{i-1}) has at least k vertex-disjoint paths between the endpoints of e_i. The certificate F contains exactly those edges e_i that fail the test.

Notice that the main difference in the computations executed by the above parallel algorithm and the online one is that the maximum number of vertex-disjoint paths between the endpoints of e_i is found in the subgraph (V, E_{i-1}) by the parallel algorithm and in (V, F) by the online algorithm. The next lemma shows that this difference does not affect F.

LEMMA 3.6. *The online certificate algorithm and its parallel version compute the same final F.*

Proof. For $t = 1, \ldots, m$, let F_t denote the F found by the online algorithm after it processes e_t, and let F_0 denote the empty set. Also, let C_p and C_q denote the final F computed by the parallel algorithm and the online algorithm, respectively. The goal is to show that $C_q = C_p$. Observe that $C_p \subseteq C_q$ because $F_{t-1} \subseteq E_{t-1}$ for all t. Thus, to prove the lemma by contradiction, it suffices to assume that there exists an edge $e_i = \{v, w\} \in C_q - C_p$. Then, by definition, there are at least k vertex-disjoint paths between v and w in (V, E_{i-1}) and there are less than k vertex-disjoint paths between v and w in (V, F_{i-1}). Therefore, there exists a set $S \subseteq V - \{v, w\}$ with $|S| < k$ such that (1) v and w are disconnected in $(V, F_{i-1}) - S$, and (2) there exists an edge $e_j \in E_{i-1} - F_{i-1}$ whose endpoints are in two different connected components of $(V, F_{i-1}) - S$. Notice that $j \leq i - 1$ because $e_j \in E_{i-1}$. Moreover, $e_j \notin F_j$ because $e_j \notin F_{i-1}$ and $F_j \subseteq F_{i-1}$. On the other hand, observe that (V, F_{i-1}) contains less than k vertex-disjoint paths between the endpoints of e_j. Then, because $F_{j-1} \subseteq F_{i-1}$, (V, F_{j-1}) also contains less than k vertex-disjoint paths between the endpoints of e_j. But then the online algorithm would have included e_j in F_j. This contradicts the earlier conclusion that $e_j \notin F_j$. Therefore, the assumption of the proof is incorrect and the lemma is correct. □

The next theorem summarizes the discussion of the parallelization of the online certificate algorithm. Let $P(n, m)$ denote the number of processors needed to find a maximum matching in $O(\log^2 n)$ time with high probability. Currently, the best value known for $P(n, m)$ is $O(m \cdot n^{3.38})$ [26].

THEOREM 3.7. *Given an undirected graph with n vertices and m edges, a certificate for k-vertex connectivity with at most $2kn$ edges can be found by a randomized algorithm in $O(\log^2 n)$ time using $m \cdot P(n, m)$ processors on a* CRCW PRAM.

Proof. The correctness of the above parallel algorithm follows from Lemmas 3.6, 3.1, and 3.4. As for the complexity, to test in parallel for the existence of k vertex-disjoint paths between two vertices, we use the well-known method of transforming this problem to the maximum matching problem [6]. □

4. A sequential algorithm for small certificates for directed graphs.

The main result of this section is stated in the following theorem.

THEOREM 4.1. *Let $G = (V, E)$ be a directed graph with n vertices and m edges. A certificate for the k-vertex connectivity of G with at most $2k^2 n$ edges can be found in $O(k \cdot m \cdot \max\{n, k\sqrt{n}\})$ sequential time.*

Notice that the running time of our directed graph certificate algorithm is the same as the best time complexity known for testing directed k-vertex connectivity [17]. Mader has shown an $O(kn)$ upper bound for the minimum size of a certificate for directed k-vertex connectivity [25]. For $k > 1$, our directed graph certificate algorithm is the best algorithm known for finding a certificate of size at most $n \cdot k^{O(1)}$.

Our proof of Theorem 4.1 is based on the next lemma. It allows certain edges of a vertex to be deleted without affecting the connectivity elsewhere. For a directed graph G and for every two distinct vertices $x, y \in G$, let $\kappa_G(x, y)$ denote the maximum number of internally vertex-disjoint directed paths from x to y in G.

LEMMA 4.2. *Let $G = (V, E)$ be a directed graph. Let z, u, v be vertices in G. Let e be an incoming edge of u in G. Let $H = (V, E - \{e\})$. If $\kappa_H(z, u) \geq k$ and $\kappa_G(z, v) \geq k$, then $\kappa_H(z, v) \geq k$.*

Proof. To prove the lemma by contradiction, assume that $\kappa_H(z, v) < k$. Then, to derive a contradiction by Menger's theorem, it suffices to show that for every vertex subset $S \subseteq V - \{z, v\}$ with $|S| < k$, there exists a directed path in $H - S$ from z to v.

Because $\kappa_G(z, v) \geq k$, G contains k internally vertex-disjoint directed paths $P_1, \ldots,$ P_k from z to v. Because $\kappa_H(z, v) < k$, the edge e must be in one of the P_i's. Without loss of generality, assume that e is in P_k. Let L be the subpath of P_k from u to v. Note that L is a directed path in H. Also, note that because the P_i's are internally vertex-disjoint, $P_1, \ldots, P_{k-1}$ cannot contain e and therefore remain directed paths in H.

There are two cases based on whether L and S intersect.

Case 1. $L \cap S \neq \emptyset$. Note that $|S - L| \leq k - 2$. Because $P_1, \ldots, P_{k-1}$ are internally vertex-disjoint in H, at least one of those P_i's does not contain any vertices from S, and remains a directed path from z to v in $H - S$. This is a desired contradiction and thus completes the proof of Case 1.

Case 2. $L \cap S = \emptyset$. Note that L is directed path in $H - S$ from u to v. Because $\kappa_H(z, u) \geq k$, H contains k internally vertex-disjoint directed paths $Q_1, \ldots, Q_k$ from z to u. Because $Q_1, \ldots, Q_k$ are internally vertex-disjoint and $|S| \leq k - 1$, at least one Q_j does not contain any vertices from S. Then Q_j remains a directed path from z to u in $H - S$. Therefore, the directed path formed by Q_j and L is a directed path from z to v in $H - S$. This is a desired contradiction and thus finishes the proof of Case 2. □

The next lemma uses Lemma 4.2 to compute an important subset of a desired certificate for a directed graph. In the lemma, let $G = (V, E)$ be a directed graph with n vertices and m edges. Let z be a vertex in G.

LEMMA 4.3. *Assume that $\kappa_G(z, v) \geq k$ for all $v \in V - \{z\}$. Then there exists a subgraph $H = (V, E')$ computable in $O(k \cdot m \cdot n)$ time with the following properties*:

1. *For all $v \in V - \{z\}$, $\kappa_H(z, v) = k$*;
2. *The indegree of z in H is zero*; *and*
3. *For all $v \in V - \{z\}$, the indegree of v in H is exactly k.*

Note that because of the indegree constraints, H is a minimum-size subgraph with $\kappa_H(z, v) = k$ for all $v \in V - \{z\}$.

Proof. H can be computed by the algorithm FIND-H given in Fig. 7. Note that deleting the incoming edges of z does not affect $\kappa(z, v)$. Then, the correctness of FIND-H follows directly from repeated applications of Lemma 4.2. As for the time complexity, each iteration of the do loop takes $O(k \cdot m)$ time [14]. So the total runing time is $O(k \cdot m \cdot n)$. □

```
Subroutine FIND-H
Input: a digraph $G = (V, E)$ and a vertex $z \in V$ with $\kappa_G(z, v) \geq k$ for all $v \in V - \{z\}$.
Output: a minimum-size subgraph $H = (V, E')$ with $\kappa_H(z, v) = k$ for all $v \in V - \{z\}$.
begin
    Let $E'$ be obtained from $E$ by removing all incoming edges of $z$;
    Let $v_1, v_2, \ldots, v_{n-1}$ be the vertices in $V - \{z\}$;
    for $i = 1$ to $n - 1$ do
        begin
            Find $k$ internally vertex-disjoint directed paths from $z$ to $v_i$ in the sub-
            graph $(V, E')$;
            Delete from $E'$ all incoming edges of $v_i$ except the $k$ incoming edges in
            the $k$ paths just found above;
        end
end.
```

FIG. 7. *Computing a subgraph H of Lemma* 4.3.

Our directed graph certificate combines the subroutine FIND-H and the classic k-vertex connectivity algorithm of Even [14]. The certificate algorithm computes a certificate F for the input graph G in five steps as follows.

1. Test the k-vertex connectivity of G. If G is not k-vertex connected, then let $F = \emptyset$ and stop; otherwise, continue the computation.

2. Pick k arbitrary vertices $x_1, \ldots, x_k \in V$. For all x_i and x_j with $i < j$, compute k internally vertex-disjoint directed paths in G from x_i to x_j and k internally vertex-disjoint directed paths from x_j to x_i. Let E_0 denote the set of the edges in all those paths.

3. Let G^+ be the graph constructed by adding to G a new vertex z and $2k$ new edges $(z, x_1), \ldots, (z, x_k)$ and $(x_1, z), \ldots, (x_k, z)$.

4. Use FIND-H to find a subgraph $H_1 = (V \cup \{z\}, E_1)$ for G^+ and z. Symmetrically, find a subgraph $H_2 = (V \cup \{z\}, E_2)$ for G^+ and z with respect to the *reverse* edge and path directions.

5. Let the final certificate F be $E_0 \cup E_1 \cup E_2$ without the edges incident with z.

The next two lemmas show the correctness of the above certificate algorithm.

LEMMA 4.4. *If G is k-vertex connected, then the subgraph (V, F) computed above is k-vertex connected.*

Proof. Let $C = (V, F)$. Let $C^+ = (V \cup \{z\}, E_0 \cup E_1 \cup E_2)$. Note that C^+ is the graph constructed by adding to C the vertex z and the $2k$ edges $(z, x_1), \ldots, (z, x_k)$ and $(x_1, z), \ldots, (x_k, z)$. Next, because $E_1 \cup E_2$ is included in C^+, by Lemma 4.3, $\kappa_{C^+}(z, v) \geq k$ and $\kappa_{C^+}(v, z) \geq k$ for all $v \in V$. Because E_0 is included in C, for all x_i and x_j with $i < j$, $\kappa_C(x_i, x_j) \geq k$ and $\kappa_C(x_j, x_i) \geq k$. Therefore, by the classic argument of Even for his k-vertex connectivity algorithm, (V, F) is indeed k-vertex connected. □

LEMMA 4.5. $|F| \leq 2k^2 n$.

Proof. If G is not k-vertex connected, then $|F| = 0$. Otherwise, the upper bound of $|F|$ follows from the facts that $|E_1| = |E_2| = kn$ and $|E_0| \leq k(k-1)(n-2+k)$. □

The next lemma finishes the proof of Theorem 4.1.

LEMMA 4.6. *The above certificate algorithm runs in $O(km \max\{n, k\sqrt{n}\})$ time.*

Proof. Testing k-vertex connectivity takes $O(km \max\{n, k\sqrt{n}\})$ time [14]. Computing E_0 takes $O(k^2 m \min\{k, \sqrt{n}\})$ time. By Lemma 4.3, computing E_1 and E_2 takes $O(kmn)$ time. These three steps dominate the time complexity. Therefore, the total running time is $O(km \max\{n, k\sqrt{n}\})$. □

5. Conclusions and open problems.

Designing graph search procedures that are efficient in several major models of computation is an important issue in algorithmic graph theory. We have shown that scan-first search can be performed extremely efficiently in the parallel, the distributed, and the sequential models. Based on this unusual efficiency, it is worth further research to find other applications for scan-first search.

We conclude with two open problems concerning graph connectivity. The first is whether the k-vertex connectivity of an n-vertex *directed* graph can be tested in $k^{O(1)}n^2$ sequential time. The second is whether a sparse certificate for the k-vertex connectivity of an n-vertex undirected graph can be found *deterministically* in time polylogarithmic *both* in k and n, using a number of processors polynomial both in k and n.

Acknowledgments. The authors wish to thank Martin Farach, Hillel Gazit, Samir Khuller, Sandeep Sen, and Éva Tardos for helpful discussions.

REFERENCES

[1] A. AGGARWAL AND R. J. ANDERSON, *A random* NC *algorithm for depth first search*, Combinatorica, 8 (1988), pp. 1–12.

[2] A. AGGARWAL, R. J. ANDERSON, AND M. Y. KAO, *Parallel depth-first search in general directed graphs*, SIAM J. Comput., 19 (1990), pp. 397–409; also appeared in the Proceedings of the 21st ACM Symposium on Theory of Computing, Seattle, WA, May 15–17, 1989, pp. 297–308.

[3] B. AWERBUCH AND D. PELEG, *Network synchronization with polylogarithmic overhead*, in Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science, 1990, Vol. II, pp. 514–522.

[4] C. BERGE, *Graphs*, second revised ed., North-Holland, New York, 1985.

[5] B. BOLLOBÁS, *Extremal Graph Theory*, Academic Press, London, 1978.

[6] A. Borodin, J. von zur Gathen, and J. Hopcroft, *Fast parallel matrix and gcd computations*, in Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 65–71.

[7] J. Cheriyan and S. N. Maheshwari, *Finding nonseparating induced cycles and independent spanning trees in 3-connected graphs*, J. Algorithms, 9 (1988), pp. 507–537.

[8] J. Cheriyan and R. Thurimella, *Finding sparse spanning subgraphs efficiently*, preprint, August 1990.

[9] ———, *On determining vertex connectivity*, Tech. Report UMIACS-TR-90-79 CS-TR-2485, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, June 1990.

[10] ———, *Algorithms for parallel k-vertex connectivity and sparse certificates*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 391–401.

[11] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491.

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

[13] S. Even, *An algorithm for determining whether the connectivity of a graph is at least k*, SIAM J. Comput., 4 (1975), pp. 393–396.

[14] ———, *Graph Algorithms*, Computer Science Press, New York, 1979.

[15] D. Fussell, V. Ramachandran, and R. Thurimella, *Finding triconnected components by local replacements*, in Proceedings of the 16th International Colloquium on Automata, Languages, and Programming, 1989, pp. 379–393; SIAM J. Comput., 22 (1993), to appear.

[16] D. Fussell and R. Thurimella, *Successive approximation in parallel graph algorithms*, Theoret. Comput. Sci., 74 (1990), pp. 19–35.

[17] Z. Galil, *Finding the vertex connectivity of graphs*, SIAM J. Comput., 9 (1980), pp. 197–199.

[18] H. Gazit, *An optimal randomized parallel algorithm for finding connected components in a graph*, Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pp. 492–501.

[19] H. Gazit and G. L. Miller, *An improved parallel algorithm that computes the* BFS *numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.

[20] V. Hadzilacos, *Connectivity requirements for byzantine agreement under restricted types of failures*, Distributed Comput., 2 (1987), pp. 95–103.

[21] A. Itai and M. Rodeh, *The multi-tree approach to reliability in distributed networks*, Inform. Comput., 79 (1988), pp. 43–59.

[22] A. Kanevsky and V. Ramachandran, *Improved algorithms for graph four-connectivity*, J. Comput. System Sci., 42 (1991), pp. 288–306.

[23] R. Karp and V. Ramachandran, *A survey of parallel algorithms for shared-memory machines*, Tech. Report. UCB/CSD 88/408, Computer Science Division, EECS, University of California at Berkeley, March 1988; in the Handbook of Theoretical Computer Science, North-Holland, Amsterdam, to appear.

[24] S. Khuller and B. Schieber, *Efficient parallel algorithms for testing connectivity and finding disjoint s-t paths in graphs*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 288–293.

[25] W. Mader, *Minimal n-fach zusammenhängende Digraphen*, J. Combin. Theory, B38 (1985), pp. 102–117.

[26] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–114.

[27] H. Nagamochi and T. Ibaraki, *Linear time algorithms for finding k-edge-connected and k-node-connected spanning subgraphs*, Algorithmica, 7 (1992), pp. 583–596.

[28] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 175–202, February 1993

DECOMPOSING FINITE-VALUED TRANSDUCERS AND DECIDING THEIR EQUIVALENCE*

ANDREAS WEBER†

Abstract. In this paper finite-valued finite transducers are investigated in connection with their inner structure. The following results are shown: A finite-valued nondeterministic generalized sequential machine (NGSM) M can be effectively decomposed into finitely many single-valued NGSMs $M_1, \dots, M_N$ such that the transduction realized by M is the union of the transductions realized by $M_1, \dots, M_N$. Using this decomposition, the equivalence of finite-valued NGSMs is decidable in deterministic double exponential time. By reduction, both results can be generalized to normalized finite transducers.

Key words. finite transducer, valuedness, finite valued, equivalence problem

AMS(MOS) subject classification. 68Q75

Introduction. A finite transducer M is a finite automaton with output. Informally, M may be regarded as a finite directed labeled graph. The vertices and edges of that graph represent the states and transitions of M, respectively. The label of an edge is the pair of input/output words consumed/produced by the corresponding transition of M. As usual, these input words are required to have length 0 or 1. The resulting nondeterministic model for M is the normalized finite transducer (NFT) — or the nondeterministic generalized sequential machine (NGSM) if only length 1 appears. The successful computations in M are represented by those paths (called accepting) in the above graph that lead from an initial to a final state (i.e., vertex). Along the edges of such a path an input word x is "transduced" into an ouput word, which is called a value for x in M. The valuedness of M is the maximal number of different values for an input word, or is infinite, depending on whether or not a maximum exists. In the former case, M is called finite valued. M is called single valued if its valuedness is at most 1. Two finite transducers are called equivalent if the transductions (or relations) realized by them coincide, i.e., if every input word has the same set of values in both transducers.

The valuedness is a structural parameter of the transduction realized by a finite transducer that received attention in connection with the equivalence problem. This problem is undecidable for NGSMs (see [G68], [I78], [L79b], and [L83]) but decidable for finite-valued NFTs [CK86]. The equivalence problem for single-valued NFTs is PSPACE-complete (see [Sch76], [BH77], [AL78], [GI83], and [HW91]). It is decidable in polynomial time whether or not an NFT is finite valued (see [W87], [W90]). Given any fixed integer d, it can be tested in polynomial time whether or not the valuedness of an NFT is greater than d [GI83]. For further background on finite transducers we refer to Berstel's textbook [B79] and to excellent survey papers by Karhumäki (see [K86], [K87]), and Culik [C90].

The basic result of this paper is the following decomposition theorem (see §2).

(1) A finite-valued NGSM M can be effectively decomposed into finitely many single-valued NGSMs $M_1, \dots, M_N$ such that the transduction realized by M is the union of the transductions realized by $M_1, \dots, M_N$.

*Received by the editors April 2, 1990; accepted for publication (in revised form) August 22, 1991. A preliminary version of this paper appeared under the title "A decomposition theorem for finite-valued transducers and an application to the equivalence problem" in the Proceedings of MFCS 1988. A full revision was done while the author was supported by a Postdoctoral Fellowship of the Japan Society for the Promotion of Science.

†Fachbereich Informatik, Johann Wolfgang Goethe-Universität, Postfach 111 932, W-6000 Frankfurt am Main 11, Germany.

In this theorem, N is of exponential order, which is optimal in certain cases. Each M_i has double exponential size and can be constructed in double exponential time. Intuitively, the theorem (1) states that a "difficult" (i.e., finite-valued) NGSM M is equivalent to some effectively constructible "disjoint union" of "easy" (i.e., single-valued) NGSMs $M_1, \ldots, M_N$. We want to point out that the main problem for the machines $M_1, \ldots, M_N$ is that the model of a "disjoint union" does not allow any communication among them. Given an input word x, each M_i has to decide autonomously which of the values for x in M it should produce as its own value. The theorem (1) directly implies two fundamental results of [W87] and [W90]; that is, an upper bound for the valuedness of a finite-valued NGSM and a characterization of the infinite valuedness of an NGSM, which can be tested in polynomial time (see §2).

In §3 we present a combinatorial word lemma. This lemma allows to use the above decomposition theorem in order to decide the equivalence of finite-valued NGSMs. Our results are as follows.

(2) The noninclusion of transductions realized by finite-valued NGSMs (and, hence, the nonequivalence of finite-valued NGSMs) can be detected by a witness of double exponential length.

(3) The inclusion of transductions realized by finite-valued NGSMs (and, hence, the equivalence of finite-valued NGSMs) is decidable in deterministic double exponential time.

Note that (2) alone implies a deterministic triple exponential time algorithm for (3). The witness length in (2) is necessarily of exponential order [N79] and the decision problems in (3) are PSPACE-hard [GJ79], even if the NGSMs in question are single valued and have only one output symbol. It is an open problem regarding whether or not the upper bounds for the witness length in (2) and for the time complexity in (3) can be improved to bounds of exponential order (see §3).

By reduction, the results (1)–(3) can be generalized to NFTs (see §4).

A noneffective version of (1) was stated in [Sch76] without complete proof. The methods used here differ from those in [Sch76]. Another attempt to establish a proof of (1) and a (nonefficient) decision procedure for (3) failed because of a fatal proof error (see [AL78], [L79a]). Our algorithm underlying (3) is quite different from the procedure by Culik and Karhumäki [CK86] deciding the equivalence of finite-valued NFTs. The latter procedure uses an algorithm by Makanin as a "subroutine"; it is proved to be recursive by employing a generalization of the (recently confirmed) Ehrenfeucht Conjecture. As far as I know, a time analysis for this procedure does not exist.

The above-mentioned word lemma generalizes a lemma by Schützenberger [Sch76] and may be of own interest. From similar work by Lisovik (see [L79a], [L80]) it can be obtained that the theorem (1) and Schützenberger's lemma imply (2). In order to derive (2) and (3) from (1) the above-mentioned word lemma can be replaced by a "machine-oriented" approach due to Gurari and Ibarra [GI81] or by an "algebraic" approach due to Turakainen (see [T88], [T89], and §3).

Our proof of (1) is based on a classification of accepting paths. To each accepting path in an NGSM M we attach a set of specifications. Given any such specification (altogether there are exponentially many), we effectively construct an NGSM realizing nothing but all pairs of words which appear as input/output along those accepting paths in M, that have the given specification. If M is finite valued, then the thus constructed NGSMs turn out to be single valued, and the union of the transductions realized by them coincides with the transduction realized by M. In order to prove this, we work on the basis of methods and technical lemmas presented in [W87] and [W90].

In a finite transducer, instead of counting different values it is quite natural to consider only their lengths. This leads from the valuedness to another parameter called length-degree [W92a]. Indeed, variants of (1)–(3) hold true when the valuedness of an NGSM is replaced by its length-degree [W92a]. In particular, the equivalence of NFTs with finite length-degree is recursively decidable. Another result similar to (1) is published in [W92b].

The work presented in this paper and a part of its sequel as described above was recently generalized by Seidl to bottom-up finite state tree transducers [S90].

1. Definitions and notations.

1.1. General notations. We use the following notations: $\mathcal{N}$, $\mathcal{Z}$, and $\mathcal{Q}$ denote the sets of all nonnegative integers, integers, and rationals, respectively. $\mathbb{N}$ and $[m]$ denote the sets $\mathcal{N}\backslash\{0\}$ and $\{1,\dots,m\}$, respectively, $[i,j]$ denotes the set $\{t \in \mathcal{Z} \mid i \le t \le j\}$ ($m \in \mathcal{N}, i,j \in \mathcal{Z}$). Let Δ be some nonempty, finite set. $\Delta^{\le l}$ denotes the set of all words in Δ^* having length at most l ($l \in \mathcal{N}$). Let $z \in \Delta^*$ and $j \in [|z|]$, then $z(j) \in \Delta$ denotes the jth letter of z. Let $z_1, z_2 \in \Delta^*$ and $j \in [\min\{|z_1|, |z_2|\}]$. We say that z_1 and z_2 *differ at position* j if $z_1(j)$ and $z_2(j)$ are distinct. Without further mention we assume that the model of computation for all our algorithms is the deterministic random access machine (RAM) without multiplications and divisions using the uniform cost criterion (see, e.g., [AHU74]).

1.2. Finite transducers. Our model of a finite transducer is the *normalized finite transducer* (NFT). Formally, an NFT is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ where Q, Σ, and Δ denote nonempty, finite sets of states, input symbols, and output symbols, respectively, $Q_I, Q_F \subseteq Q$ denote sets of initial and final (or accepting) states, respectively, and δ is a finite subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times \Delta^* \times Q$. Σ (Δ) is called the input (output) alphabet, δ is called the transition relation. Each element of δ denotes a *transition*. Of course, M is *nondeterministic*. M is called a *nondeterministic generalized sequential machine* (NGSM) if δ is a finite subset of $Q \times \Sigma \times \Delta^* \times Q$. In this paper we mainly deal with NGSMs.

The mode of operation of M is described by paths. A *path* π (of length m) is a word

$$(q_1, x_1, z_1) \dots (q_m, x_m, z_m) q_{m+1} \in (Q \times (\Sigma \cup \{\varepsilon\}) \times \Delta^*)^m \cdot Q$$

so that $(q_1, x_1, z_1, q_2), \dots, (q_m, x_m, z_m, q_{m+1})$ are transitions. π is said to lead from q_1 to q_{m+1}, to consume $x := x_1 \dots x_m \in \Sigma^*$, to produce $z := z_1 \dots z_m \in \Delta^*$, and to realize $(x, z) \in \Sigma^* \times \Delta^*$. π is called *accepting* if $q_1 \in Q_I$ and $q_{m+1} \in Q_F$. Whenever convenient we identify a transition (p, a, z, q) with the path $(p, a, z)q$ of length 1 and vice versa. We define $\hat{\delta} := \{(p, x, z, q) \in Q \times \Sigma^* \times \Delta^* \times Q \mid (x, z)$ is realized by some path in M leading from p to $q\}$. If M is an NGSM, then δ equals $\hat{\delta} \cap Q \times \Sigma \times \Delta^* \times Q$. In this case we rename $\hat{\delta}$ by δ. Let $\pi_1 = \pi_1' q_1$ and $\pi_2 = \pi_2' q_2$ be paths in M leading from p_1 to q_1 and from p_2 to q_2, respectively. If q_1 and p_2 coincide, then we define the path $\pi_1 \circ \pi_2 := \pi_1' \pi_2' q_2$. Note that the operation "$\circ$" on paths is associative.

The *transduction* (or *relation*) *realized* by M, denoted by $T(M)$, is the set of pairs (in $\Sigma^* \times \Delta^*$) realized by all accepting paths in M. The *language recognized* by M, denoted by $L(M)$, is the domain of $T(M)$, i.e., the set of words (in Σ^*) consumed by all accepting paths in M. Two NFTs are *equivalent* if the transductions realized by them coincide.

If $(x, z) \in \Sigma^* \times \Delta^*$ belongs to $T(M)$, then z is called a *value* for x in M. The *valuedness* of $x \in \Sigma^*$ in M (short form: $\mathrm{val}_M(x)$) is the number of all different values for x in M. The *valuedness* of M (short form: $\mathrm{val}(M)$) is the supremum of the set $\{\mathrm{val}_M(x) \mid x \in \Sigma^*\}$. Note that, for a given $x \in \Sigma^*$, $\mathrm{val}_M(x)$ may be infinite (see §4), whereas it is clearly finite if M is an NGSM. M is called *infinite valued* (*finite valued*, *single valued*) if its valuedness is infinite (finite, at most 1, respectively).

A state of M is called *useful* if it appears on some accepting path; otherwise, this state is called *useless*. Useless states are irrelevant to the valuedness in M. If all states of M are useful, then M is called *trim*. A state p of M is said to be *connected* with a state q of M (short form: $p \overset{M}{\longleftrightarrow} q$) if some paths lead from p to q and from q to p. An equivalence class with respect to the relation "$\overset{M}{\longleftrightarrow}$" is called a *strong component* of M.

Let $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ be another NFT. We define some local structural parameters of M and M_0: $\text{val}(\delta)$ is defined as the minimal $k_1 \geq 1$ such that, for all $(p, a, q) \in Q \times (\Sigma \cup \{\varepsilon\}) \times Q$, $\#\{z \in \Delta^* \mid (p, a, z, q) \in \delta\}$ is at most k_1. $\text{diff}(\delta, \delta_0)$ is defined as the minimal $k_2 \geq 0$ such that, for all pairs $((p, a, z, q), (\tilde{p}, a, \tilde{z}, \tilde{q}))$ of transitions in M and M_0 consuming the same $a \in \Sigma \cup \{\varepsilon\}$, $||z| - |\tilde{z}||$ is at most k_2. We set $\text{diff}(\delta) := \text{diff}(\delta, \delta)$. $\text{im}(\delta)$ is the set of ε and of all words $z \in \Delta^*$ produced by any transition of M. We set $\text{iml}(\delta) := \max\{|z| \mid z \in \text{im}(\delta)\}$.[1]

The *size* of δ, denoted by $\|\delta\|$, is defined as 1 plus the sum of $1 + |z|$ over all transitions (p, a, z, q) of M. The *size* of M, denoted by $\|M\|$, is defined as the sum of $\#Q$, $\#\Sigma$, $\#\Delta$, and $\|\delta\|$.

Note. $\text{val}(\delta) \leq \#\text{im}(\delta) \leq \min\{\|\delta\|, \#(\Delta^{\leq \text{iml}(\delta)})\}$, $\text{diff}(\delta, \delta_0) \leq \max\{\text{iml}(\delta), \text{iml}(\delta_0)\}$, and $\text{diff}(\delta) \leq \text{iml}(\delta) \leq \|\delta\| - 1$.

In the rest of this subsection we assume that $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ is an NGSM.

Let $x = x_1 \dots x_m \in \Sigma^*$ $(x_1, \dots, x_m \in \Sigma)$. The *graph of accepting paths* in M consuming x (short form: $G_M(x)$) is the directed graph (V, E) where

$$\begin{aligned} V &:= \{(q, j) \in Q \times \{0, \dots, m\} \mid \exists q_I \in Q_I \, \exists q_F \in Q_F \, \exists z_1, z_2 \in \Delta^* : \\ &\qquad (q_I, x_1 \dots x_j, z_1, q) \in \delta \;\;\&\;\; (q, x_{j+1} \dots x_m, z_2, q_F) \in \delta\}, \quad \text{and} \\ E &:= \{((p, j-1), (q, j)) \in V^2 \mid j \in [m] \;\;\&\;\; \exists z \in \Delta^* : (p, x_j, z, q) \in \delta\}. \end{aligned}$$

Note that the set of all paths in $G_M(x)$ leading from $Q_I \times \{0\}$ to $Q_F \times \{m\}$ corresponds to the set of all accepting paths in M consuming x projected to their Q- and Σ-components. Each vertex of $G_M(x)$ is situated on such a path.

Let $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ be another NGSM. Let $x = x_1 \dots x_m \in \Sigma^*$ $(x_1, \dots, x_m \in \Sigma)$. Let

$$\pi = (q_1, x_1, z_1) \dots (q_m, x_m, z_m) q_{m+1} \quad \text{and} \quad \tilde{\pi} = (\tilde{q}_1, x_1, \tilde{z}_1) \dots (\tilde{q}_m, x_m, \tilde{z}_m) \tilde{q}_{m+1}$$

be paths in M and M_0, respectively, both consuming x. We define

$$\text{diff}\,(\pi, \tilde{\pi}) := \max\{||z_1 \dots z_l| - |\tilde{z}_1 \dots \tilde{z}_l|| \mid 0 \leq l \leq m\}.$$

Note that $\text{diff}\,(\pi, \tilde{\pi})$ is at most $m \cdot \text{diff}(\delta, \delta_0)$.

1.3. Criteria for finite transducers. Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be an NFT. The following criteria (IV1) and (IV2) were introduced in [W87] and [W90][2] in order to characterize the infinite valuedness of M.

(IV1) There are useful states $p, q_1, q_2, q_3, q \in Q$ such that, for some words $u, v, w \in \Sigma^*$ and $\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{v}_1, \tilde{v}_2, \tilde{v}_3, \tilde{w}_1, \tilde{w}_2, \tilde{w}_3 \in \Delta^*$, the following holds: $\tilde{v}_1$ and $\tilde{v}_2$ have distinct lengths, $(p, u, \tilde{u}_1, q_1), (q_1, v, \tilde{v}_1, q_1), (q_1, w, \tilde{w}_1, p) \in \hat{\delta}$, $(p, u, \tilde{u}_2, q_2)$, $(q_2, v, \tilde{v}_2, q_2), (q_2, w, \tilde{w}_2, q) \in \hat{\delta}$, and $(q, u, \tilde{u}_3, q_3), (q_3, v, \tilde{v}_3, q_3), (q_3, w, \tilde{w}_3, q) \in \hat{\delta}$.

[1] $\text{iml}(\delta)$ is denoted by $\text{length}(\delta)$ in [W87] and [W90].

[2] Note that (IV1) appears as (IV3) in [W87] and as (IV1) in [W90], whereas (IV2) slightly differs from (IV2) in [W87], appears as (IV2)′ in [W90], and is equivalent to (IV2) there [W90, Lem. 2.3(i)].

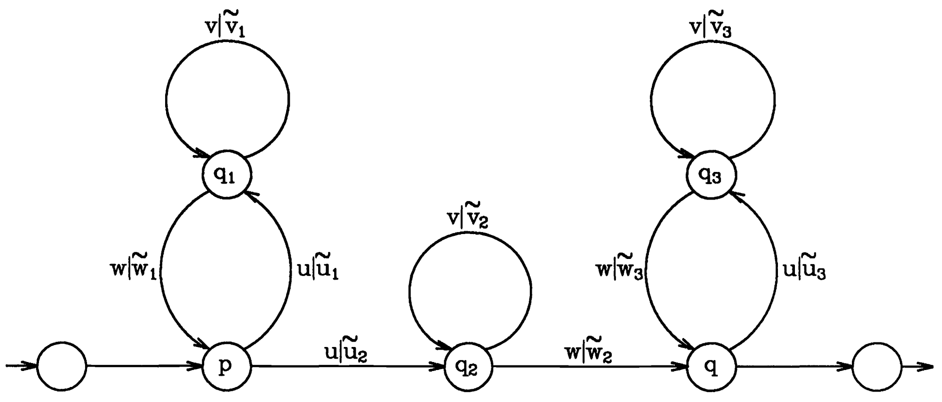


FIG. 1

(IV2) There are useful states $p, q \in Q$ such that, for some words $v \in \Sigma^*$, $\tilde{v}_1 \in \Delta^+$ and $\tilde{v}_2, \tilde{v}_3, \tilde{v}_4 \in \Delta^*$, $\tilde{v}_2$ and $\tilde{v}_3$ differ at some position $j \in [\min\{|\tilde{v}_2|, |\tilde{v}_3|\}]$, and $(p, v, \tilde{v}_1, p), (p, v, \tilde{v}_2, q), (p, v, \tilde{v}_3, q), (q, v, \tilde{v}_4, q) \in \hat{\delta}$.

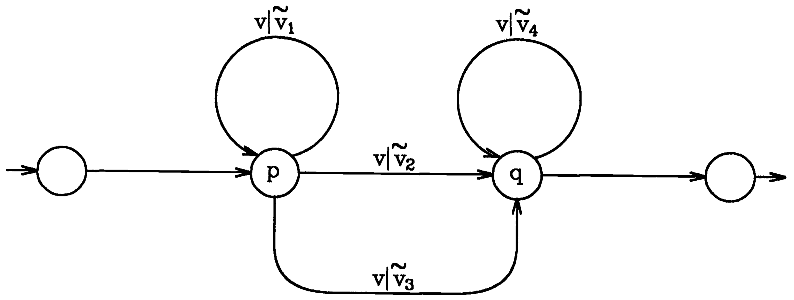


FIG. 2

1.4. Finite automata. Our model of a finite automaton is the *nondeterministic finite automaton with ε-moves* (ε-NFA). Formally, an ε-NFA is a 5-tuple $M = (Q, \Sigma, \delta, Q_I, Q_F)$ whose components Q, Σ, δ, Q_I, and Q_F have the same meaning as for an NFT $(Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ except that δ is a subset of $Q \times (\Sigma \cup \{\varepsilon\}) \times Q$. Each element of δ denotes a *transition*. M is called a *nondeterministic finite automaton* (NFA) if δ is a subset of $Q \times \Sigma \times Q$. M is called a *deterministic finite automaton* (DFA) if it is an NFA such that $\#Q_I = 1$ and $(p, a, q_1), (p, a, q_2) \in \delta$ implies $q_1 = q_2$.

A *path* π (of length m) in M is a word $(q_1, x_1) \dots (q_m, x_m) q_{m+1} \in (Q \times (\Sigma \cup \{\varepsilon\}))^m \cdot Q$ so that $(q_1, x_1, q_2), \dots, (q_m, x_m, q_{m+1})$ are transitions. π is said to lead from q_1 to q_{m+1} and to consume $x := x_1 \dots x_m \in \Sigma^*$. π is called *accepting* if $q_1 \in Q_I$ and $q_{m+1} \in Q_F$. We define $\hat{\delta} := \{(p, x, q) \in Q \times \Sigma^* \times Q \mid x$ is consumed by some path in M leading from p to $q\}$. If M is an NFA, then δ equals $\hat{\delta} \cap Q \times \Sigma \times Q$. In this case we rename $\hat{\delta}$ by δ. Let $\pi_1 = \pi_1' q_1$ and $\pi_2 = \pi_2' q_2$ be paths in M leading from p_1 to q_1 and from p_2 to q_2, respectively. If q_1 and p_2 coincide, then we define the path $\pi_1 \circ \pi_2 := \pi_1'\pi_2'q_2$. Note that the operation "$\circ$" on paths is associative.

The *language recognized* by M, denoted by $L(M)$, is the set of words consumed by all accepting paths in M. Two ε-NFAs are *equivalent* if the languages recognized by them coincide.

The *size* of M, denoted by $\|M\|$, is defined as the sum of $\#Q$, $\#\Sigma$, and $1 + \#\delta$.

2. Decomposing finite-valued NGSMs. In this section we prove the following decomposition theorem.

THEOREM 2.1. *Let M be an* NGSM, *which does not comply with any of the criteria* (IV1), (IV2). *Then, $O(2^{\mathrm{poly}\|M\|})$ many single-valued* NGSMs *$M_1, \ldots, M_N$ — each of size $O(2^{2^{\mathrm{lin}\|M\|}})$ — effectively exist such that $T(M)$ equals $T(M_1) \cup \ldots \cup T(M_N)$. In detail, let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ and $n := \#Q$, then N is at most*

$$\begin{cases} (5^{1/2} \cdot 3^{2/3})^n \cdot n^{4\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+3)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{diff}(\delta)} & \text{if } \#\Delta > 1, \\ (5^{1/2} \cdot 3^{4/3})^n \cdot n^{6\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+2)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1, \end{cases}$$

and each M_i ($i \in [N]$) has at most $2^{n\cdot(2^{n+1}+12)} \cdot (1 + \mathrm{diff}(\delta)) \cdot (2 + \mathrm{iml}(\delta))$ states and size at most $2^{n\cdot(2^{n+2}+24)} \cdot \|M\|^5$.

Using a variant of the construction for Theorem 2.1 we obtain the following theorem.

THEOREM 2.2. *Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$, n, N and $M_1, \ldots, M_N$ be as in Theorem 2.1. Then,* NFAs *$M_1', \ldots, M_N'$ — each of size $O(2^{2^{\mathrm{lin}\|M\|}})$ — effectively exist such that, for all $i \in [N]$, M_i' recognizes $\Sigma^* \backslash L(M_i)$. In detail, each M_i' ($i \in [N]$) has at most $2^{n\cdot(2^{n+1}+8)} \cdot 2^{2n^8\cdot(1+\mathrm{diff}(\delta))\cdot(2+\mathrm{iml}(\delta))}$ states (and more states than M_i) and size at most $2^{n\cdot(2^{n+2}+17)} \cdot 2^{5n^8\cdot\|M\|^2}$.*

A time analysis of the constructions for the Theorems 2.1 and 2.2 yields:

THEOREM 2.3. *Let M be an* NGSM *as in the Theorems* 2.1 *and* 2.2. *The* NGSMs *$M_1, \ldots, M_N$ and the* NFAs *$M_1', \ldots, M_N'$ of these theorems can be constructed in* DTIME $(2^{2^{\mathrm{lin}\|M\|}})$.

According to Lemma A.1 in the Appendix, the Theorems 2.1–2.3 apply to finite-valued NGSMs as well. Intuitively, Theorem 2.1 therefore states that a finite-valued NGSM M can be effectively decomposed into finitely many single-valued NGSMs. By reduction, Theorem 2.1 can be generalized to NFTs (Theorem 4.1).

Let M be an NGSM as in the Theorems 2.1–2.3. It turns out that the quite large and a little complicated NGSMs $M_1, \ldots, M_N$ and NFAs $M_1', \ldots, M_N'$ constructed for M in these theorems, while consuming the same word, need the greater part of their capability in order to carry out exactly the same "basic work." Intuitively spoken these machines are doing so because the model of a "disjoint union" of finite transducers does not allow any communication among them. The author believes that this missing communication is the main reason why $M_1, \ldots, M_N$ and $M_1', \ldots, M_N'$ are so large and complicated.

In §3 we use the Theorems 2.1–2.3 in order to decide the equivalence of finite-valued NGSMs. Here, using Lemma A.1, we state the following immediate corollaries of Theorem 2.1.

COROLLARY 2.4. *Let M be an* NGSM. *M is finite valued if and only if finitely many single-valued* NGSMs *$M_1, \ldots, M_N$ effectively exist such that $T(M)$ equals $T(M_1) \cup \ldots \cup T(M_N)$.*

COROLLARY 2.5 (see [W90, Thm. 2.1]). *Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be a finite-valued* NGSM *with n states. Then, the valuedness of M is not greater than the upper bound for N stated in Theorem* 2.1.

COROLLARY 2.6 [W90, Thm. 2.2]. *Let M be an* NGSM. *M is infinite-valued if and only if it complies with at least one of the criteria* (IV1), (IV2).

Note that, unless $n = 1$, the upper bound for the valuedness of a finite-valued NGSM stated in [W90, Thm. 2.1] is smaller than the upper bound in Corollary 2.5 and the upper bound for N in Theorem 2.1. The Theorems 4.1 and 4.2 of [W90] imply that each improvement of any of these bounds has to stop above $\#\Delta^{n^3 \cdot \mathrm{diff}(\delta)/343}$ if $\#\Delta > 1$ and at 2^{n-1} if $\#\Delta = 1$. Thus, if $\#\Delta > 1$ and $\mathrm{diff}(\delta) > 0$, all these bounds are asymptotically optimal apart from a constant factor in the exponent.

It is decidable in polynomial time whether or not an NGSM complies with (IV1) or (IV2) (see [W90, §3]). Therefore, Corollary 2.6 implies: It is decidable in polynomial time whether or not an NGSM is infinite valued [W90, Thm. 3.1].

In the rest of this section we prove the Theorems 2.1–2.3, successively.

Proof of Theorem 2.1. Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be an NGSM with n states, which does not comply with any of the criteria (IV1), (IV2). We may assume that M is trim. The set of accepting paths in M is denoted by Π. Let $Q_1, \ldots, Q_k$ be an order of the strong components of M so that for all $i, j \in [k]$ the following holds.

$$\delta \cap Q_i \times \Sigma^* \times \Delta^* \times Q_j \neq \emptyset \quad \Longrightarrow \quad i \leq j.$$

Let $n_i := \#Q_i$ $(i = 1, \ldots, k)$.

Our proof of Theorem 2.1 consists of the following steps.

(1) We define a set S of potential path specifications having cardinality at most

$$\begin{cases} 5^{n/2} \cdot 3^{2n/3} \cdot 2^{(n+3)\cdot(n-1)} \cdot n^{4\cdot(n-1)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{diff}(\delta)} & \text{if } \#\Delta > 1, \\ 5^{n/2} \cdot 3^{4n/3} \cdot 2^{(n+2)\cdot(n-1)} \cdot n^{6\cdot(n-1)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1. \end{cases}$$

(2) We define a mapping $\varphi : \Pi \longrightarrow 2^S \backslash \{\emptyset\}$, such that every $\sigma \in \varphi(\pi)$ acts as a specification of the path $\pi \in \Pi$, and the following holds: If $\pi, \pi' \in \Pi$ realize $(x, z), (x, z') \in \Sigma^* \times \Delta^*$, respectively, and if $\varphi(\pi) \cap \varphi(\pi')$ is nonempty, then z and z' coincide.

(3) For all $\sigma \in S$ we effectively construct an NGSM $\bar{M}_\sigma$ realizing the transduction $\{(x, z) \in \Sigma^* \times \Delta^* \mid (x, z) \text{ is realized by some } \pi \in \Pi \text{ so that } \sigma \in \varphi(\pi)\}$. Each $\bar{M}_\sigma$ $(\sigma \in S)$ has at most $2^{n\cdot(2^{n+1}+12)} \cdot (1 + \mathrm{diff}(\delta)) \cdot (2 + \mathrm{iml}(\delta))$ states and size at most $2^{n\cdot(2^{n+2}+24)} \cdot \|M\|^5$.

From (1)–(3) it follows that $T(M)$ equals $\bigcup_{\sigma \in S} T(\bar{M}_\sigma)$, and that each $\bar{M}_\sigma$ $(\sigma \in S)$ is single valued. Therefore, the steps (1)–(3) prove the theorem. Note that $N = \#S$ and the NGSMs $\bar{M}_\sigma$ $(\sigma \in S)$ are playing the role of $M_1, \ldots, M_N$.

In order to carry out step (1) we need the following proposition.

PROPOSITION 2.7. *Let $n = \sum_{i=1}^k n_i$, where $n_1, \ldots, n_k \in \mathcal{N}$. Then, $\prod_{i=1}^k n_i \leq 3^{n/3}$ and $\prod_{i=1}^k (n_i^2 + 1) \leq 5^{n/2}$.*

Proof. It is easy to show by induction on j that, for each $j \in \mathcal{N}$, $j^3 \leq 3^j$ and $(j^2+1)^2 \leq 5^j$. In turn, this implies: $\prod_{i=1}^k n_i \leq \prod_{i=1}^k 3^{n_i/3} = 3^{n/3}$ & $\prod_{i=1}^k (n_i^2+1) \leq \prod_{i=1}^k 5^{n_i/2} = 5^{n/2}$. $\square$

Execution of step (1). We define a set S of potential path specifications:

$$S := \bigcup_{l\geq 0} \bigcup_{1\leq i_0<\ldots<i_l\leq k} \bigcup_{0<j_1<\ldots<j_l<2^{n+1}} (Q_I \cap Q_{i_0}) \times$$
$$\prod_{\lambda=1}^{l} [\{j_\lambda\} \times Q_{i_{\lambda-1}} \times (\mathrm{im}(\delta) \cup \{z \in \Delta^* \mid |z| \leq (n^2 \cdot \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i - 1) \cdot \mathrm{diff}(\delta)\} \cup$$
$$\{(t,\tilde{p},\tilde{q}) \in \mathcal{Z} \times Q^2 \mid |t| \leq (n^2 \cdot \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i - 1) \cdot \mathrm{diff}(\delta) \ \ \& \ \ \tilde{p} \overset{\mathrm{M}}{\longleftrightarrow} \tilde{q}\}) \times Q_{i_\lambda}]$$
$$\times (Q_F \cap Q_{i_l}).$$

Using Proposition 2.7, we can prove the announced upper bound for the cardinality of S as follows.

$$\begin{aligned}
\#S &= \sum_{l\geq 0} \sum_{1\leq i_0<\ldots<i_l\leq k} \sum_{0<j_1<\ldots<j_l<2^{n+1}} \\
&\quad \#(Q_I \cap Q_{i_0}) \cdot \prod_{\lambda=1}^{l} [n_{i_{\lambda-1}} \cdot (\#(\mathrm{im}(\delta) \cup \Delta^{\leq(n^2\cdot\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i-1)\cdot\mathrm{diff}(\delta)}) \\
&\quad +(2 \cdot (n^2 \cdot \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i - 1) \cdot \mathrm{diff}(\delta) + 1) \cdot \sum_{i=1}^{k} n_i^2) \cdot n_{i_\lambda}] \cdot \#(Q_F \cap Q_{i_l}) \\
&\leq \sum_{l\geq 0} \sum_{1\leq i_0<\ldots<i_l\leq k} \sum_{0<j_1<\ldots<j_l<2^{n+1}} (\prod_{\lambda=0}^{l} n_{i_\lambda}^2) \cdot \\
&\quad \prod_{\lambda=1}^{l} [\#\mathrm{im}(\delta) \cdot \#\Delta^{\leq(n^2\cdot\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i-1)\cdot\mathrm{diff}(\delta)} \cdot 2n^4 \cdot (\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i) \cdot (1+\mathrm{diff}(\delta))] \\
&\leq \sum_{l\geq 0} \sum_{1\leq i_0<\ldots<i_l\leq k} \\
&\quad [2^{(n+1)\cdot l} \cdot (\prod_{\lambda=0}^{l} n_{i_\lambda}^2) \cdot (\#\mathrm{im}(\delta))^l \cdot 2^l \cdot n^{4l} \cdot (\prod_{\lambda=1}^{l} (\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i)) \cdot (1+\mathrm{diff}(\delta))^l \\
&\quad \cdot \begin{cases} (\prod_{\lambda=1}^{l} 2 \cdot \#\Delta^{(n^2\cdot\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i-1)\cdot\mathrm{diff}(\delta)})] & \text{if } \#\Delta > 1, \\ (\prod_{\lambda=1}^{l} n^2 \cdot (\sum_{i=i_{\lambda-1}}^{i_\lambda} n_i) \cdot (1+\mathrm{diff}(\delta)))] & \text{if } \#\Delta = 1 \end{cases} \\
&\leq 5^{n/2} \cdot 2^{(n+1)\cdot(n-1)} \cdot (\#\mathrm{im}(\delta))^{n-1} \cdot 2^{n-1} \cdot n^{4\cdot(n-1)} \cdot 3^{2n/3} \cdot (1+\mathrm{diff}(\delta))^{n-1} \\
&\quad \cdot \begin{cases} 2^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{diff}(\delta)} & \text{if } \#\Delta > 1, \\ n^{2\cdot(n-1)} \cdot 3^{2n/3} \cdot (1+\mathrm{diff}(\delta))^{n-1} & \text{if } \#\Delta = 1 \end{cases} \\
&\leq \begin{cases} 5^{n/2} \cdot 3^{2n/3} \cdot 2^{(n+3)\cdot(n-1)} \cdot n^{4\cdot(n-1)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1+\mathrm{diff}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{diff}(\delta)} & \text{if } \#\Delta > 1, \\ 5^{n/2} \cdot 3^{4n/3} \cdot 2^{(n+2)\cdot(n-1)} \cdot n^{6\cdot(n-1)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1+\mathrm{diff}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1. \end{cases}
\end{aligned}$$

Note. Proposition 2.7 implies

$$\sum_{l\geq 0}\ \sum_{1\leq i_0<\ldots<i_l\leq k}\ \prod_{\lambda=0}^{l} n_{i_\lambda}^2 = \prod_{i=1}^{k}(n_i^2+1)-1 \leq 5^{n/2}-1.$$

Execution of step (2). Let $x = x_1 \ldots x_m \in \Sigma^*$ $(x_1, \ldots, x_m \in \Sigma)$. Consider $G_M(x) = (V, E)$. Let $\mu \in \{0, \ldots, m\}$. We define $\text{set}(x, \mu) := \{q \in Q \,|\, (q, \mu) \in V\}$. $\text{set}(x, \mu)$ denotes the set of states at column μ in $G_M(x)$. Consider the uniquely determined $A_1, \ldots, A_{d+1} \in 2^Q$ and $y_1, \ldots, y_d \in \Sigma^*$ such that $x = y_1 \ldots y_d$, d is odd and the following holds (see Fig. 3).

$$\begin{array}{rll} \forall j = 1, \ldots, d+1: & A_j = \text{set}(x, |y_1 \cdots y_{j-1}|) & \& \\ \forall j = 1, \ldots, d: & j \text{ odd} \implies & \\ & |(y_1 \cdots y_{j-1})y_j| = \max\{0 \leq \mu \leq m \,|\, A_j = \text{set}(x, \mu)\} & \& \\ \forall j = 1, \ldots, d: & j \text{ even} \implies y_j \in \Sigma. & \end{array}$$

Thus, for each odd $j \in [d]$, A_{j+1} is the "last occurrence" of A_j in the sequence $\text{set}(x, 0)$, $\text{set}(x, 1), \ldots, \text{set}(x, m)$. Clearly, $d \leq 2^{n+1} - 1$. *Note*: If $x \notin L(M)$, then $d = 1$ and $A_1 = A_2 = \emptyset$.

Assume that $\pi \in \Pi$ is an accepting path consuming x and producing some $z \in \Delta^*$. We shall define the set $\varphi(\pi) \subseteq S$ of specifications of π, and we shall show that this set is nonempty.

Consider the uniquely determined paths $\pi_1, \ldots, \pi_d$ and the uniquely determined $z_1, \ldots, z_d \in \Delta^*$ and $p_1', q_1', \ldots, p_d', q_d' \in Q$ such that $\pi = \pi_1 \circ \ldots \circ \pi_d$ and, for each $j = 1, \ldots, d$, π_j realizes (y_j, z_j) and leads from p_j' to q_j' (see Fig. 3). By construction, $z = z_1 \ldots z_d$, $p_1' \in A_1 \subseteq Q_I$, $q_{j-1}' = p_j' \in A_j$ $(j = 2, \ldots, d)$, $q_d' \in A_{d+1} \subseteq Q_F$, and $\{z_j \,|\, j \in [d], \text{j even}\} \subseteq \text{im}(\delta)$. We define

$$J := \{j \in [d] \,|\, \neg(p_j' \xleftrightarrow{\text{M}} q_j')\}.$$

Let $l \in \{0, \ldots, k-1\}$ and $1 \leq j_1 < \ldots < j_l \leq d$ so that $J = \{j_1, \ldots, j_l\}$. Let $1 \leq i_0 < i_1 < \ldots < i_l \leq k$ so that $p_1' \in Q_{i_0}$, $p_{j_\lambda}' \in Q_{i_{\lambda-1}}$, $q_{j_\lambda}' \in Q_{i_\lambda}$ $(\lambda = 1, \ldots, l)$, and $q_d' \in Q_{i_l}$ (see Fig. 3). For each $j := j_\lambda \in J$ set $\tilde{n}_j := \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i$.

Let $j \in \mathbb{N}$, and let π_0 be a path in M realizing some $(y_0, z_0) \in \Sigma^* \times \Delta^*$ and leading from some $p \in Q$ to some $q \in Q$. Let $1 \leq i(p) \leq i(q) \leq k$ so that $p \in Q_{i(p)}$ and $q \in Q_{i(q)}$. Set $\tilde{n} := \sum_{i=i(p)}^{i(q)} n_i$. We define $\varphi_j(\pi_0) \in 2^{\Delta^* \cup \mathcal{Z} \times Q^2}$:

$$\begin{array}{rcl} \varphi_j(\pi_0) & := & \{z_0\}, \quad \text{if } j \text{ is even or } |z_0| \text{ is at most } (n^2 \cdot \tilde{n} - 1) \cdot \text{diff}(\delta), \text{ and else,} \\ \varphi_j(\pi_0) & := & \{(t, \tilde{p}, \tilde{q}) \in \mathcal{Z} \times Q^2 \,|\, |t| \leq (n^2 \cdot \tilde{n} - 1) \cdot \text{diff}(\delta) \quad \& \quad \tilde{p} \xleftrightarrow{\text{M}} \tilde{q} \quad \& \\ & & \exists \tilde{z} \in \Delta^* \quad \exists \text{ a path } \tilde{\pi} \text{ in } M \text{ realizing } (y_0, \tilde{z}) \text{ and leading from } \tilde{p} \text{ to } \tilde{q}: \\ & & \text{diff}(\pi_0, \tilde{\pi}) \leq (n^3 - 1) \cdot \text{diff}(\delta) \quad \& \quad t = |z_0| - |\tilde{z}|\}. \end{array}$$

We are now ready to define $\varphi(\pi) \in 2^S$:

$$\varphi(\pi) := \{p_1'\} \times \prod_{\lambda=1}^{l} [\{j_\lambda\} \times \{p_{j_\lambda}'\} \times \varphi_{j_\lambda}(\pi_{j_\lambda}) \times \{q_{j_\lambda}'\}] \times \{q_d'\}.$$

Next, we prove that $\varphi(\pi)$ is nonempty, i.e., we have to show that, for all $j \in J$, $\varphi_j(\pi_j)$ is nonempty. Let $j := j_\lambda \in J$ such that $z_j \notin \varphi_j(\pi_j)$. Then, j is odd and $|z_j|$ is greater

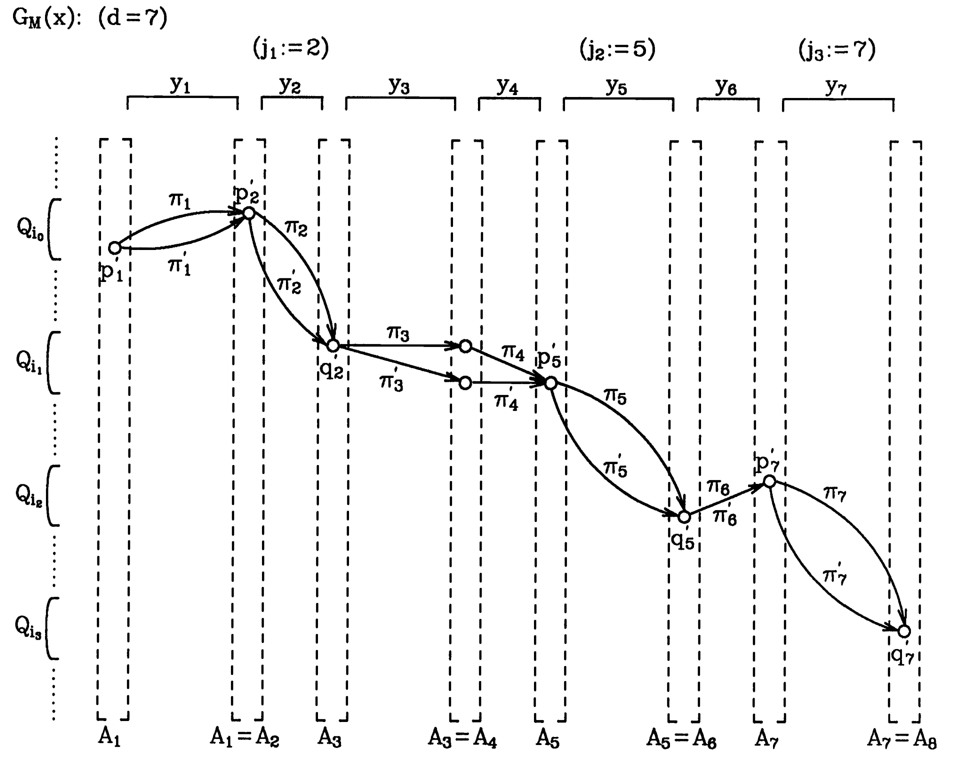


FIG. 3

than $(n^2 \cdot \tilde{n}_j - 1) \cdot \text{diff}(\delta)$. Since j is odd and $y_j \neq \varepsilon$, we can apply Lemma A.2 in the Appendix to $A := A_j = A_{j+1}$, $y := y_j$, $p' := p'_j \in A$, and $q' := q'_j \in A$, which yields

(*) There are $p, q \in A, p_1, p_2, p_3, p_4 \in Q, l_1, l_2 \geq 0$ and $z_1^{(1)}, \ldots, z_4^{(1)}, z_6^{(1)}, \ldots, z_9^{(1)} \in \Delta^*$ so that $(p, y^{l_1}, z_1^{(1)}, p_1), (p_1, y, z_2^{(1)}, p_2), (p_2, y^{l_2}, z_3^{(1)}, p) \in \delta$, $(p, y^{l_1}, z_4^{(1)}, p')$, $(q', y^{l_2}, z_6^{(1)}, q) \in \delta$ and $(q, y^{l_1}, z_7^{(1)}, p_3), (p_3, y, z_8^{(1)}, p_4), (p_4, y^{l_2}, z_9^{(1)}, q) \in \delta$.

Recall that π_j realizes (y, z_j) and leads from p' to q', and that $p' \in Q_{i_{\lambda-1}}$ and $q' \in Q_{i_\lambda}$. Select $\tilde{p} := p_1$, $\tilde{q} := p_2$, $\tilde{z} := z_2^{(1)}$, and $t := |z_j| - |\tilde{z}|$. Let $\tilde{\pi}$ be an arbitrary path realizing $(y, \tilde{z})$ and leading from $\tilde{p}$ to $\tilde{q}$. Since M does not comply with the criterion (IV1), we can apply Lemma A.3 in the Appendix to $U_1 := \bigcup_{i=1}^{i_{\lambda-1}-1} Q_i$, $U_2 := \bigcup_{i=i_{\lambda-1}}^{i_\lambda - 1} Q_i$, $U_3 := Q_{i_\lambda}$, $U_4 := \bigcup_{i=i_\lambda+1}^{k} Q_i$, $A, y, p' \in A \cap U_2$, $q' \in A \cap U_3$, the assertion (*), π_j, and $\tilde{\pi}$. Lemma A.3 implies: $\text{diff}(\pi_j, \tilde{\pi}) \leq (n^2 \cdot \tilde{n}_j - 1) \cdot \text{diff}(\delta)$. Thus, $(t, \tilde{p}, \tilde{q})$ belongs to $\varphi_j(\pi_j)$.

Assume that $\pi, \pi' \in \Pi$ are accepting paths consuming x and producing some $z, z' \in \Delta^*$, respectively, such that $\varphi(\pi) \cap \varphi(\pi')$ is nonempty. We shall show that z and z' coincide.

Consider the uniquely determined paths $\pi_1, \ldots, \pi_d, \pi'_1, \ldots, \pi'_d$ and the uniquely determined $z_1, \ldots, z_d, z'_1, \ldots, z'_d \in \Delta^*$ and $p_j^{(1)}, q_j^{(1)}, p_j^{(2)}, q_j^{(2)} \in Q$ $(j = 1, \ldots, d)$ such that $\pi = \pi_1 \circ \ldots \circ \pi_d$, $\pi' = \pi'_1 \circ \ldots \circ \pi'_d$ and, for each $j = 1, \ldots, d$, π_j realizes (y_j, z_j) and leads from $p_j^{(1)}$ to $q_j^{(1)}$ and π'_j realizes (y_j, z'_j) and leads from $p_j^{(2)}$ to $q_j^{(2)}$. By construction, $z = z_1 \ldots z_d$ and $z' = z'_1 \ldots z'_d$. We define

$$J \ := \ \{j \in [d] \mid \neg(p_j^{(1)} \xleftrightarrow{M} q_j^{(1)})\} \quad \text{and}$$

$$J' \;:=\; \{j \in [d] \mid \neg(p_j^{(2)} \overset{\mathrm{M}}{\longleftrightarrow} q_j^{(2)})\}.$$

Since $\varphi(\pi) \cap \varphi(\pi')$ is nonempty, we know: $J = J'$, $p_1^{(1)} = p_1^{(2)} =: p'_1$, $p_j^{(1)} = p_j^{(2)} =: p'_j$, $q_j^{(1)} = q_j^{(2)} =: q'_j$ $(j \in J)$, $q_d^{(1)} = q_d^{(2)} =: q'_d$. Let $l \in \{0,\dots,k-1\}$ and $1 \le j_1 < \dots < j_l \le d$ so that $J = \{j_1,\dots,j_l\}$. Let $1 \le i_0 < i_1 < \dots < i_l \le k$ so that $p'_1 \in Q_{i_0}$, $p'_{j_\lambda} \in Q_{i_{\lambda-1}}$, $q'_{j_\lambda} \in Q_{i_\lambda}$ $(\lambda = 1,\dots,l)$, and $q'_d \in Q_{i_l}$ (see Fig. 3). For each $j := j_\lambda \in J$ set $\tilde{n}_j := \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i$.

Set $j_0 := 0$, $j_{l+1} := d+1$, $q'_0 := p'_1$ and $p'_{d+1} := q'_d$. Let $\lambda \in \{0,\dots,l\}$. By construction, for some $y \in \Sigma^*$, $(q'_{j_\lambda}, y, z_{j_\lambda+1}\cdots z_{j_{\lambda+1}-1}, p'_{j_{\lambda+1}}) \in \delta$ and $(q'_{j_\lambda}, y, z'_{j_\lambda+1}\cdots z'_{j_{\lambda+1}-1}, p'_{j_{\lambda+1}}) \in \delta$. Since M does not comply with any of the criteria (IV1), (IV2), and since q'_{j_λ} is connected with $p'_{j_{\lambda+1}}$, this implies: $z_{j_\lambda+1}\cdots z_{j_{\lambda+1}-1} = z'_{j_\lambda+1}\cdots z'_{j_{\lambda+1}-1}$ (see Fig. 3).

Let $j := j_\lambda \in J$. Since $\varphi(\pi) \cap \varphi(\pi')$ is nonempty, we know that $\varphi_j(\pi_j) \cap \varphi_j(\pi'_j) \neq \emptyset$. If j is even or $|z_j|$ is at most $(n^2 \cdot \tilde{n}_j - 1)\cdot \mathrm{diff}(\delta)$, then $\{z_j\} = \varphi_j(\pi_j) = \varphi_j(\pi'_j) = \{z'_j\}$, i.e., $z_j = z'_j$. Otherwise, consider $(t, \tilde{p}, \tilde{q}) \in \mathcal{Z} \times Q^2$ so that $(t, \tilde{p}, \tilde{q}) \in \varphi_j(\pi_j) \cap \varphi_j(\pi'_j)$. Since M does not comply with any of (IV1), (IV2), and since $\tilde{p}$ is connected with $\tilde{q}$, there is exactly one $\tilde{z} \in \Delta^*$ such that $(\tilde{p}, y_j, \tilde{z}, \tilde{q}) \in \delta$. Hence, we know: $|z_j| - |\tilde{z}| = t = |z'_j| - |\tilde{z}|$, i.e., $|z_j| = |z'_j|$.

Since j is odd and $y_j \neq \varepsilon$, we can apply Lemma A.2 to $A := A_j = A_{j+1}$, $y := y_j$, $p' := p'_j \in A$, and $q' := q'_j \in A$, which yields $(*)$. Let $\tilde{\pi}$ be an arbitrary path realizing $(y, z_2^{(1)})$ and leading from p_1 to p_2. Since M does not comply with (IV1), we can apply Lemma A.3 to $U_1 := \bigcup_{i=1}^{i_{\lambda-1}-1} Q_i$, $U_2 := \bigcup_{i=i_{\lambda-1}}^{i_\lambda-1} Q_i$, $U_3 := Q_{i_\lambda}$, $U_4 := \bigcup_{i=i_\lambda+1}^{k} Q_i$, A, y, $p' \in A \cap U_2$, $q' \in A \cap U_3$, the assertion $(*)$, π_j, and $\tilde{\pi}$. Lemma A.3 implies: $||z_j| - |z_2^{(1)}|| \le (n^2 \cdot \tilde{n}_j - 1)\cdot \mathrm{diff}(\delta)$. Thus, we know in addition to $(*)$

$$(p', y, z_j, q'), (p', y, z'_j, q') \in \delta \quad \& \quad |z_j| = |z'_j| \quad \& \quad z_2^{(1)} \neq \varepsilon.$$

Since M does not comply with (IV2), this implies: $z_j = z'_j$. In summary, we have shown that z and z' coincide.

Execution of step (3). Let $\sigma \in S$. We will effectively construct an NFT $M_\sigma = (Q_\sigma, \Sigma, \Delta, \delta_\sigma, Q_{I,\sigma}, Q_{F,\sigma})$, which realizes the transduction $\{(x,z) \in \Sigma^* \times \Delta^* \mid (x,z)$ is realized by some $\pi \in \Pi$ so that $\sigma \in \varphi(\pi)\}$, and which has the property that any path in M_σ consuming ε also produces ε. It is easy to see that M_σ is equivalent to the (effectively constructible) NGSM $\bar{M}_\sigma = (Q_\sigma, \Sigma, \Delta, \bar{\delta}_\sigma, \bar{Q}_{I,\sigma}, Q_{F,\sigma})$ where

$$\begin{aligned}
\bar{Q}_{I,\sigma} &:= \{q \in Q_\sigma \mid \exists p \in Q_{I,\sigma}\colon (p,\varepsilon,\varepsilon,q) \in (\hat{\delta_\sigma})\} \quad \text{and}\\
\bar{\delta}_\sigma &:= \{(p,a,z,q) \in Q_\sigma \times \Sigma \times \Delta^* \times Q_\sigma \mid \exists r \in Q_\sigma\colon\\
&\qquad (p,a,z,r) \in \delta_\sigma \quad \& \quad (r,\varepsilon,\varepsilon,q) \in (\hat{\delta_\sigma})\}.
\end{aligned}$$

Having constructed M_σ, we prove that it works as stated above. Finally we observe that $\#Q_\sigma$ and $\|\bar{M}_\sigma\|$ are bounded as desired.

Let $l \ge 0$, $1 \le i_0 < \dots < i_l \le k$ and $0 < j_1 < \dots < j_l < 2^{n+1}$ so that $\sigma = (q_I, (j_1, p'_{j_1}, \sigma_{j_1}, q'_{j_1}), \dots, (j_l, p'_{j_l}, \sigma_{j_l}, q'_{j_l}), q_F)$ where $q_I \in Q_I \cap Q_{i_0}$, $p'_{j_\lambda} \in Q_{i_{\lambda-1}}$, $q'_{j_\lambda} \in Q_{i_\lambda}$ $(\lambda = 1,\dots,l)$, and $q_F \in Q_F \cap Q_{i_l}$. For each $j := j_\lambda \in \{j_1,\dots,j_l\}$ set $\tilde{n}_j := \sum_{i=i_{\lambda-1}}^{i_\lambda} n_i$. Define $J := \{j_1,\dots,j_l\} = J_1 \,\dot\cup\, J_2$ where $J_1 := \{j \in J \mid z_j := \sigma_j \in \Delta^*\}$ and $J_2 := \{j \in J \mid (t_j, \tilde{p}_j, \tilde{q}_j) := \sigma_j \in \mathcal{Z} \times Q^2\}$. Let $j \in J$. If $j \in J_1$, then $z_j \in \mathrm{im}(\delta)$ or $|z_j|$ is at most $(n^2 \cdot \tilde{n}_j - 1)\cdot \mathrm{diff}(\delta)$. Otherwise, $|t_j|$ is at most $(n^2 \cdot \tilde{n}_j - 1)\cdot \mathrm{diff}(\delta)$, and $\tilde{p}_j$ is connected with $\tilde{q}_j$.

By construction of φ we know: If, for some $j \in J_2$, j is even or if, for some $j \in J_1$, j is odd and $|z_j| > (n^2 \cdot \tilde{n}_j - 1) \cdot \mathrm{diff}(\delta)$ or j is even and $z_j \notin \mathrm{im}(\delta)$, then σ does not belong to $\varphi(\Pi)$. In this case we can select M_σ so that $T(M_\sigma) = \emptyset$. Let us therefore assume that, for all $j \in J_2$, j is odd and that, for all $j \in J_1$, either j is odd and $|z_j| \leq (n^2 \cdot \tilde{n}_j - 1) \cdot \mathrm{diff}(\delta)$ or j is even and $z_j \in \mathrm{im}(\delta)$.

Before we construct M_σ in detail, we explain the desired mode of operation of an arbitrary accepting path π_σ in this machine. Assume that π_σ realizes $(x, z) \in \Sigma^* \times \Delta^*$. Let $x_1, \ldots, x_m \in \Sigma$ so that $x = x_1 \ldots x_m$. Consider $G_M(x) = (V, E)$. Let $\mu \in \{0, \ldots, m\}$. We define

$$\begin{aligned} \mathrm{att}(x,\mu) &:= \{s \in Q \mid \exists r \in Q_I \, \exists z \in \Delta^* \colon (r, x_1 \ldots x_\mu, z, s) \in \delta\} \quad \text{and} \\ \mathrm{der}(x,\mu) &:= \{r \in Q \mid \exists s \in Q_F \, \exists z \in \Delta^* \colon (r, x_{\mu+1} \ldots x_m, z, s) \in \delta\}; \end{aligned}$$

$\mathrm{att}(x,\mu)$ and $\mathrm{der}(x,\mu)$ denote the set of states attainable from Q_I with $x_1 \ldots x_\mu$ and the set of states derivable to Q_F with $x_{\mu+1} \ldots x_m$, respectively. In step (2), $\mathrm{set}(x,\mu)$ was defined as the set of states at column μ in $G_M(x)$. Clearly, this set equals $\mathrm{att}(x,\mu) \cap \mathrm{der}(x,\mu)$. The reader may further recall from step (2) the definition of $A_1, \ldots, A_{d+1} \in 2^Q$ and of $y_1, \ldots, y_d \in \Sigma^*$ (where $x = y_1 \ldots y_d$ and d is odd and at most $2^{n+1} - 1$).

The path π_σ consists of five components that correspond to five components of Q_σ. Roughly spoken, the first four components of π_σ provide the index $j \in [d]$ of the word y_j currently consumed by π_σ. These components are independent of σ and z. The fifth component of π_σ guesses an accepting path $\pi \in \Pi$ realizing (x, z) and uses the basic information provided by the first four components in order to verify "on line" that σ belongs to $\varphi(\pi)$. Note that π_σ inherits z from π.

The first component of π_σ constantly contains $(A_1, \ldots, A_{2^{n+1}}) \in (2^Q)^{2^{n+1}}$, where $A_1, \ldots, A_{d+1}$ are as above and $A_{d+2}, \ldots, A_{2^{n+1}}$ are arbitrary. $(A_1, \ldots, A_{2^{n+1}})$ is guessed at the beginning of π_σ. Note that we need $A_{d+2}, \ldots, A_{2^{n+1}}$ only in order to make Q_σ "well typed."

The next three components of π_σ drive a nondeterministic process, which verifies the correctness of $A_1, \ldots, A_{d+1}$ and uses these sets in order to provide the index $j \in [d]$ of the word y_j currently consumed by π_σ. Assume that, for some $\mu \in \{0, \ldots, m\}$, π_σ has consumed $x_1 \ldots x_\mu$. Then, the second (deterministic) and third (nondeterministic) components of π_σ contain $\mathrm{att}(x,\mu)$ and $\mathrm{der}(x,\mu)$, respectively.

The fourth component of π_σ contains some $(j, \alpha) \in [d+1] \times [3]$ so that the following holds: If $\alpha = 1$, then either $j \leq d$ and π_σ can, after one transition realizing $(\varepsilon, \varepsilon)$, begin to consume the letters of y_j or π_σ guesses that $j = d + 1$ and accepts. In the latter case, of course, j is even and greater than j_l. If $\alpha \in \{2, 3\}$, then $j \leq d$. If $\alpha = 2$, then j is even and π_σ is ready to consume the only letter of y_j. If $\alpha = 3$ and j is even, then y_j has been completely consumed. If $\alpha = 3$ and j is odd, then either π_σ is ready to consume the next letter of y_j or y_j has been completely consumed, depending on the guess of π_σ. Whenever all letters of a y_j have been consumed, π_σ increments j by 1 on a transition realizing $(\varepsilon, \varepsilon)$. At the beginning of π_σ its fourth component contains $(1, 1)$. The distinction between the values 2 and 3 for α is needed in order to ensure that $|y_j| = 1$ for all even $j \in [d]$.

The second and third component of π_σ is used to check that, for each $j \in [d + 1]$, $A_j = \mathrm{set}(x, |y_1 \ldots y_{j-1}|)$ and that, for each odd $j \in [d]$, A_{j+1} is the "last occurrence" of A_j in the sequence $\mathrm{set}(x, 0), \mathrm{set}(x, 1), \ldots, \mathrm{set}(x, m)$. Therefore, these components contribute to the verification of the first and the fourth component of π_σ.

For the fifth component of π_σ we first of all need, for all $j \in J$, an NGSM $M_{\sigma,j}$ realizing the transduction

$$T_{\sigma,j} := \{(y_0, z_0) \in \Sigma^* \times \Delta^* \mid (y_0, z_0) \text{ is realized by some path } \pi_0 \text{ in } M \text{ leading from } p'_j \text{ to } q'_j \text{ so that } \sigma_j \in \varphi_j(\pi_0)\}.$$

Informally spoken, an accepting path in $M_{\sigma,j}$ simply guesses (in its first component) a path π_0 in M leading from p'_j to q'_j, realizes the same $(y_0, z_0) \in \Sigma^* \times \Delta^*$ as π_0 and verifies (on its three other components) that σ_j belongs to $\varphi_j(\pi_0)$. The verification procedure directly arises from the definition of φ_j in step (2). The detailed construction of $M_{\sigma,j}$ is given below.

The fifth component of π_σ verifies that, for some $\pi \in \Pi$ realizing (x, z), σ belongs to $\varphi(\pi)$. Following the definition of φ this component operates as follows: For all $\lambda \in \{0, \ldots, l\}$, while π_σ consumes $y_{j_\lambda+1}, \ldots, y_{j_{\lambda+1}-1}$, one after another, it guesses and verifies a path in M consuming $y_{j_\lambda+1} \ldots y_{j_{\lambda+1}-1}$, producing some $z_{j_\lambda+1}, \ldots, z_{j_{\lambda+1}-1} \in \Delta^*$, one after another, and leading from q'_{j_λ} to $p'_{j_{\lambda+1}}$ (where $j_0 := 0$, $j_{l+1} := d+1$, $q'_0 := q_I$, $p'_{d+1} := q_F$). For all $j := j_\lambda \in J$ ($\lambda \in [l]$), while π_σ consumes y_j, this component guesses and verifies an accepting path in $M_{\sigma,j}$ consuming y_j and producing some $z_j \in \Delta^*$. The index $j \in [d]$ of the word y_j currently consumed by π_σ is read from its fourth component. π_σ inherits $z = z_1 \ldots z_d \in \Delta^*$ from the combination of the above paths.

Let $j \in J$. We construct the NGSM $M_{\sigma,j} = (Q_{\sigma,0}, \Sigma, \Delta, \delta_{\sigma,j}, Q_{I,\sigma,j}, Q_{F,\sigma,j})$:

$$\begin{aligned} Q_{\sigma,0} &:= Q \times Q \times [-(n^3-1)\cdot \mathrm{diff}(\delta), (n^3-1)\cdot \mathrm{diff}(\delta)] \\ &\quad \times [0, \max\{\mathrm{iml}(\delta), 1 + (n^3-1)\cdot \mathrm{diff}(\delta)\}], \end{aligned}$$

if $j \in J_1$:

$$\begin{aligned} Q_{I,\sigma,j} &:= \{(p'_j, p'_j, 0, 0)\}, \\ Q_{F,\sigma,j} &:= \{(q'_j, q'_j, 0, |z_j|)\}, \\ \delta_{\sigma,j} &:= \{((p,p,0,\tau), a, z, (q,q,0,\tau')) \in Q_{\sigma,0} \times \Sigma \times \Delta^* \times Q_{\sigma,0} \mid (p,a,z,q) \in \delta \\ &\quad \& \quad \tau' = \tau + |z| \le |z_j| \quad \& \quad \forall \nu \in [|z|]\colon\ z(\nu) = z_j(\tau+\nu)\}, \end{aligned}$$

if $j \in J_2$:

$$\begin{aligned} Q_{I,\sigma,j} &:= \{(p'_j, \tilde{p}_j, 0, 0)\}, \\ Q_{F,\sigma,j} &:= \{(q'_j, \tilde{q}_j, t_j, \tau) \in Q_{\sigma,0} \mid \tau > (n^2 \cdot \tilde{n}_j - 1)\cdot \mathrm{diff}(\delta)\}, \\ \delta_{\sigma,j} &:= \{((p,\tilde{p},t,\tau), a, z, (q,\tilde{q},t',\tau')) \in Q_{\sigma,0} \times \Sigma \times \Delta^* \times Q_{\sigma,0} \mid (p,a,z,q) \in \delta \\ &\quad \& \quad \exists \tilde{z} \in \Delta^*\colon \quad (\tilde{p}, a, \tilde{z}, \tilde{q}) \in \delta \quad \& \quad t' = t + |z| - |\tilde{z}| \\ &\quad \& \quad \tau' = \min\{\tau + |z|, 1 + (n^3-1)\cdot \mathrm{diff}(\delta)\}\}. \end{aligned}$$

Now we are ready to construct the NFT $M_\sigma = (Q_\sigma, \Sigma, \Delta, \delta_\sigma, Q_{I,\sigma}, Q_{F,\sigma})$:

$$\begin{aligned} Q_\sigma &:= (2^Q)^{2^{n+1}} \times 2^Q \times 2^Q \times ([2^{n+1}] \times [3]) \times Q_{\sigma,0}, \\ Q_{I,\sigma} &:= \{((A_1, \ldots, A_{2^{n+1}}), B_1, B_2, (1,1), (q_I, q_I, 0, 0)) \in Q_\sigma \mid \\ &\quad (\forall 1 \le j_1 < j_2 \le 2^{n+1}\colon \quad A_{j_1} = A_{j_2} \quad \Leftrightarrow \quad j_2 = j_1 + 1 \quad \& \quad j_1 \text{ odd}) \\ &\quad \& \quad B_1 = Q_I \quad \& \quad B_1 \cap B_2 = A_1\}, \\ Q_{F,\sigma} &:= \{((A_1, \ldots, A_{2^{n+1}}), B_1, B_2, (j,1), (q_F, q_F, 0, 0)) \in Q_\sigma \mid \\ &\quad B_2 = Q_F \quad \& \quad j \text{ even} \quad \& \quad j > j_l\}, \end{aligned}$$

$$\begin{aligned}
\delta_\sigma &:= \delta_\sigma^{(1)} \cup \delta_\sigma^{(2)} \cup \delta_\sigma^{(3)}, \quad \text{where} \\
\delta_\sigma^{(1)} &:= \{(((A_1,\dots,A_{2^n+1}),B_1,B_2,(j,1),(q,q,0,0)),\varepsilon,\varepsilon, \\
&\quad ((A_1,\dots,A_{2^n+1}),B_1,B_2,(j,\alpha'),(q,\tilde{q},0,0))) \in Q_\sigma \times \{\varepsilon\} \times \{\varepsilon\} \times Q_\sigma \mid \\
&\quad (j \text{ odd} \;\Rightarrow\; \alpha' = 3) \;\;\&\;\; (j \text{ even} \;\Rightarrow\; \alpha' = 2) \;\;\& \\
&\quad (j \notin J \;\Rightarrow\; q = \tilde{q}) \;\;\&\;\; (j \in J \;\Rightarrow\; (q,\tilde{q},0,0) \in Q_{I,\sigma,j})\}, \\
\delta_\sigma^{(2)} &:= \{(((A_1,\dots,A_{2^n+1}),B_1,B_2,(j,\alpha),(p,\tilde{p},t,\tau)),a,z, \\
&\quad ((A_1,\dots,A_{2^n+1}),B_1',B_2',(j,\alpha'),(q,\tilde{q},t',\tau'))) \in Q_\sigma \times \Sigma \times \Delta^* \times Q_\sigma \mid \\
&\quad B_1' = \{s \in Q \mid \exists r \in B_1\, \exists z' \in \Delta^*\colon (r,a,z',s) \in \delta\} \;\;\& \\
&\quad B_2 = \{r \in Q \mid \exists s \in B_2'\, \exists z' \in \Delta^*\colon (r,a,z',s) \in \delta\} \;\;\& \\
&\quad B_1' \cap B_2' \notin \{A_1,\dots,A_{j-1}\} \;\;\&\;\; (j \text{ odd} \;\Rightarrow\; \alpha = \alpha' = 3) \;\;\& \\
&\quad (j \text{ even} \;\Rightarrow\; (\alpha,\alpha') = (2,3)) \;\;\&\;\; (j \notin J \;\Rightarrow\; (p,a,z,q) \in \delta \;\;\& \\
&\quad (p,\tilde{p},t,\tau) = (p,p,0,0) \;\;\&\;\; (q,\tilde{q},t',\tau') = (q,q,0,0)) \;\;\& \\
&\quad (j \in J \;\Rightarrow\; ((p,\tilde{p},t,\tau),a,z,(q,\tilde{q},t',\tau')) \in \delta_{\sigma,j})\}, \\
\delta_\sigma^{(3)} &:= \{(((A_1,\dots,A_{2^n+1}),B_1,B_2,(j,3),(q,\tilde{q},t,\tau)),\varepsilon,\varepsilon, \\
&\quad ((A_1,\dots,A_{2^n+1}),B_1,B_2,(j+1,1),(q,q,0,0))) \in Q_\sigma \times \{\varepsilon\} \times \{\varepsilon\} \times Q_\sigma \mid \\
&\quad B_1 \cap B_2 = A_{j+1} \;\;\&\;\; (j \notin J \;\Rightarrow\; (q,\tilde{q},t,\tau) = (q,q,0,0)) \;\;\& \\
&\quad (j \in J \;\Rightarrow\; (q,\tilde{q},t,\tau) \in Q_{F,\sigma,j})\}.
\end{aligned}$$

FACT 2.8. *For all $j \in J$, the* NGSM $M_{\sigma,j}$ *realizes the transduction* $T_{\sigma,j}$.

Proof.

Case 1. $j \in J_1$, i.e., $\sigma_j = z_j \in \Delta^*$. Then, we have: $T_{\sigma,j} = \{(y_0, z_j) \in \Sigma^* \times \Delta^* \mid (y_0, z_j)$ is realized by some path π_0 in M leading from p_j' to $q_j'\} = T(M_{\sigma,j})$. Note that for the first equality we have used our assumption that j is even or $|z_j| \le (n^2 \cdot \tilde{n}_j - 1) \cdot \text{diff}(\delta)$.

Case 2. $j \in J_2$, i.e., $\sigma_j = (t_j, \tilde{p}_j, \tilde{q}_j) \in \mathcal{Z} \times Q^2$. Then, we have: $T_{\sigma,j} = \{(y_0, z_0) \in \Sigma^* \times \Delta^* \mid$ there are paths π_0 and $\tilde{\pi}$ in M realizing (y_0, z_0) and $(y_0, \tilde{z})$, for some $\tilde{z} \in \Delta^*$, and leading from p_j' to q_j' and from $\tilde{p}_j$ to $\tilde{q}_j$, respectively, so that $\text{diff}(\pi_0, \tilde{\pi}) \le (n^3 - 1) \cdot \text{diff}(\delta)$, $t_j = |z_0| - |\tilde{z}|$, and $|z_0| > (n^2 \cdot \tilde{n}_j - 1) \cdot \text{diff}(\delta)\} = T(M_{\sigma,j})$. Note that for the first equality we have used our assumption that j is odd. □

The NFT M_σ works as desired above. Thus, using Fact 2.8, it is easy to establish that this NFT realizes the transduction $\{(x,z) \in \Sigma^* \times \Delta^* \mid (x,z)$ is realized by some $\pi \in \Pi$ so that $\sigma \in \varphi(\pi)\}$.

Note that, for each transition (r,a,z,s) of M_σ, either $(a,z) = (\varepsilon,\varepsilon)$ or there are $p,q \in Q$ such that (p,a,z,q) is a transition of M. Any path in M_σ consuming ε has length at most 4 and also produces ε. Of course, the construction of M_σ is effective (see Theorem 2.3 for details). The following fact states upper bounds for $\#Q_\sigma$ and $\|\bar{M}_\sigma\|$ and for some other useful parameters of $M_{\sigma,j}$ $(j \in J)$, M_σ and $\bar{M}_\sigma$.

FACT 2.9. *The following assertions are true.*

(i) $\#Q_{\sigma,0} \le 2n^8 \cdot (1 + \text{diff}(\delta)) \cdot (2 + \text{iml}(\delta))$.

(ii) *For all* $j \in J$: $\|\delta_{\sigma,j}\| \le \#Q_{\sigma,0}^2 \cdot \|\delta\|$, $\|M_{\sigma,j}\| \le \#Q_{\sigma,0}^2 \cdot \|M\| \le 4n^{16} \cdot \|M\|^5$.

(iii) $\#Q_\sigma \le 2^{n \cdot (2^{n+1}+12)} \cdot (1 + \text{diff}(\delta)) \cdot (2 + \text{iml}(\delta))$.

(iv) $\text{diff}(\bar{\delta}_\sigma) = \text{diff}(\delta_\sigma) \le \text{diff}(\delta)$, $\text{im}(\bar{\delta}_\sigma) = \text{im}(\delta_\sigma) \subseteq \text{im}(\delta)$, $\text{iml}(\bar{\delta}_\sigma) = \text{iml}(\delta_\sigma) \le \text{iml}(\delta)$, $\|\delta_\sigma\| \le \#Q_\sigma^2 \cdot (1 + \|\delta\|)$, $\|\bar{\delta}_\sigma\| \le \#Q_\sigma^2 \cdot \|\delta\|$, $\|M_\sigma\| \le \#Q_\sigma^2 \cdot (1 + \|M\|)$, $\|\bar{M}_\sigma\| \le \#Q_\sigma^2 \cdot \|M\| \le 2^{n \cdot (2^{n+2}+24)} \cdot \|M\|^5$.

(v) *Let* $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ *be another* NGSM. *Then,* $\text{diff}(\bar{\delta}_\sigma, \delta_0) \le \text{diff}(\delta, \delta_0)$.

Proof. The assertions (i) and (ii) are obvious from the definition of $M_{\sigma,j}$ $(j \in J)$. The assertions (iii)–(v) follow from the definition of M_σ and $\bar{M}_\sigma$. Note that for the proof of (iii) we apply (i) and the estimation $n^8 \le 3^{8n/3} \le 2^{5n}$ (cf. proof of Proposition 2.7). □

This completes the proof of Theorem 2.1. □

Proof of Theorem 2.2. Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ and $n = \#Q$ be as in Theorem 2.1. Let S and $M_\sigma, \bar{M}_\sigma$ $(\sigma \in S)$ be as in the proof of this theorem. The NGSMs $\bar{M}_\sigma$ $(\sigma \in S)$ are playing the role of $M_1, \ldots, M_N$ in Theorem 2.1, where $N = \#S$. Therefore, all we have to do here is to construct effectively, for each $\sigma \in S$, an NFA $\bar{M}'_\sigma$ recognizing $\Sigma^* \backslash L(\bar{M}_\sigma) = \Sigma^* \backslash L(M_\sigma)$ and having at most $2^{n\cdot(2^{n+1}+8)} \cdot 2^{2n^8\cdot(1+\text{diff}(\delta))\cdot(2+\text{iml}(\delta))}$ states and size at most $2^{n\cdot(2^{n+2}+17)} \cdot 2^{5n^8\cdot\|M\|^2}$.

Let $\sigma \in S$. Following the main lines of the construction of M_σ in step (3) of the proof of Theorem 2.1, we will effectively construct an ε-NFA $M'_\sigma = (Q'_\sigma, \Sigma, \delta'_\sigma, Q'_{I,\sigma}, Q'_{F,\sigma})$ which recognizes $\Sigma^* \backslash L(M_\sigma)$, i.e., the language $\{x \in \Sigma^* \mid$ there is no $\pi \in \Pi$ consuming x so that $\sigma \in \varphi(\pi)\}$. It is easy to see that M'_σ is equivalent to the (effectively constructible) NFA $\bar{M}'_\sigma = (Q'_\sigma, \Sigma, \bar{\delta}'_\sigma, \bar{Q}'_{I,\sigma}, Q'_{F,\sigma})$ where

$$\begin{aligned}
\bar{Q}'_{I,\sigma} &:= \{q \in Q'_\sigma \mid \exists p \in Q'_{I,\sigma} \colon (p, \varepsilon, q) \in (\hat{\delta'_\sigma})\} \quad \text{and} \\
\bar{\delta}'_\sigma &:= \{(p, a, q) \in Q'_\sigma \times \Sigma \times Q'_\sigma \mid \exists r \in Q'_\sigma \colon (p, a, r) \in \delta'_\sigma \quad \& \quad (r, \varepsilon, q) \in (\hat{\delta'_\sigma})\}.
\end{aligned}$$

Having constructed M'_σ, we will see that it works as stated above. Finally we observe that $\#Q'_\sigma$ and $\|\bar{M}'_\sigma\|$ are bounded as desired.

Let $\sigma = (q_I, (j_1, p'_{j_1}, \sigma_{j_1}, q'_{j_1}), \ldots, (j_l, p'_{j_l}, \sigma_{j_l}, q'_{j_l}), q_F) \in S$ and $J, J_1, J_2, \tilde{n}_j$ $(j \in J)$, z_j $(j \in J_1)$ and $(t_j, \tilde{p}_j, \tilde{q}_j)$ $(j \in J_2)$ be given as in step (3) of the proof of Theorem 2.1. Following this proof we also assume here that, for all $j \in J_2$, j is odd and that, for all $j \in J_1$, either j is odd and $|z_j| \le (n^2 \cdot \tilde{n}_j - 1) \cdot \text{diff}(\delta)$ or j is even and $z_j \in \text{im}(\delta)$.

We recall from step (3) of the proof of Theorem 2.1 the NGSM $M_{\sigma,j} = (Q_{\sigma,0}, \Sigma, \Delta, \delta_{\sigma,j}, Q_{I,\sigma,j}, Q_{F,\sigma,j})$ $(j \in J)$ realizing the transduction $T_{\sigma,j}$. Let $j \in J$. Using the well-known subset construction we obtain from $M_{\sigma,j}$ the following DFA $M'_{\sigma,j} = (Q'_{\sigma,0}, \Sigma, \delta'_{\sigma,j}, Q'_{I,\sigma,j}, Q'_{F,\sigma,j})$ recognizing the language $\Sigma^* \backslash L(M_{\sigma,j})$.

$$\begin{aligned}
Q'_{\sigma,0} &:= 2^{Q_{\sigma,0}}, \\
Q'_{I,\sigma,j} &:= \{Q_{I,\sigma,j}\}, \\
Q'_{F,\sigma,j} &:= 2^{Q_{\sigma,0} \backslash Q_{F,\sigma,j}}, \\
\delta'_{\sigma,j} &:= \{(B, a, B') \in Q'_{\sigma,0} \times \Sigma \times Q'_{\sigma,0} \mid \\
&\qquad B' = \{s \in Q_{\sigma,0} \mid \exists r \in B \, \exists z \in \Delta^* \colon (r, a, z, s) \in \delta_{\sigma,j}\}\}.
\end{aligned}$$

According to the definition of $T_{\sigma,j}$, $L(M'_{\sigma,j}) = \{y_0 \in \Sigma^* \mid$ there is no path π_0 in M consuming y_0 and leading from p'_j to q'_j so that $\sigma_j \in \varphi_j(\pi_0)\}$.

Before we construct M'_σ in detail we explain the desired mode of operation of an arbitrary accepting path π'_σ in this machine. Assume that π'_σ consumes $x \in \Sigma^*$. Let $x_1, \ldots, x_m \in \Sigma$ so that $x = x_1 \ldots x_m$. From step (2) of the proof of Theorem 2.1 the reader may recall the definition of $y_1, \ldots, y_d \in \Sigma^*$ (where $x = y_1 \ldots y_d$ and d is odd and at most $2^{n+1} - 1$). We further ask once again to recall the main lines of the construction of $M_\sigma = (Q_\sigma, \Sigma, \Delta, \delta_\sigma, Q_{I,\sigma}, Q_{F,\sigma})$ in step (3) of this proof.

The path π'_σ consists of five components that correspond to five components of Q'_σ. Concerning the first four components, π'_σ operates exactly as an accepting path in M_σ consuming x. In particular, the value $(j, \alpha) \in [d + 1] \times [3]$ of the fourth component of π'_σ, if $j \le d$, determines the index j of the word y_j currently consumed by π'_σ. At the beginning respectively end of π'_σ its fourth component contains $(1,1)$ respectively $(d+1,1)$. One reason for π'_σ to accept x is that at its end $d+1 \le j_l$. Let us assume here that $j_l < d+1$.

The fifth component of π'_σ verifies that there is no $\pi \in \Pi$ consuming x such that σ belongs to $\varphi(\pi)$. Following the definition of φ this component verifies that (a) or (b), depending on its guess, holds: (a) For some (guessed) $\lambda \in \{0,\dots,l\}$ there is no path in M consuming $y_{j_\lambda+1}\cdots y_{j_{\lambda+1}-1}$ and leading from q'_{j_λ} to $p'_{j_{\lambda+1}}$ (where $j_0 := 0$, $j_{l+1} := d+1, q'_0 := q_I, p'_{d+1} := q_F$). (b) For some (guessed) $j := j_\lambda \in J$ ($\lambda \in [l]$) there is an accepting path in $M'_{\sigma,j}$ consuming y_j. Of course, condition (a) is verified (as described below) while π'_σ consumes $y_{j_\lambda+1},\dots,y_{j_{\lambda+1}-1}$, one after another, and condition (b) is verified by simulating $M'_{\sigma,j}$ while π'_σ consumes y_j. The index $j \in [d]$ of the word y_j currently consumed by π'_σ is read from its fourth component.

Technically the fifth component of π'_σ contains some $(B_3,\beta) \in Q'_{\sigma,0} \times [3]$. The set $Q'_{\sigma,0}$ is needed for the simulation of $M'_{\sigma,j}$ ($j \in J$). The value for β can never decrease along π'_σ. $\beta = 2$ means that π'_σ is about to verify (a) or (b). $\beta = 1$ ($\beta = 3$) means that this verification still has to be done (is already completed, respectively). If $\beta \in \{1,3\}$, then $B_3 = \emptyset$. If $\beta = 2$, if π'_σ is about to verify (a) for $\lambda \in \{0,\dots,l\}$, and if π'_σ has consumed the prefix y of $y_{j_\lambda+1}\cdots y_{j_{\lambda+1}-1}$, then $B_3 = \{(q,q,0,0) \mid \exists z' \in \Delta^*\colon\ (q'_{j_\lambda}, y, z', q) \in \delta\}$.

Now we construct the ε-NFA $M'_\sigma = (Q'_\sigma, \Sigma, \delta'_\sigma, Q'_{I,\sigma}, Q'_{F,\sigma})$:

$$\begin{aligned}
Q'_\sigma &:= (2^Q)^{2^{n+1}} \times 2^Q \times 2^Q \times ([2^{n+1}] \times [3]) \times (Q'_{\sigma,0} \times [3]),\\
Q'_{I,\sigma} &:= \{((A_1,\dots,A_{2^{n+1}}), B_1, B_2, (1,1), (B_3,\beta)) \in Q'_\sigma \mid\\
&\quad\ (\forall 1 \le j_1 < j_2 \le 2^{n+1}\colon\ \ A_{j_1} = A_{j_2} \ \Leftrightarrow\ j_2 = j_1 + 1 \ \ \&\ \ j_1 \text{ odd})\\
&\quad\ \&\ \ B_1 = Q_I \ \ \&\ \ B_1 \cap B_2 = A_1\\
&\quad\ \&\ \ (B_3,\beta) \in \{(\emptyset,1), (\{(q_I,q_I,0,0)\},2)\}\},\\
Q'_{F,\sigma} &:= \{((A_1,\dots,A_{2^{n+1}}), B_1, B_2, (j,1), (B_3,\beta)) \in Q'_\sigma \mid\\
&\quad\ B_2 = Q_F \ \ \&\ \ j \text{ even} \ \ \&\ \ (j \le j_l \ \ \vee\ \ (B_3,\beta) = (\emptyset,3)\\
&\quad\ \vee\ \ (B_3,\beta) \in 2^{Q_{\sigma,0}\setminus\{(q_F,q_F,0,0)\}} \times \{2\})\},\\
\delta'_\sigma &:= \delta^{(4)}_\sigma \cup \delta^{(5)}_\sigma \cup \delta^{(6)}_\sigma, \text{where}\\
\delta^{(4)}_\sigma &:= \{(((A_1,\dots,A_{2^{n+1}}), B_1, B_2, (j,1), (B_3,\beta)), \varepsilon,\\
&\quad\ ((A_1,\dots,A_{2^{n+1}}), B_1, B_2, (j,\alpha'), (B'_3,\beta'))) \in Q'_\sigma \times \{\varepsilon\} \times Q'_\sigma \mid\\
&\quad\ (j \text{ odd} \ \Rightarrow\ \alpha' = 3) \ \ \&\ \ (j \text{ even} \ \Rightarrow\ \alpha' = 2) \ \ \&\\
&\quad\ [(B_3,\beta) = (B'_3,\beta') \in \{(\emptyset,1),(\emptyset,3)\} \ \ \vee\\
&\quad\ (j \notin J \ \ \&\ \ (B_3,\beta) = (B'_3,\beta') \in Q'_{\sigma,0} \times \{2\}) \ \ \vee\\
&\quad\ (j \in J \ \ \&\ \ (B_3,\beta) \in 2^{Q_{\sigma,0}\setminus\{(p'_j,p'_j,0,0)\}} \times \{2\} \ \ \&\ \ (B'_3,\beta') = (\emptyset,3))\\
&\quad\ \vee\ \ (j \in J \ \ \&\ \ (B_3,\beta) = (\emptyset,1) \ \ \&\ \ (B'_3,\beta') \in Q'_{I,\sigma,j} \times \{2\})]\},\\
\delta^{(5)}_\sigma &:= \{(((A_1,\dots,A_{2^{n+1}}), B_1, B_2, (j,\alpha), (B_3,\beta)), a,\\
&\quad\ ((A_1,\dots,A_{2^{n+1}}), B'_1, B'_2, (j,\alpha'), (B'_3,\beta))) \in Q'_\sigma \times \Sigma \times Q'_\sigma \mid\\
&\quad\ B'_1 = \{s \in Q \mid \exists r \in B_1\, \exists z' \in \Delta^*\colon\ (r,a,z',s) \in \delta\} \ \ \&\\
&\quad\ B_2 = \{r \in Q \mid \exists s \in B'_2\, \exists z' \in \Delta^*\colon\ (r,a,z',s) \in \delta\} \ \ \&\\
&\quad\ B'_1 \cap B'_2 \notin \{A_1,\dots,A_{j-1}\} \ \ \&\\
&\quad\ (j \text{ odd} \ \Rightarrow\ \alpha = \alpha' = 3) \ \ \&\ \ (j \text{ even} \ \Rightarrow\ (\alpha,\alpha') = (2,3)) \ \ \&\\
&\quad\ [(B_3,\beta) = (B'_3,\beta') \in \{(\emptyset,1),(\emptyset,3)\} \ \ \vee\ \ (j \notin J \ \ \&
\end{aligned}$$

$$\begin{aligned}
& B_3' = \{(q,q,0,0) \mid \exists\,(p,p,0,0) \in B_3\, \exists z' \in \Delta^*\colon\ (p,a,z',q) \in \delta\} \\
& \&\quad \beta = 2) \quad \vee \quad (j \in J \quad \& \quad (B_3,a,B_3') \in \delta'_{\sigma,j} \quad \& \quad \beta = 2)]\}, \\
\delta_\sigma^{(6)} \quad := \quad & \{(((A_1,\dots,A_{2^{n+1}}),B_1,B_2,(j,3),(B_3,\beta)),\varepsilon, \\
& ((A_1,\dots,A_{2^{n+1}}),B_1,B_2,(j+1,1),(B_3',\beta'))) \in Q'_\sigma \times \{\varepsilon\} \times Q'_\sigma \mid \\
& B_1 \cap B_2 = A_{j+1} \quad \& \quad [(B_3,\beta) = (B_3',\beta') \in \{(\emptyset,1),(\emptyset,3)\} \quad \vee \\
& (j \notin J \quad \& \quad (B_3,\beta) = (B_3',\beta') \in Q'_{\sigma,0} \times \{2\}) \quad \vee \\
& (j \in J \quad \& \quad (B_3,\beta) = (\emptyset,1) \quad \& \quad (B_3',\beta') = (\{(q_j',q_j',0,0)\},2)) \quad \vee \\
& (j \in J \quad \& \quad (B_3,\beta) \in Q'_{F,\sigma,j} \times \{2\} \quad \& \quad (B_3',\beta') = (\emptyset,3))]\}.
\end{aligned}$$

The ε-NFA M'_σ works as desired above. Thus, it is easy to establish that this ε-NFA recognizes the language $\{x \in \Sigma^* \mid \text{there is no } \pi \in \Pi \text{ consuming } x \text{ so that } \sigma \in \varphi(\pi)\}$. Of course, the construction of M'_σ is effective (see Theorem 2.3 for details). The following fact states upper bounds for $\#Q'_\sigma$ and $\|\bar{M}'_\sigma\|$ and for some other useful parameters of $M'_{\sigma,j}$ $(j \in J)$, M'_σ and $\bar{M}'_\sigma$.

FACT 2.10. *The following assertions are true.*

(i) $\#Q'_{\sigma,0} = 2^{\#Q_{\sigma,0}} \le 2^{2n^8\cdot(1+\mathrm{diff}(\delta))\cdot(2+\mathrm{iml}(\delta))}$.

(ii) For all $j \in J$: $\#\delta'_{\sigma,j} = \#Q'_{\sigma,0} \cdot \#\Sigma$, $\|M'_{\sigma,j}\| \le 2 \cdot \#Q'_{\sigma,0} \cdot \|M\| \le 2^{3n^8\cdot\|M\|^2}$.

(iii) $\#Q_\sigma \le \#Q'_\sigma \le 2^{n\cdot(2^{n+1}+8)} \cdot 2^{2n^8\cdot(1+\mathrm{diff}(\delta))\cdot(2+\mathrm{iml}(\delta))}$.

(iv) $\#\delta'_\sigma \le (\#Q'_\sigma)^2 \cdot (1+\#\Sigma)$, $\#\bar{\delta}'_\sigma \le (\#Q'_\sigma)^2 \cdot \#\Sigma$,
$\max\{\|M'_\sigma\|, \|\bar{M}'_\sigma\|\} \le 2 \cdot (\#Q'_\sigma)^2 \cdot \|M\| \le 2^{n\cdot(2^{n+2}+17)} \cdot 2^{5n^8\cdot\|M\|^2}$.

Proof. The assertions (i) and (ii) follow from the definition of $M'_{\sigma,j}$ $(j \in J)$ and from Fact 2.9(i). Using (i), the assertions (iii) and (iv) are obvious from the definition of M'_σ and $\bar{M}'_\sigma$. □

This completes the proof of Theorem 2.2. □

Proof of Theorem 2.3. Let $M = (Q,\Sigma,\Delta,\delta,Q_I,Q_F)$ and $n = \#Q$ be as in the Theorems 2.1 and 2.2. Let S and $\bar{M}_\sigma, \bar{M}'_\sigma$ $(\sigma \in S)$ be as in the proofs of these theorems. The NGSMs $\bar{M}_\sigma$ $(\sigma \in S)$ and the NFAs $\bar{M}'_\sigma$ $(\sigma \in S)$ are playing the role of $M_1,\dots,M_N$ in Theorem 2.1 and of $M_1',\dots,M_N'$ in Theorem 2.2, respectively, where $N = \#S = O(2^{\mathrm{poly}\|M\|})$. Therefore, all we have to do here is to show that S and each $\bar{M}_\sigma$ and $\bar{M}'_\sigma$ $(\sigma \in S)$ can be constructed in $\mathrm{DTIME}(2^{2^{\mathrm{lin}\|M\|}})$.

We recall from the proof of Theorem 2.1 the definition of $Q_1,\dots,Q_k$ and of $n_i := \#Q_i$ $(i \in [k])$. In order to prepare the construction of S and $\bar{M}_\sigma, \bar{M}'_\sigma$ $(\sigma \in S)$ we have to carry out the following procedure.

ALGORITHM 1.

1. Remove all useless states from M. Let without loss of generality M be trim.
2. Compute $Q_1,\dots,Q_k$ and $n_1,\dots,n_k$.
3. Compute $\mathrm{im}(\delta)$, $\mathrm{iml}(\delta)$, $\mathrm{diff}(\delta)$ and $(n^3-1)\cdot\mathrm{diff}(\delta)$.
4. Compute $Q_I \cap Q_i$, $Q_F \cap Q_i$ (for all $1 \le i \le k$) and $(n^2 \cdot \sum_{i=i_1}^{i_2} n_i - 1)\cdot\mathrm{diff}(\delta)$ (for all $1 \le i_1 \le i_2 \le k$).

Note that in 1 and 2 we apply depth-first search, the computation of strongly connected components and topological sort (see [AHU74]) to directed graphs attached to M. It is easy to see that Algorithm 1 can be realized in $O(\mathrm{poly}\|M\|)$ time. Using the results of Algorithm 1, all members of the set S can be generated in $O(\#S \cdot \mathrm{poly}\|M\|)$ time.

Let $\sigma \in S$. Let $\sigma = (q_I,(j_1,p'_{j_1},\sigma_{j_1},q'_{j_1}),\dots,(j_l,p'_{j_l},\sigma_{j_l},q'_{j_l}),q_F)$ and J, J_1, J_2, $\tilde{n}_j$ $(j \in J)$, z_j $(j \in J_1)$ and $(t_j,\tilde{p}_j,\tilde{q}_j)$ $(j \in J_2)$ be given as in step (3) of the proof of Theorem 2.1. We recall from there the NGSM $M_{\sigma,j} = (Q_{\sigma,0},\Sigma,\Delta,\delta_{\sigma,j},Q_{I,\sigma,j},Q_{F,\sigma,j})$

$(j \in J)$ and the NFT $M_\sigma = (Q_\sigma, \Sigma, \Delta, \delta_\sigma, Q_{I,\sigma}, Q_{F,\sigma})$. We recall from the proof of Theorem 2.2 the DFA $M'_{\sigma,j} = (Q'_{\sigma,0}, \Sigma, \delta'_{\sigma,j}, Q'_{I,\sigma,j}, Q'_{F,\sigma,j})$ $(j \in J)$ and the ε-NFA $M'_\sigma = (Q'_\sigma, \Sigma, \delta'_\sigma, Q'_{I,\sigma}, Q'_{F,\sigma})$.

In order to construct $M_{\sigma,j}$ (for all $j \in J$), M_σ and $\bar{M}_\sigma$ we may proceed as follows.

ALGORITHM 2.

1. Generate all members of $Q_{\sigma,0}$.
2. For all $j \in J$ do the following:
   (a) Generate all members of $Q_{I,\sigma,j}$ and $Q_{F,\sigma,j}$.
   (b) For all $(r,s) \in Q^2_{\sigma,0}$ and for all $(p,a,z,q) \in \delta$, decide whether $(r,a,z,s) \in \delta_{\sigma,j}$.
3. Using $Q_{\sigma,0}$, generate all members of Q_σ, $Q_{I,\sigma}$, and $Q_{F,\sigma}$.
4. Using $M_{\sigma,j}$ $(j \in J)$, decide for all $(r,s) \in Q^2_\sigma$ whether $(r,\varepsilon,\varepsilon,s) \in \delta_\sigma$, and for all $(p,a,z,q) \in \delta$ whether $(r,a,z,s) \in \delta_\sigma$.
5. Using M_σ, construct the NGSM $\bar{M}_\sigma$.

Note that in 5 we apply again depth-first search. Recall that $\#J < n$ and $\|M_{\sigma,j}\| = O(\text{poly}\|M\|)$ $(j \in J)$. It is easy to see that Algorithm 2 can be realized using $O(\#Q^2_{\sigma,0} \cdot \text{poly}\|M\|)$ time for 1–2, $O(\#Q^2_\sigma \cdot \text{poly}(2^{\|M\|}))$ time for 3–4, and $O(\text{poly}\|M_\sigma\|)$ time for 5. According to Fact 2.9, all these bounds are of order $O(2^{2^{\text{lin}\|M\|}})$.

In order to construct $M'_{\sigma,j}$ (for all $j \in J$), M'_σ and $\bar{M}'_\sigma$ we may proceed as follows.

ALGORITHM 3.

1. Using $Q_{\sigma,0}$, generate all members of $Q'_{\sigma,0}$.
2. For all $j \in J$ do the following.
   (a) Using $M_{\sigma,j}$, generate all members of $Q'_{I,\sigma,j}$, and $Q'_{F,\sigma,j}$.
   (b) Using $\delta_{\sigma,j}$, determine for all $(B,a) \in Q'_{\sigma,0} \times \Sigma$ the unique $B' \in Q'_{\sigma,0}$ so that $(B,a,B') \in \delta'_{\sigma,j}$.
3. Using $Q'_{\sigma,0}$, generate all members of Q'_σ, $Q'_{I,\sigma}$, and $Q'_{F,\sigma}$.
4. Using $M'_{\sigma,j}$ $(j \in J)$, decide for all $(r,a,s) \in Q'_\sigma \times (\Sigma \cup \{\varepsilon\}) \times Q'_\sigma$ whether $(r,a,s) \in \delta'_\sigma$.
5. Using M'_σ, construct the NFA $\bar{M}'_\sigma$.

Note that in 5 we apply again depth-first search. Recall that $\#J < n$, $\|M_{\sigma,j}\| = O(\text{poly}\|M\|)$ and $\|M'_{\sigma,j}\| = O(2^{\text{poly}\|M\|})$ $(j \in J)$. It is easy to see that Algorithm 3 can be realized using $O(\#Q'_{\sigma,0} \cdot \text{poly}\|M\|)$ time for 1–2, $O((\#Q'_\sigma)^2 \cdot 2^{\text{poly}\|M\|})$ time for 3–4, and $O(\text{poly}\|M'_\sigma\|)$ time for 5. According to Fact 2.10, all these bounds are of order $O(2^{2^{\text{lin}\|M\|}})$.

This completes the proof of Theorem 2.3. □

3. Deciding the equivalence of finite-valued NGSMs. Let M_0 and M be NGSMs with coinciding input (and output) alphabets so that M is finite valued. Applying the Theorems 2.1–2.3 to M we can construct in DTIME$(\|M_0\|^{2^{\text{poly}\|M\|}})$ a nondeterministic finite 1-turn $2N$-counter machine (see [GI81]), where $N = O(2^{\text{poly}\|M\|})$ is as in Theorem 2.1, which has $O(\|M_0\|^{2^{\text{poly}\|M\|}})$ transitions, and which recognizes the language $L_0 := \{x \in L(M_0) \mid \text{there is a value } z \text{ for } x \text{ in } M_0 \text{ so that } (x,z) \notin T(M)\}$. Applying a result of Gurari and Ibarra [GI81] to this counter machine we obtain the following.

(1) If $T(M_0)$ is not included in $T(M)$, then there is an $x \in L_0$ such that $|x|$ is of order $O(\|M_0\|^{2^{\text{poly}\|M\|}})$.

(2) It is decidable in DTIME$(\|M_0\|^{2^{\text{poly}\|M\|}})$ whether or not $T(M_0)$ is included in $T(M)$.

(1) means: Noninclusion of $T(M_0)$ in $T(M)$ can be detected by a witness of length $O(\|M_0\|^{2^{\mathrm{poly}\|M\|}})$. We remark that the bounds in (1) and (2) are rather "machine oriented." It seems lengthy to analyse them exactly.

In this section we present a combinatorial word lemma (Lemma 3.6), which yields, together with the Theorems 2.1–2.2, an exact version of (1) (see Theorem 3.1). From this and from the Theorems 2.1–2.3 a proof of (2) can be easily derived (see Theorem 3.2).

The main result of this section is the following theorem.

THEOREM 3.1. *Let M_0 and M be* NGSMs *with coinciding input* (*and output*) *alphabets so that M is finite valued. If $T(M_0)$ is not included in $T(M)$, then there is a witness $(x, z) \in T(M_0)\backslash T(M)$ such that $|x|$ is of order $O(\|M_0\| \cdot 2^{2^{\mathrm{poly}\|M\|}})$. In detail, let $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$, $n_0 := \#Q_0$, $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$, and $n := \#Q$, then $|x| < e \cdot N! \cdot n_0 \cdot \bar{n}^N$, where*

$$N \leq \begin{cases} (5^{1/2} \cdot 3^{2/3})^n \cdot n^{4\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+3)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{diff}(\delta)} & \text{if } \#\Delta > 1, \\ (5^{1/2} \cdot 3^{4/3})^n \cdot n^{6\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+2)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1 + \mathrm{diff}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1, \end{cases}$$

and $\bar{n} \leq 2^{n\cdot(2^{n+1}+8)} \cdot 2^{2n^8\cdot(1+\mathrm{diff}(\delta))\cdot(2+\mathrm{iml}(\delta))}$.

We will also show that the Theorems 2.1–2.3 and 3.1 imply the following theorem.

THEOREM 3.2. *Let M_0 and M be* NGSM*s with coinciding input* (*and output*) *alphabets so that M is finite valued. It is decidable in* DTIME $(\|M_0\|^{2^{\mathrm{poly}\|M\|}})$ *whether or not $T(M_0)$ is included in $T(M)$.*

We want to point out that the basic problem of the Theorems 3.1 and 3.2 is to verify that some $(x, z) \in T(M_0)$ does *not* belong to $T(M)$. By reduction, the Theorems 3.1 and 3.2 can be generalized to NFTs (Theorems 4.2 and 4.3). Going with the title of this section we state the following immediate corollary to Theorem 3.2.

THEOREM 3.3 (see [CK86]). *Let M_0 and M be finite-valued* NGSMs *with coinciding input* (*and output*) *alphabets. It is decidable in* DTIME $(2^{2^{\mathrm{poly}(\|M_0\|+\|M\|)}})$ *whether or not M_0 and M are equivalent.*

Note. Theorem 3.1 alone implies a

$$\mathrm{DTIME}\,(2^{\|M_0\|^{2^{\mathrm{poly}\|M\|}}}) \quad \text{and} \quad \mathrm{DSPACE}\,(\|M_0\|^{2^{\mathrm{poly}\|M\|}})$$

algorithm for the decision problem in Theorem 3.2. This algorithm just tests all

$$O(2^{\|M_0\|^{2^{\mathrm{poly}\|M\|}}})$$

candidates (x, z) with "small" $|x|$, as proposed by Theorem 3.1, on membership in $T(M_0)$ $\backslash T(M)$.

In [N79] Nozaki constructed for any given $n \geq 6$ two nonequivalent NFAs having n states and 4 input symbols such that the shortest word in the symmetric difference of the languages recognized by them has length at least $2^{n-4} + 1$. From this follows that each improvement of the upper bound for $|x|$ stated in Theorem 3.1 has to stop at $2^{n-4} + 2$, even if $\#\Delta = 1$ and M_0 and M are single valued. If $\#\Delta = 1$, then we conjecture that the upper bound for $|x|$ in Theorem 3.1 can be improved to $O(2^{\mathrm{poly}(\|M_0\|+\|M\|)})$.

Since the equivalence problem for NFAs is PSPACE-complete (see [GJ79]), the decision problems in the Theorems 3.2 and 3.3 are PSPACE-hard, even if M_0 and M are single valued and have only one output symbol. It is an open question whether these decision problems can be solved in DTIME $(2^{\mathrm{poly}(\|M_0\|+\|M\|)})$, even in the case of a unitary output alphabet.

In order to prove the Theorems 3.1 and 3.2 we first of all introduce a little Ramsey theory including an upper bound for certain Ramsey numbers (Proposition 3.4). After that, we will establish a combinatorial word lemma (Lemma 3.6), which generalizes a word lemma by Schützenberger [Sch76] (Lemma 3.5) and may be of interest. Finally, we will use the Theorems 2.1–2.3 and Lemma 3.6 in order to prove the Theorems 3.1 and 3.2, successively.

For a general background on Ramsey theory we refer to [GRS80]. Let $A \subseteq \mathbb{N}$, then $\binom{A}{2}$ denotes the set $\{B \subseteq A \mid \#B = 2\}$. Let $N \in \mathbb{N}$, then the *Ramsey number* $R(3;N)$ denotes the minimal $l \in \mathbb{N}$ having the following property.

($*$) Given any mapping $\varphi\colon \binom{[l]}{2} \longrightarrow [N]$, there is a set $A \subseteq [l]$ such that $\#A = 3$ and, for some $i \in [N]$, $\varphi(\binom{A}{2}) = \{i\}$.

By Ramsey's theorem, such an l exists. Note that every $l \geq R(3;N)$ has property ($*$) and that, for every $N' \geq N$, $R(3;N') \geq R(3;N)$. As exact values, it is only known that $R(3;1) = 3$, $R(3;2) = 6$, and $R(3;3) = 17$ (see [Be79, Chap. 19.3]). In general, $R(3;N) \geq (3^N + 3)/2$ (see [GRS80, Chap. 4.5]). Here, we need the following upper bound for $R(3;N)$.

PROPOSITION 3.4. *Let $N \in \mathbb{N}$. Then, $R(3;N) \leq 1 + N! \cdot \sum_{i=0}^{N}(1/i!) \leq \lceil e \cdot N! \rceil$.*

Proof. See [Be79, Chap. 19.3]. □

LEMMA 3.5 (Schützenberger [Sch76]). *Let Δ be some nonempty finite set. Let $z_1, z_2, z_3, z_4, z_1', z_2', z_3', z_4' \in \Delta^*$ such that $z_1z_4 = z_1'z_4'$, $z_1z_2z_4 = z_1'z_2'z_4'$, and $z_1z_3z_4 = z_1'z_3'z_4'$. Then, $z_1z_2z_3z_4 = z_1'z_2'z_3'z_4'$.*

Proof. Because of symmetry we may assume that $|z_1| \geq |z_1'|$. $z_1z_4 = z_1'z_4'$ implies that, for some $z \in \Delta^*$, $z_1 = z_1'z$ and $zz_4 = z_4'$. Thus, $zz_2 = z_2'z$ and $zz_3 = z_3'z$. In summary, we know: $z_1z_2z_3z_4 = z_1'zz_2z_3z_4 = z_1'z_2'zz_3z_4 = z_1'z_2'z_3'zz_4 = z_1'z_2'z_3'z_4'$. □

LEMMA 3.6. *Let Δ be some nonempty finite set, and let K be a nonempty finite subset of $\mathbb{N}$. Let $l \geq R(3;\#K)$, let $\varphi\colon \binom{[l]}{2} \longrightarrow 2^K\backslash\{\emptyset\}$, and let $z_{i,j} \in \Delta^*$ ($i \in K \cup \{0\}$, $j \in [l+1]$) such that, for all $1 \leq j_1 < j_2 \leq l$ and for all $i \in \varphi(\{j_1, j_2\})$,*

$$z_{0,1} \cdots z_{0,j_1} z_{0,j_2+1} \cdots z_{0,l+1} = z_{i,1} \cdots z_{i,j_1} z_{i,j_2+1} \cdots z_{i,l+1}.$$

Then, for some $i_0 \in K$, $z_{0,1} \ldots z_{0,l+1} = z_{i_0,1} \ldots z_{i_0,l+1}$.

Proof. According to the definition of $R(3;\#K)$, there are $i_0 \in K$ and $1 \leq j_1 < j_2 < j_3 \leq l$ such that $i_0 \in \varphi(\{j_1,j_2\}) \cap \varphi(\{j_1,j_3\}) \cap \varphi(\{j_2,j_3\})$. Thus, we can apply Lemma 3.5 to $z_1 := z_{0,1} \ldots z_{0,j_1}$, $z_2 := z_{0,j_1+1} \ldots z_{0,j_2}$, $z_3 := z_{0,j_2+1} \ldots z_{0,j_3}$, $z_4 := z_{0,j_3+1} \ldots z_{0,l+1}$ and $z_1' := z_{i_0,1} \ldots z_{i_0,j_1}$, $z_2' := z_{i_0,j_1+1} \ldots z_{i_0,j_2}$, $z_3' := z_{i_0,j_2+1} \ldots z_{i_0,j_3}$, $z_4' := z_{i_0,j_3+1} \ldots z_{i_0,l+1}$, which yields

$$z_{0,1} \ldots z_{0,l+1} = z_1z_2z_3z_4 = z_1'z_2'z_3'z_4' = z_{i_0,1} \ldots z_{i_0,l+1}. \qquad \square$$

Proof of Theorem 3.1. Let $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ and $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be NGSMs with n_0 and n states, respectively, so that M is finite valued. If $L(M_0) \not\subseteq L(M)$, then the well-known subset construction (see, e.g., [HU79]) applied to an NFA recognizing $L(M)$ yields a witness $(x,z) \in T(M_0)\backslash T(M)$ such that $|x|$ is smaller than $n_0 \cdot 2^n$. Thus, we may assume that $L(M_0) \subseteq L(M)$. By Lemma A.1 we can apply the Theorems 2.1 and 2.2 to M. Let $M_1, \ldots, M_N$ and $M_1', \ldots, M_N'$ be the NGSMs and NFAs, respectively, the existence of which is claimed in these theorems.

Let $M_i = (Q_i, \Sigma, \Delta, \delta_i, Q_{I,i}, Q_{F,i})$, $n_i := \#Q_i$, $M_i' = (Q_i', \Sigma, \delta_i', Q_{I,i}', Q_{F,i}')$, $n_i' := \#Q_i'$, $\bar{n}_i := \max\{n_i, n_i'\}$ $(i = 1, \dots, N)$, and let $\bar{n} := \max\{\bar{n}_1, \dots, \bar{n}_N\}$. Then, $M_1, \dots, M_N$ are single valued, $T(M)$ equals $T(M_1) \cup \dots \cup T(M_N)$, M_i' recognizes $\Sigma^* \backslash L(M_i)$ $(i = 1, \dots, N)$, and N and $\bar{n}$ are bounded as desired.

Using Proposition 3.4, the theorem follows from the assertion $(*)$.

$(*)$ If $T(M_0)$ is not included in $T(M)$ and if $(x, z) \in T(M_0) \backslash \bigcup_{i=1}^N T(M_i)$ such that $|x|$ is minimal, then $|x|$ is smaller than $(R(3; N) - 1) \cdot n_0 \cdot \prod_{i=1}^N \bar{n}_i$.

Proof of $(*)$. Let (x, z) be as in $(*)$. We assume that $|x| \geq (R(3; N) - 1) \cdot n_0 \cdot \prod_{i=1}^N \bar{n}_i$. Define $K := \{i \in [N] \mid x \in L(M_i)\}$ and $K' := [N] \backslash K$. Since $x \in L(M_0) \subseteq L(M)$, K is nonempty. Let π_0 be an accepting path in M_0 realizing (x, z). Let π_i (for all $i \in K$) and π_i' (for all $i \in K'$) be accepting paths in M_i and M_i', respectively, all consuming x. Since $|x| \geq (R(3; N) - 1) \cdot n_0 \cdot \prod_{i=1}^N \bar{n}_i$, there are at least $R(3; N)$ different prefixes $y_1, y_1 y_2, \dots, y_1 \dots y_l$ of $x = y_1 \dots y_l y_{l+1}$ and states $q_i \in Q_i$ (for all $i \in K \cup \{0\}$) and $q_i' \in Q_i'$ (for all $i \in K'$) such that the following holds.

- For each $i \in K \cup \{0\}$: Consider the uniquely determined paths $\pi_{i,1}, \dots, \pi_{i,l+1}$ in M_i consuming $y_1, \dots, y_{l+1}$, respectively, such that $\pi_i = \pi_{i,1} \circ \dots \circ \pi_{i,l+1}$. Then, each of the paths $\pi_{i,2}, \dots, \pi_{i,l}$ leads from q_i to q_i.
- For each $i \in K'$: Consider the uniquely determined paths $\pi_{i,1}', \dots, \pi_{i,l+1}'$ in M_i' consuming $y_1, \dots, y_{l+1}$, respectively, such that $\pi_i' = \pi_{i,1}' \circ \dots \circ \pi_{i,l+1}'$. Then, each of the paths $\pi_{i,2}', \dots, \pi_{i,l}'$ leads from q_i' to q_i'.

For all $(i, j) \in (K \cup \{0\}) \times [l+1]$ let $z_{i,j} \in \Delta^*$ be the word produced by $\pi_{i,j}$. Consider the mapping $\varphi \colon \binom{[l]}{2} \longrightarrow 2^K$, where for all $1 \leq j_1 < j_2 \leq l$

$$\varphi(\{j_1, j_2\}) := \{i \in K \mid z_{0,1} \dots z_{0,j_1} z_{0,j_2+1} \dots z_{0,l+1} = z_{i,1} \dots z_{i,j_1} z_{i,j_2+1} \dots z_{i,l+1}\}.$$

Since $|x|$ is minimal and $M_1, \dots, M_N$ are single valued, each $\varphi(\{j_1, j_2\})$ is nonempty. Thus, we can apply Lemma 3.6 to K, l, φ, and to $z_{i,j} \in \Delta^*$ $(i \in K \cup \{0\}, j \in [l+1])$, which yields an $i_0 \in K$ such that $(x, z) = (x, z_{0,1} \dots z_{0,l+1}) = (x, z_{i_0,1} \dots z_{i_0,l+1}) \in T(M_{i_0})$ (Contradiction!) Therefore, $|x|$ is smaller than $(R(3; N) - 1) \cdot n_0 \cdot \prod_{i=1}^N \bar{n}_i$. □

Note that in previous versions of the proof of Theorem 3.1 (see [W88]) Ramsey numbers were used only in an implicit way. We want to point out that from work by Lisovik ([L79a, Lem. 1.1], see also [L80, Thm. 1]) it can be derived that Schützenberger's lemma (Lemma 3.5) implies the above assertion $(*)$ with the upper bound $3 \cdot N! \cdot n_0 \cdot \prod_{i=1}^N \bar{n}_i$ for $|x|$. Lisovik's proof methods are basically similar to ours. The above assertion $(*)$ can be also proved using an approach due to Turakainen [T89], which is based on some results on $\mathcal{Q}$-rational languages [T88, §2]. By this alternative proof, the upper bound for $|x|$ in Theorem 3.1 can be improved to $3^N \cdot n_0 \cdot \bar{n}^N$.

Proof of Theorem 3.2. Let $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ and $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be NGSMs with n_0 and n states, respectively, so that M is finite valued. If $L(M_0) \not\subseteq L(M)$, then $T(M_0) \not\subseteq T(M)$. The above condition is decidable in DTIME $(\mathrm{poly}\|M_0\| \cdot 2^{\mathrm{lin}\|M\|})$ (see, e.g., [HU79]), so we may assume that $L(M_0) \subseteq L(M)$. By Lemma A.1 we can apply the Theorems 2.1 and 2.2 to M. Let $M_1, \dots, M_N$ and $M_1', \dots, M_N'$ be the NGSMs and NFAs, respectively, constructed by these theorems. Let $M_i = (Q_i, \Sigma, \Delta, \delta_i, Q_{I,i}, Q_{F,i})$, $n_i := \#Q_i$, $M_i' = (Q_i', \Sigma, \delta_i', Q_{I,i}', Q_{F,i}')$, $n_i' := \#Q_i'$, $\bar{n}_i := \max\{n_i, n_i'\}$ $(i = 1, \dots, N)$, and let $\bar{n} := \max\{\bar{n}_1, \dots, \bar{n}_N\}$. Then, $M_1, \dots, M_N$ are single valued, $T(M)$ equals $T(M_1) \cup \dots \cup T(M_N)$, M_i' recognizes $\Sigma^* \backslash L(M_i)$ $(i = 1, \dots, N)$, the quantities $N = O(2^{\mathrm{poly}\|M\|})$ and $\bar{n} = O(2^{2^{\mathrm{lin}\|M\|}})$ are bounded as in Theorem 3.1, and, for each $i \in [N]$, $\mathrm{im}(\delta_i)$ is included in $\mathrm{im}(\delta)$ and $\mathrm{diff}(\delta_i, \delta_0)$ is at most $\mathrm{diff}(\delta, \delta_0)$. The last two properties are stated in Fact 2.9.

Using Proposition 3.4, the assertion $(*)$ in the proof of Theorem 3.1 implies

$(\#)$ $T(M_0)$ is not included in $T(M)$ if and only if there is an accepting path π_0 in M_0 realizing some $(x, z_0) \in \Sigma^* \times \Delta^*$ and there are subsets K, K_1, K_2 of $[N]$ such that the following assertions are true.

- $K = K_1 \cup K_2$ and $K_1 \cap K_2 = \emptyset$.
- For each $i \in K$: There is an accepting path π_i in M_i and a word $z_i \in \Delta^*$ so that π_i realizes (x, z_i), $\operatorname{diff}(\pi_i, \pi_0)$ is at most $3 \cdot N! \cdot n_0 \cdot \bar{n}^N \cdot \operatorname{diff}(\delta_i, \delta_0)$, $|z_i|$ is distinct from $|z_0|$ (if $i \in K_1$) and z_i and z_0 differ at some position $j \in \min\{|z_i|, |z_0|\}$ (if $i \in K_2$).
- For each $i \in [N] \backslash K$: There is an accepting path π_i' in M_i' consuming x.

Note. $(\#)$ remains true if we omit all upper bounds for $\operatorname{diff}(\pi_i, \pi_0)$.

We will transform the assertion $(\#)$ into a simple characterization of the noninclusion at issue, which is based on graphs. For this, let us consider $K, K_1, K_2, K' \subseteq [N]$ so that $K = K_1 \cup K_2$, $K_1 \cap K_2 = \emptyset$ and $K' = [N] \backslash K$. Then, following [W90, §3], we construct the directed graph $G_{K_1,K_2} = (V, E)$ (for comments see below):

$$\begin{aligned}
V \;:=\;& Q_0 \times \prod_{i \in K_1} (Q_i \times Z_i) \times \prod_{i \in K_2} (Q_i \times (\Delta \,\dot{\cup}\, \{1,2\}) \times Z_i) \times \prod_{i \in K'} Q_i', \\
& \text{where} \quad Z_i := \{t \in \mathcal{Z} \mid |t| \le 3 \cdot N! \cdot n_0 \cdot \bar{n}^N \cdot \operatorname{diff}(\delta_i, \delta_0)\} \quad (i \in K), \\
E \;:=\;& \{((p_0, ((p_i, t_i))_{i \in K_1}, ((p_i, b_i, t_i))_{i \in K_2}, (p_i')_{i \in K'}), \\
& (q_0, ((q_i, t_i'))_{i \in K_1}, ((q_i, b_i', t_i'))_{i \in K_2}, (q_i')_{i \in K'}) \in V^2 \mid \\
& \exists a \in \Sigma \quad \exists (z_i)_{i \in K \cup \{0\}} \in \prod_{i \in K \cup \{0\}} \Delta^* : \\
& [\forall i \in K \cup \{0\} \colon (p_i, a, z_i, q_i) \in \delta_i] \quad \& \quad [\forall i \in K' \colon (p_i', a, q_i') \in \delta_i'] \quad \& \\
& [\forall i \in K_1 \colon t_i' = t_i + |z_i| - |z_0|] \quad \& \quad [\forall i \in K_2 \colon \\
& (b_i = b_i' = 1 \quad \& \quad t_i' = t_i + |z_i| - |z_0|) \quad \vee \\
& (b_i = 1 \quad \& \quad \exists j \in [|z_i|] \colon \quad b_i' = z_i(j) \in \Delta \quad \& \quad t_i' = t_i + j - |z_0| > 0) \quad \vee \\
& (b_i = 1 \quad \& \quad \exists j \in [|z_0|] \colon \quad b_i' = z_0(j) \in \Delta \quad \& \quad t_i' = t_i + |z_i| - j < 0) \quad \vee \\
& (b_i = b_i' \in \Delta \quad \& \quad t_i' = t_i - |z_0| > 0) \quad \vee \\
& (b_i = b_i' \in \Delta \quad \& \quad t_i' = t_i + |z_i| < 0) \quad \vee \\
& ((b_i, t_i) \in \Delta \times [|z_0|] \quad \& \quad b_i \ne z_0(t_i) \quad \& \quad (b_i', t_i') = (2, 0)) \quad \vee \\
& ((b_i, -t_i) \in \Delta \times [|z_i|] \quad \& \quad b_i \ne z_i(-t_i) \quad \& \quad (b_i', t_i') = (2, 0)) \quad \vee \\
& (b_i = 1 \quad \& \quad \exists j \in [|z_i|] \colon \quad t_i + j \in [|z_0|] \quad \& \quad z_i(j) \ne z_0(t_i + j) \quad \& \\
& (b_i', t_i') = (2, 0)) \quad \vee \quad (b_i = b_i' = 2 \quad \& \quad t_i = t_i' = 0)]\}.
\end{aligned}$$

The construction of G_{K_1,K_2} requires some explanation.

First of all, a path π in G_{K_1,K_2}, regarding its behavior on the Q_i- and Q_i'-components of V, yields an $(N+1)$-tuple of paths π_i in M_i $(i \in K \cup \{0\})$ and π_i' in M_i' $(i \in K')$, all consuming the same word in Σ^*. Let $z_i \in \Delta^*$ be the word produced by π_i $(i \in K \cup \{0\})$. Let $i \in K_1$. Assume that, on the Z_i-component of V, the given path π leads from $t_i \in Z_i$ to $t_i' \in Z_i$. Then, $t_i' - t_i = |z_i| - |z_0|$. Let $i \in K_2$. Assume that, on the $(\Delta \cup \{1,2\}) \times Z_i$-component of V, π leads from $(1, 0)$ to $(2, 0)$. Then, z_i and z_0 differ at some position $j \in \min\{|z_i|, |z_0|\}$. In order to verify this, the path π "guesses" a position $j \in [|z_i|]$ (or, alternatively, a position $j \in [|z_0|]$), remembers the letter $z_i(j)$ (respectively, $z_0(j)$) until the jth letter of z_0 (respectively, z_i) can be seen, and verifies that $z_i(j)$ and $z_0(j)$ are distinct.

Conversely, let us consider an $(N+1)$-tuple $((\pi_i)_{i\in K\cup\{0\}}, (\pi'_i)_{i\in K'})$ of paths in M_i and in M'_i (as above), all consuming the same word in Σ^*. Let $z_i \in \Delta^*$ be the word produced by π_i ($i \in K \cup \{0\}$). Assume that, for all $i \in K$, $\mathrm{diff}(\pi_i, \pi_0)$ is at most $3 \cdot N! \cdot n_0 \cdot \bar{n}^N \cdot \mathrm{diff}(\delta_i, \delta_0)$ and that, for all $i \in K_2$, z_i and z_0 differ at some position $j \in \min\{|z_i|, |z_0|\}$. Then, the given $(N+1)$-tuple of paths induces a path π in G_{K_1,K_2} as follows: The behavior of π on the Q_i- and Q'_i-components of V is inherited from the given paths. On the Z_i-component of V ($i \in K_1$) π leads from 0 to $|z_i| - |z_0|$. On the $(\Delta \cup \{1,2\}) \times Z_i$-component of V ($i \in K_2$) π leads from $(1,0)$ to $(2,0)$; the detailed behavior of π on this component is as above.

We define the sets $V_{I,K_1,K_2}, V_{F,K_1,K_2} \subseteq V$ as follows.

$$\begin{aligned} V_{I,K_1,K_2} &:= Q_{I,0} \times \prod_{i\in K_1} (Q_{I,i} \times \{0\}) \times \prod_{i\in K_2} (Q_{I,i} \times \{(1,0)\}) \times \prod_{i\in K'} Q'_{I,i}, \\ V_{F,K_1,K_2} &:= Q_{F,0} \times \prod_{i\in K_1} (Q_{F,i} \times Z_i\backslash\{0\}) \times \prod_{i\in K_2} (Q_{F,i} \times \{(2,0)\}) \times \prod_{i\in K'} Q'_{F,i}. \end{aligned}$$

Then, the above discussion shows that (#) implies

(§) $T(M_0)$ is not included in $T(M)$ if and only if, for some $K_1, K_2 \subseteq [N]$ with $K_1 \cap K_2 = \emptyset$, there is a path in G_{K_1,K_2} leading from V_{I,K_1,K_2} to V_{F,K_1,K_2}.

Note. $\#V \le (6\cdot N!)^N \cdot n_0^{N+1} \cdot \bar{n}^{N\cdot(N+1)} \cdot (1+\mathrm{diff}(\delta,\delta_0))^N \cdot (2+\#\Delta)^N$ and $\#E \le \#V^2$, i.e., $\#V$ and $\#E$ are of order $O(\|M_0\|^{2^{\mathrm{poly}\|M\|}})$.

We sketch an algorithm that decides according to (§) whether or not $T(M_0)$ is included in $T(M)$:

1. Construct the NGSMs $M_1, \ldots, M_N$ and the NFAs $M'_1, \ldots, M'_N$.
2. For each $K_1, K_2 \subseteq [N]$ with $K_1 \cap K_2 = \emptyset$ do the following:
   (a) Construct G_{K_1,K_2}, V_{I,K_1,K_2} and V_{F,K_1,K_2}.
   (b) Decide whether or not there is a path in G_{K_1,K_2} leading from V_{I,K_1,K_2} to V_{F,K_1,K_2}.

Note that in step 2(b) we apply depth-first search (see [AHU74]) to G_{K_1,K_2}. According to Theorem 2.3, step 1 of the above algorithm requires DTIME $(2^{2^{\mathrm{lin}\|M\|}})$. The loop in step 2 is passed 3^N times. Let $K_1, K_2 \subseteq [N]$ such that $K_1 \cap K_2 = \emptyset$, and let $G_{K_1,K_2} = (V, E)$. Step 2 for (K_1, K_2) can be realized using $O(\mathrm{poly}(\#V + \|M_0\| \cdot \|M\|^N))$ time for 2(a) and $O(\mathrm{poly}(\#V + \#E))$ time for 2(b). All these bounds are of order $O(\|M_0\|^{2^{\mathrm{poly}\|M\|}})$.

This completes the proof of Theorem 3.2. □

4. Normalized finite transducers. In this section we generalize the main results of this paper (Theorems 2.1–2.3 and 3.1–3.2) to NFTs. The outcome is stated in the Theorems 4.1–4.3.

THEOREM 4.1. *Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be a finite-valued* NFT *with n states. Then, $O(2^{\mathrm{poly}\|M\|})$ many single-valued* NFTs *$M_1, \ldots, M_N$ and ε-*NFAs *$M'_1, \ldots, M'_N$ — each of size $O(2^{2^{\mathrm{lin}\|M\|}})$ — effectively exist such that $T(M)$ equals $T(M_1) \cup \ldots \cup T(M_N)$ and M'_i recognizes $\Sigma^*\backslash L(M_i)$ ($i = 1, \ldots, N$). In detail, N is at most*

$$\begin{cases} (5^{1/2}\cdot 3^{2/3})^n \cdot n^{4\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+3)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1+\mathrm{iml}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\mathrm{iml}(\delta)} & \text{if } \#\Delta > 1, \\ (5^{1/2}\cdot 3^{4/3})^n \cdot n^{6\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+2)} \cdot (\#\mathrm{im}(\delta))^{n-1} & \\ \cdot(1+\mathrm{iml}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1. \end{cases}$$

The NFTs $M_1, \dots, M_N$, *and the* ε-NFAs $M'_1, \dots, M'_N$, *can be constructed in* DTIME $(2^{2^{\text{lin}\|M\|}})$.

THEOREM 4.2. *Let* M_0 *and* M *be* NFTs *with coinciding input (and output) alphabets so that* M *is finite valued. If* $T(M_0)$ *is not included in* $T(M)$, *then there is a witness* $(x, z) \in T(M_0)\backslash T(M)$ *such that* $|x|$ *is of order* $O(\|M_0\| \cdot 2^{2^{\text{poly}\|M\|}})$. *In detail, let* $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$, $n_0 := \#Q_0$, $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$, *and* $n := \#Q$, *then* $|x| < e \cdot N! \cdot n_0 \cdot \bar{n}^N$, *where*

$$N \leq \begin{cases} (5^{1/2} \cdot 3^{2/3})^n \cdot n^{4\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+3)} \cdot (\#\text{im}(\delta))^{n-1} & \\ \cdot(1 + \text{iml}(\delta))^{n-1} \cdot \#\Delta^{2\cdot(n^3-1)\cdot\text{iml}(\delta)} & \text{if } \#\Delta > 1, \\ (5^{1/2} \cdot 3^{4/3})^n \cdot n^{6\cdot(n-1)} \cdot 2^{(n-1)\cdot(n+2)} \cdot (\#\text{im}(\delta))^{n-1} & \\ \cdot(1 + \text{iml}(\delta))^{2\cdot(n-1)} & \text{if } \#\Delta = 1, \end{cases}$$

and $\bar{n} \leq 2^{n\cdot(2^{n+1}+8)} \cdot 2^{2n^8\cdot(2+\text{iml}(\delta))^2}$.

THEOREM 4.3. *Let* M_0 *and* M *be* NFTs *with coinciding input (and output) alphabets so that* M *is finite valued. It is decidable in* DTIME $(\|M_0\|^{2^{\text{poly}\|M\|}})$ *whether or not* $T(M_0)$ *is included in* $T(M)$.

We prove the Theorems 4.1–4.3 by reduction to the Theorems 2.1–2.3 and 3.1–3.2. For this reduction we adopt from [W90, §5] the criterion (ε-IV) and the two subsequent propositions.

Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be an NFT. The criterion (ε-IV) reads as follows.

(ε-IV) There is a useful state $q \in Q$ such that, for some word $\tilde{v} \in \Delta^+$, $(q, \varepsilon, \tilde{v}, q) \in \hat{\delta}$.

PROPOSITION 4.4 (see [W90, §5]). *Let* $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ *be an* NFT *with* n *states. Then, the following assertions are true.*

(i) *If* M *complies with* (ε-*IV*), *then some word in* $\Sigma^{\leq 2\cdot(n-1)}$ *has infinite valuedness in* M.

(ii) *If* M *complies with* (ε-*IV*), *then it is infinite valued.*

(iii) *It is decidable in* DTIME (poly$\|M\|$), *whether or not* M *complies with* (ε-*IV*).

Proof. If M complies with (ε-IV), then $(u, \tilde{u}), (w, \tilde{w}) \in \Sigma^* \times \Delta^*$ and $\tilde{v} \in \Delta^+$ exist such that uw has length at most $2 \cdot (n-1)$ and, for all $i \in \mathcal{N}$, $\tilde{u}\tilde{v}^i\tilde{w}$ is a value for uw in M. Thus, $\text{val}(M) = \text{val}_M(uw) = \infty$. This implies (i) and (ii). For assertion (iii) we refer to the proof of Theorem 5.3 in [W90]. □

PROPOSITION 4.5 (see [W90, Lem. 5.5]). *Let* $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ *be an* NFT *with* n *states, which does not comply with* (ε-IV). *Then, an* NGSM $M' = (Q, \Sigma', \Delta, \delta', Q_I, Q_F)$, *where* $\Sigma' := \Sigma \dot{\cup} \{a_0\}$, *effectively exists such that the following assertions are true.*

(i) $\text{im}(\delta') = \text{im}(\delta)$, $\text{diff}(\delta') \leq \text{iml}(\delta') = \text{iml}(\delta)$,
$\|\delta\| \leq \|\delta'\| \leq \|\delta\| + n$, $\|M\| < \|M'\| \leq \|M\| + n + 1$.

(ii) M' *has the same valuedness as* M.

(iii) $\forall m \in \mathcal{N} \quad \forall x_1, \dots, x_m \in \Sigma \quad \forall \lambda_1, \dots, \lambda_{m+1} \in \mathcal{N} \quad \forall z \in \Delta^*$:
$(a_0^{\lambda_1}x_1 \dots a_0^{\lambda_m}x_m a_0^{\lambda_{m+1}}, z) \in T(M') \implies (x_1 \dots x_m, z) \in T(M)$.

(iv) $\forall m \in \mathcal{N} \quad \forall x_1, \dots, x_m \in \Sigma \quad \forall z \in \Delta^* \quad \forall \lambda_1, \dots, \lambda_{m+1} \geq n-1$:
$(x_1 \dots x_m, z) \in T(M) \implies (a_0^{\lambda_1}x_1 \dots a_0^{\lambda_m}x_m a_0^{\lambda_{m+1}}, z) \in T(M')$.

(v) M' *can be constructed in* DTIME(poly$\|M\|$).

Proof. We construct the NGSM $M' = (Q, \Sigma', \Delta, \delta', Q_I, Q_F)$:

$$\begin{aligned} \Sigma' &:= \Sigma \dot{\cup} \{a_0\}, \\ \delta' &:= (\delta \cap Q \times \Sigma \times \Delta^* \times Q) \cup \{(p, a_0, z, q) \mid (p, \varepsilon, z, q) \in \delta\} \\ &\quad \cup \{(q, a_0, \varepsilon, q) \mid q \in Q\}. \end{aligned}$$

The assertions (i), (iii), and (v) are obvious. The assertion (iv) can be proved as Claim 2 in the proof of Lemma 5.5 in [W90]. The assertion (ii) immediately follows from (iii) and (iv). □

Proof of Theorem 4.1. Let $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ be a finite-valued NFT with n states. By Proposition 4.4(ii), M does not comply with (ε-IV). Let $M^{(1)} = (Q, \Sigma^{(1)}, \Delta, \delta^{(1)}, Q_I, Q_F)$, where $\Sigma^{(1)} := \Sigma \,\dot\cup\, \{a_0\}$, be the finite-valued NGSM of size $O(\|M\|)$ which is attached to M in Proposition 4.5. $M^{(1)}$ can be constructed in DTIME (poly$\|M\|$). Applying the Theorems 2.1 and 2.2 to $M^{(1)}$ we obtain single-valued NGSMs $M_1^{(1)}, \ldots, M_N^{(1)}$ and NFAs $M_1^{(2)}, \ldots, M_N^{(2)}$ — each of size $O(2^{2^{\mathrm{lin}\|M^{(1)}\|}})$ — such that $T(M^{(1)})$ equals $T(M_1^{(1)}) \cup \ldots \cup T(M_N^{(1)})$, $M_i^{(2)}$ recognizes $(\Sigma^{(1)})^* \backslash L(M_i^{(1)})$ ($i = 1, \ldots, N$), and N is bounded as desired. By Theorem 2.3, $M_1^{(1)}, \ldots, M_N^{(1)}$ and $M_1^{(2)}, \ldots, M_N^{(2)}$ can be constructed in DTIME $(2^{2^{\mathrm{lin}\|M^{(1)}\|}})$.

Let $i \in [N]$. We attach to $M_i^{(1)} = (Q_i^{(1)}, \Sigma^{(1)}, \Delta, \delta_i^{(1)}, Q_{I,i}^{(1)}, Q_{F,i}^{(1)})$ the NFT $M_i = (Q_i, \Sigma, \Delta, \delta_i, Q_{I,i}, Q_{F,i})$, where $Q_i := Q_i^{(1)} \times \{0, \ldots, n-1\}$, $Q_{I,i} := Q_{I,i}^{(1)} \times \{0\}$, $Q_{F,i} := Q_{F,i}^{(1)} \times \{n-1\}$, and

$$\begin{aligned}\delta_i \;:=\;& \{((p, n-1), a, z, (q, 0)) \mid a \in \Sigma,\ (p, a, z, q) \in \delta_i^{(1)}\} \cup \\ & \{((p, j-1), \varepsilon, z, (q, j)) \mid j \in [n-1],\ (p, a_0, z, q) \in \delta_i^{(1)}\}.\end{aligned}$$

We attach to $M_i^{(2)} = (Q_i^{(2)}, \Sigma^{(1)}, \delta_i^{(2)}, Q_{I,i}^{(2)}, Q_{F,i}^{(2)})$ the ε-NFA $M_i' = (Q_i', \Sigma, \delta_i', Q_{I,i}', Q_{F,i}')$, where $Q_i' := Q_i^{(2)} \times \{0, \ldots, n-1\}$, $Q_{I,i}' := Q_{I,i}^{(2)} \times \{0\}$, $Q_{F,i}' := Q_{F,i}^{(2)} \times \{n-1\}$, and

$$\begin{aligned}\delta_i' \;:=\;& \{((p, n-1), a, (q, 0)) \mid a \in \Sigma,\ (p, a, q) \in \delta_i^{(2)}\} \cup \\ & \{((p, j-1), \varepsilon, (q, j)) \mid j \in [n-1],\ (p, a_0, q) \in \delta_i^{(2)}\}.\end{aligned}$$

Obviously, $\|M_i\| \le n \cdot \|M_i^{(1)}\|$ and $\|M_i'\| \le n \cdot \|M_i^{(2)}\|$. Thus, M_i and M_i' are of size $O(2^{2^{\mathrm{lin}\|M\|}})$. Given $M_i^{(1)}$, M_i can be constructed in DTIME (poly($n + \|M_i^{(1)}\|$)). Given $M_i^{(2)}$, M_i' can be constructed in DTIME (poly($n + \|M_i^{(2)}\|$)). Thus, $M_1, \ldots, M_N$ and $M_1', \ldots, M_N'$ can be constructed in DTIME $(2^{2^{\mathrm{lin}\|M\|}})$.

Proposition 4.5 and the definition of $M_1, \ldots, M_N$ and $M_1', \ldots, M_N'$ yields for all $x_1, \ldots, x_m \in \Sigma$, $z \in \Delta^*$ and $i \in [N]$:

$$\begin{aligned}(x_1 \ldots x_m, z) \in T(M) &\iff (a_0^{n-1}x_1 \ldots a_0^{n-1}x_m a_0^{n-1}, z) \in T(M^{(1)}) \quad \& \\ (x_1 \ldots x_m, z) \in T(M_i) &\iff (a_0^{n-1}x_1 \ldots a_0^{n-1}x_m a_0^{n-1}, z) \in T(M_i^{(1)}) \quad \& \\ x_1 \ldots x_m \in L(M_i') &\iff a_0^{n-1}x_1 \ldots a_0^{n-1}x_m a_0^{n-1} \in L(M_i^{(2)}).\end{aligned}$$

From this follows that $M_1, \ldots, M_N$ are all single valued, $T(M)$ equals $T(M_1) \cup \ldots \cup T(M_N)$, and M_i' recognizes $\Sigma^* \backslash L(M_i)$ ($i = 1, \ldots, N$). □

Proof of the Theorems 4.2 and 4.3. According to Proposition 4.4, we may assume that $M_0 = (Q_0, \Sigma, \Delta, \delta_0, Q_{I,0}, Q_{F,0})$ and $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ are NFTs with n_0 and n states, respectively, so that M is finite valued and M_0 and M do not comply with (ε-IV). Define $\tilde n := \max\{n, n_0\}$. Let $M' = (Q, \Sigma', \Delta, \delta', Q_I, Q_F)$, where $\Sigma' := \Sigma \,\dot\cup\, \{a_0\}$, be the finite-valued NGSM of size $O(\|M\|)$ which is attached to M in Proposition 4.5. M' can be constructed in DTIME (poly$\|M\|$).

We attach to M_0 the NGSM $M_0' = (Q_0', \Sigma', \Delta, \delta_0', Q_{I,0}', Q_{F,0}')$ where $Q_0' := Q_0 \times \{0, \ldots, \tilde{n}-1\}$, $Q_{I,0}' := Q_{I,0} \times \{0\}$, $Q_{F,0}' := Q_{F,0} \times \{\tilde{n}-1\}$, and

$$\begin{aligned}\delta_0' \; := \; & \{((p, \tilde{n}-1), a, z, (q, 0)) \mid a \in \Sigma,\ (p, a, z, q) \in \delta_0\} \cup \\ & \{((p, j-1), a_0, z, (q, j)) \mid j \in [\tilde{n}-1] \quad \& \\ & ((p, \varepsilon, z, q) \in \delta_0 \quad \vee \quad (p, z) = (q, \varepsilon) \in Q_0 \times \Delta^*)\}.\end{aligned}$$

Note that $\|M_0'\| \leq \tilde{n} \cdot (\|M_0\| + n_0)$. M_0' can be constructed in DTIME $(\mathrm{poly}(\|M_0\| + \|M\|))$. By definition, M_0' only recognizes words of the form $a_0^{\tilde{n}-1} x_1 \ldots a_0^{\tilde{n}-1} x_m a_0^{\tilde{n}-1}$, where $x_1, \ldots, x_m \in \Sigma$.

Recall that M_0 does not comply with (ε-IV). Proposition 4.5 and the definition of M_0' yields for all $x_1, \ldots, x_m \in \Sigma$ and $z \in \Delta^*$:

$$\begin{aligned}(x_1 \ldots x_m, z) \in T(M) \quad &\Longleftrightarrow \quad (a_0^{\tilde{n}-1} x_1 \ldots a_0^{\tilde{n}-1} x_m a_0^{\tilde{n}-1}, z) \in T(M') \quad \& \\ (x_1 \ldots x_m, z) \in T(M_0) \quad &\Longleftrightarrow \quad (a_0^{\tilde{n}-1} x_1 \ldots a_0^{\tilde{n}-1} x_m a_0^{\tilde{n}-1}, z) \in T(M_0').\end{aligned}$$

This implies that $T(M_0') \subseteq T(M')$ if and only if $T(M_0) \subseteq T(M)$. Thus, Theorem 3.2 applied to M_0' and M' yields Theorem 4.3.

In order to prove Theorem 4.2, let us assume that $T(M_0)$ is not included in $T(M)$, i.e., $T(M_0')$ is not included in $T(M')$. Applying Theorem 3.1 to M_0' and M', we obtain a witness $(x', z) \in T(M_0') \backslash T(M')$ such that $|x'| < e \cdot N! \cdot (n_0 \cdot \tilde{n}) \cdot \tilde{n}^N$ and N and $\tilde{n}$ are bounded as desired. Let $x_1, \ldots, x_m \in \Sigma$ so that $x' = a_0^{\tilde{n}-1} x_1 \ldots a_0^{\tilde{n}-1} x_m a_0^{\tilde{n}-1}$. Then, by the above properties of M and M_0, $(x_1 \ldots x_m, z)$ belongs to $T(M_0) \backslash T(M)$ and $|x_1 \ldots x_m| = m \leq |x'|/\tilde{n} < e \cdot N! \cdot n_0 \cdot \tilde{n}^N$. □

A. Appendix. For the sake of completeness, we state here three lemmas from [W90], which are fundamental for our proof of Theorem 2.1.

LEMMA A.1 [W90, Lems. 2.4 and 2.3(i)]. *Let M be an* NGSM. *If M complies with at least one of the criteria* (IV1), (IV2), *then it is infinite valued.*

LEMMA A.2 [W90, Lem. 2.9]. *Let* $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ *be an* NGSM. *Let* $A \in 2^Q \backslash \{\emptyset\}$ *and* $y \in \Sigma^+$ *such that the following holds: For all* $r \in A$ *there are* $s \in A$ *and* $z \in \Delta^*$ *so that* $(r, y, z, s) \in \delta$, *and for all* $s \in A$ *there are* $r \in A$ *and* $z \in \Delta^*$ *so that* $(r, y, z, s) \in \delta$. *Let* $p', q' \in A$. *Then, the following assertion is true: There are* $p, q \in A$, $p_1, p_2, p_3, p_4 \in Q$, $l_1, l_2 \geq 0$, *and* $z_1, \ldots, z_4, z_6, \ldots, z_9 \in \Delta^*$ *so that* $(p, y^{l_1}, z_1, p_1), (p_1, y, z_2, p_2), (p_2, y^{l_2}, z_3, p) \in \delta$, $(p, y^{l_1}, z_4, p'), (q', y^{l_2}, z_6, q) \in \delta$, *and* $(q, y^{l_1}, z_7, p_3), (p_3, y, z_8, p_4), (p_4, y^{l_2}, z_9, q) \in \delta$.

LEMMA A.3 [W90, Lem. 2.10]. *Let* $M = (Q, \Sigma, \Delta, \delta, Q_I, Q_F)$ *be a trim* NGSM *with n states, which does not comply with the criterion* (IV1). *Let* $U_1, U_2, U_3, U_4 \in 2^Q$ *so that* $Q = U_1 \mathbin{\dot\cup} U_2 \mathbin{\dot\cup} U_3 \mathbin{\dot\cup} U_4$ *and, for all* $1 \leq i < j \leq 4$, $\delta \cap U_j \times \Sigma^* \times \Delta^* \times U_i = \emptyset$. *Let* $A \in 2^Q \backslash \{\emptyset\}$, $y \in \Sigma^+$, $p' \in A \cap U_2$, *and* $q' \in A \cap U_3$ *such that the assertion of Lemma* A.2 *is true. Let* $p_1, p_2 \in Q$ *be taken from this assertion. If* π *and* $\tilde{\pi}$ *are paths in M consuming y and leading from* p' *to* q' *and from* p_1 *to* p_2, *respectively, then* $\mathrm{diff}(\pi, \tilde{\pi})$ *is at most* $((\#U_1 + \#U_2) \cdot (\#U_2 + \#U_3) \cdot (\#U_3 + \#U_4) - 1) \cdot \mathrm{diff}(\delta)$. *In particular, if* $\pi, \tilde{\pi}$ *produce some* $z, \tilde{z} \in \Delta^*$, *respectively, then* $||z| - |\tilde{z}|| \leq \mathrm{diff}(\pi, \tilde{\pi}) \leq (n^2 \cdot (\#U_2 + \#U_3) - 1) \cdot \mathrm{diff}(\delta)$.

Acknowledgment. During all stages of my work on this paper Helmut Seidl acted as an inexhaustible source for useful stimulations and valuable comments. The proposals by one of the referees led to an improved presentation. I highly appreciate all this help.

REFERENCES

[AHU74] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[AL78] A.V. Anisimov and L.P. Lisovik, *Equivalence problems for finite-automaton mappings into free and commutative semigroups*, Cybernetics, 14 (1978), pp. 321–327.

[Be79] C. Berge, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1979.

[B79] J. Berstel, *Transductions and Context-Free Languages*, Teubner, Stuttgart, 1979.

[BH77] M. Blattner and T. Head, *Single-valued a-transducers*, J. Comput. System Sci., 15 (1977), pp. 310–327.

[C90] K. Culik II, *New techniques for proving the decidability of equivalence problems*, Theoret. Comput. Sci., 71 (1990), pp. 29–45.

[CK86] K. Culik II and J. Karhumäki, *The equivalence of finite valued transducers* (*on* HDTOL *languages*) *is decidable*, Theoret. Comput. Sci., 47 (1986), pp. 71–84.

[GJ79] M. Garey and D. Johnson, *Computers and Intractability*, Freeman, San Francisco, CA, 1979.

[GRS80] R. Graham, B. Rothschild, and J. Spencer, *Ramsey Theory*, Wiley, New York, 1980.

[G68] T. Griffiths, *The unsolvability of the equivalence problem for Λ-free nondeterministic generalized machines*, J. ACM, 15 (1968), pp. 409–413.

[GI81] E. Gurari and O. Ibarra, *The complexity of decision problems for finite-turn multicounter machines*, J. Comput. System Sci., 22 (1981), pp. 220–229.

[GI83] ———, *A note on finite-valued and finitely ambiguous transducers*, Math. Systems Theory, 16 (1983), pp. 61–66.

[HW91] T. Head and A. Weber, *Deciding code related properties by means of finite transducers*, Proc. SEQUENCES 1991, to appear.

[HU79] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[I78] O. Ibarra, *The unsolvability of the equivalence problem for ε-free* NGSM*'s with unary input* (*output*) *alphabet and applications*, SIAM J. Comput., 7 (1978), pp. 524–532.

[K86] J. Karhumäki, *The equivalence of mappings on languages*, Proc. 4th IMYCS, 1986, in Lecture Notes in Comput. Sci., 281, Springer-Verlag, Berlin, Heidelberg, 1987, pp. 26–38.

[K87] ———, *On recent trends in formal language theory*, Proc. 14th ICALP, 1987, in Lecture Notes in Comput. Sci., 267, Springer-Verlag, Berlin, Heidelberg, 1987, pp. 136–162.

[L79a] L.P. Lisovik, *Equivalence problem for finitely ambiguous finite automata over semigroups*, Cybernetics, 15 (1979), pp. 463–467.

[L79b] ———, *The identity problem for regular events over the direct product of free and cyclic semigroups* (in Ukrainian and Russian), Dopovidi Akad. Nauk Ukrain. RSR Ser. A, 1979, No. 6, pp. 410–413.

[L80] ———, *Strict sets and finite semigroup coverings*, Cybernetics, 16 (1980), pp. 13–17.

[L83] ———, *Minimal undecidable identity problem for finite-automaton mappings*, Cybernetics, 19 (1983), pp. 160–165.

[N79] A. Nozaki, *Equivalence problem of non-deterministic finite automata*, J. Comput. System Sci., 18 (1979), pp. 8–17.

[Sch76] M.P. Schützenberger, *Sur les relations rationnelles entre monoïdes libres*, Theoret. Comput. Sci., 3 (1976), pp. 243–259.

[S90] H. Seidl, *Equivalence of finite-valued bottom-up finite state tree transducers is decidable*, Proc. CAAP 1990, in Lecture Notes in Comput. Sci., 431, Springer-Verlag, Berlin, Heidelberg, 1990, pp. 269–284.

[T88] P. Turakainen, *On some transducer equivalence problems for families of languages*, Intern. J. Comput. Math., 23 (1988), pp. 99–124.

[T89] ———, personal communication.

[W87] A. Weber, *Über die Mehrdeutigkeit und Wertigkeit von endlichen Automaten und Transducern*, dissertation, Goethe-Universität, Frankfurt am Main, Germany, 1987.

[W88] ———, *A decomposition theorem for finite-valued transducers and an application to the equivalence problem*, Proc. MFCS 1988, in Lecture Notes in Comput. Sci., 324, Springer-Verlag, Berlin, Heidelberg, 1988, pp. 552–562.

[W90] ———, *On the valuedness of finite transducers*, Acta Inform., 27 (1990), pp. 749–780.

[W92a] A. WEBER, *On the lengths of values in a finite transducer*, Acta Inform., 29 (1992), to appear (see also, Proc. MFCS 1989, in Lecture Notes in Comput. Sci., 379, Springer-Verlag, Berlin, Heidelberg, 1989, pp. 523–533).

[W92b] ———, *Decomposing a k-valued transducer into k unambiguous ones*, Proc. Latin 1992, in Lecture Notes in Computer Science, 583, Springer-Verlag, Berlin, Heidelberg, 1992, pp. 503-515.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 203–210, February 1993

ONE MORE OCCURRENCE OF VARIABLES MAKES SATISFIABILITY JUMP FROM TRIVIAL TO NP-COMPLETE*

JAN KRATOCHVÍL[†], PETR SAVICKÝ[‡], AND ZSOLT TUZA[§]

Abstract. A Boolean formula in a conjunctive normal form is called a $(k, s) - formula$ if every clause contains exactly k variables and every variable occurs in at most s clauses. The $(k, s)-$SAT problem is the SATISFIABILITY problem restricted to $(k, s)-$formulas. It is proved that for every $k \geq 3$ there is an integer $f(k)$ such that $(k, s)-$SAT is trivial for $s \leq f(k)$ (because every $(k, s)-$formula is satisfiable) and is NP-complete for $s \geq f(k) + 1$. Moreover, $f(k)$ grows exponentially with k, namely, $\lfloor 2^k/ek \rfloor \leq f(k) \leq 2^{k-1} - 2^{k-4} - 1$ for $k \geq 4$.

Key words. satisfiabilty, Boolean formula, conjunctive normal form, NP-completeness

AMS(MOS) subject classifications. 68Q15, 68Q25, 05CXX

1. Introduction.

A Boolean formula in conjunctive normal form is called a $(k, s)-$*formula* if every clause contains exactly k distinct variables and every variable occurs in at most s clauses. A $(k, s)-$formula is called a $(k, = s)-$*formula* if every variable occurs in exactly s clauses. We denote by $(k, s)-$SAT (respectively, $(k, = s)-$SAT) the SATISFIABILITY problem restricted to $(k, s)-$ (respectively, $(k, = s)-$) formulas, namely,

(k, s)–SAT	$(k, = s)$–SAT
Instance: A (k, s)–formula Φ.	Instance: A $(k, = s)$–formula Φ.
Question: Is Φ satisfiable?	Question: Is Φ satisfiable?

Tovey [T] proved that every $(3, 3)-$formula is satisfiable, but $(3, 4)-$SAT is NP-complete. Thus considering $(3, s)-$SAT as a decision problem parametrized by s, the problem is either trivial (i.e., one can output the affirmative answer without computation) for $s \leq 3$ or NP-complete for $s \geq 4$. Such a sudden jump from triviality to NP-completeness occurs for every $k \geq 3$.

THEOREM 1.1. *For every $k \geq 3$, there is an integer $f(k)$ such that*
(i) *every $(k, s)-$formula with $s \leq f(k)$ is satisfiable*;
(ii) *$(k, s)-$SAT is NP-complete for every $s \geq f(k) + 1$.*

The proof of this result, presented in §2, extends the basic ideas of the argument given by Tovey [T] for $k = 3$. The main task of this article is to derive reasonable bounds on $f(k)$. Note that obviously $f(k) = \max\{s \mid \text{every } (k, s)-\text{formula is satisfiable}\}$, and we may extend the definition of $f(k)$ to $k = 1, 2$ in this way. We will also prove the following strengthening of part (ii) of Theorem 1.1 in §2.

THEOREM 1.2. *The $(k, = s)-$SAT problem is NP-complete for every $k \geq 3$ and $s \geq f(k) + 1$.*

As a nice application of Hall's marriage theorem, Tovey [T] proved that $f(k) \geq k$ for every k. He also conjectured that every $(k, 2^{k-1} - 1)-$formula is satisfiable. The following result disproves this conjecture. It also shows, however, that the exponential upper bound 2^{k-1} is essentially best possible, apart from a factor linear in k.

THEOREM 1.3. *For every $k \geq 4$, $\lfloor \frac{2^k}{ek} \rfloor \leq f(k) \leq 2^{k-1} - 2^{k-4} - 1$.*

*Received by the editors July 23, 1990; accepted for publication (in revised form) September 24, 1991.
[†]Department of Algebra, Charles University, Sokolovská 83, 186 00 Prague, Czechoslovakia.
[‡]Department of Logic, Faculty of Philosophy, Charles University, Nám. J. Palacha 1, 100 00 Praha 1, Czechoslovakia.
[§]Computer and Automation Institute, Hungarian Academy of Sciences, Kende u. 13-17 H-1111 Budapest, Hungary.

The lower bound given in this theorem is proved in §3. The upper bound is related to the existence of nonsatisfiable formulas. We show in §4 that every nonsatisfiable (k, ∞)–formula has at least 2^k clauses. We also show that every nonsatisfiable $(k, f(k)+1)$–formula contains more than $\log_2 f(k)$ (and hence more than $k - \log_2 k - \log_2 e$) variables that occur in exactly $f(k) + 1$ clauses. Finally, the upper bound on $f(k)$ is proved in §5.

In the subsequent sections, we use the following convention. Formulas Φ and Ψ are called SAT-*equivalent* if either both are satisfiable or both are nonsatisfiable. The number of clauses of a formula Φ that contain a variable x (positive or negative) is called the *degree* of x and it is denoted by $deg_\Phi x$. Formulas are called *disjoint* if their variable sets are disjoint. If a clause c contains a variable x, say, $c = (x^\epsilon \wedge x_1^{\epsilon_1} \wedge \ldots \wedge x_{k-1}^{\epsilon_{k-1}})$ (where $\epsilon_i \in \{1, -1\}$ and $x^1 = x$ and $x^{-1} = \neg x$), then $c - x$ is the clause $(x_1^{\epsilon_1} \wedge \ldots \wedge x_{k-1}^{\epsilon_{k-1}})$. For a real t, we denote by $\lceil t \rceil$ (respectively, $\lfloor t \rfloor$) the least integer not smaller than t (respectively, the greatest integer not exceeding t).

2. The jump. In this section, we prove Theorems 1.1 and 1.2. The following concept will play an important role.

DEFINITION 2.1. A (k, s)–formula Φ containing a variable x is called an x-implying (k, s)-formula if

(i) the variable x occurs in at most $s - 1$ clauses of Φ,
(ii) x receives the value TRUE in every satisfying truth assignment for Φ,
(iii) Φ is satisfiable.

LEMMA 2.2. *If $k \geq 3$ and s are such that an x–implying (k, s)–formula exists, then (k, s)–SAT is* NP-*complete.*

Proof. We show that k–SAT $\propto$ (k, s)–SAT, provided an x–implying (k, s)–formula exists. In our terminology, k–SAT is (k, ∞)–SAT (i.e., SATISFIABILITY restricted to formulas with exactly k variables per clause, but with an unbounded number of occurrences of variables) and it is known to be NP-complete for every fixed $k \geq 3$ [GJ].

Let Φ be an x–implying (k, s)–formula and let Ψ be an instance of k–SAT. We will show how to reduce the degrees of variables in Ψ using disjoint copies of Φ, i.e., we will construct a (k, s)–formula $\widetilde{\Psi}$ SAT-equivalent to Ψ.

Let y be a variable of Ψ that occurs in more than s clauses, say, in the clauses $c_1, c_2, \ldots, c_m,$ $m > s$ (if Ψ has no such variables, we are done). Let $\Phi_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq k-2$, be disjoint copies of the formula Φ, with the variable x of Φ being renamed as $x_{i,j}$ in $\Phi_{i,j}$ (thus each $\Phi_{i,j}$ is an $x_{i,j}$–implying (k, s)–formula, and these formulas and the formula Ψ have pairwise disjoint sets of variables).

A formula Ψ' is defined as a result of a local replacement in Ψ in the following way. The variable y is replaced by m new variables $y_1, y_2, \ldots, y_m$, where each y_i replaces y in the clause c_i (so that $y_i \in c_i$ in Ψ' if and only if $y \in c_i$ in Ψ and $\neg y_i \in c_i$ in Ψ' if and only if $\neg y \in c_i$ in Ψ). In addition, Ψ' contains the formulas $\Phi_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq k-2$ and the clauses $d_i = (y_i \vee \neg y_{i+1} \vee \neg x_{i,1} \vee \neg x_{i,2} \vee \ldots \vee \neg x_{i,k-2})$, $1 \leq i \leq m$ (the subscripts of y_i are read modulo m).

Obviously, Ψ' has fewer variables of degree greater than s. We claim that Ψ' is satisfiable if and only if Ψ is satisfiable. Suppose ϕ' : {variables of Ψ'} $\rightarrow$ {TRUE,FALSE} is a truth assignment that satisfies Ψ'. By the definition of Φ, $\phi'(x_{i,j})$ = TRUE for all $1 \leq i \leq m$ and $1 \leq j \leq k-2$. Hence the clauses $\{d_i | 1 \leq i \leq m\}$ are satisfied by the variables $\{y_i | 1 \leq i \leq m\}$. It follows that $\phi'(y_i)$ have the same value, say, $\phi'(y_i) = \phi'_y \in$ {TRUE,FALSE} for all $1 \leq i \leq m$. Then ϕ : { variables of Ψ} $\rightarrow${TRUE,FALSE} defined by

$$\phi(x) = \begin{cases} \phi'_y & \text{if} \quad x = y \\ \phi'(x) & \text{if} \quad x \neq y \end{cases}$$

is a truth assignment that satisfies Ψ. Conversely, if Ψ is satisfiable, a satisfying truth assignment for Ψ' is guaranteed by condition (iii) of Definition 2.1.

If Ψ contains more variables of degree greater than s, we repeat the construction above letting each of these variables play the role of y, until finally a (k,s)–formula $\widetilde{\Psi}$ SAT-equivalent to Ψ is obtained. Note that if Φ contains k_Φ variables and h_Φ clauses and Ψ contains k_Ψ variables of degree $\leq s$, n variables of degree greater than s (say, of degrees $m_1, m_2, \ldots, m_n$) and h_Ψ clauses, then $\widetilde{\Psi}$ contains $k_\Psi + \sum_{i=1}^n m_i(1 + k_\Phi(k-2))$ variables and $h_\Psi + \sum_{i=1}^n m_i(1 + h_\Phi(k-2))$ clauses, and it is thus constructible in polynomial time. □

LEMMA 2.3. *If $k \geq 3$ and s are such that a nonsatisfiable (k,s)–formula exists, then there exists an x–implying (k,s)–formula as well.*

Proof. Let Φ be a minimal nonsatisfiable (k,s)–formula (i.e., if C is the set of clauses of Φ, then for every $C' \subsetneq C$, $\Phi' = \bigwedge C'$ is satisfiable). Choose a variable x and a clause c which contains x (or $\neg x$). Define $\widetilde{C} = (C \setminus \{c\}) \cup \{\widetilde{c}\}$, with $\widetilde{c} = (c - x) \cup \{\widetilde{x}\}$, where $\widetilde{x}$ is a new extra variable not occurring in Φ ($\widetilde{x}$ is positive in $\widetilde{c}$). We claim that $\widetilde{\Phi}$ is an $\widetilde{x}$–implying (k,s)–formula.

Since $\widetilde{x}$ occurs in just one clause, $\widetilde{\Phi}$ obviously is a (k,s)–formula and condition (i) of Definition 2.1 holds in $\widetilde{\Phi}$ (note that $s \geq 2$ follows from the nonsatisfiability of Φ). If $\widetilde{\Phi}$ is satisfied by a truth assignment $\widetilde{\phi}$, then $\widetilde{\phi}(\widetilde{x}) = \text{TRUE}$ (otherwise $\widetilde{\phi}$ would satisfy Φ as well, contradicting the nonsatisfiability of Φ). On the other hand, it follows from the minimality of Φ that $\Phi' = \bigwedge(C \setminus \{c\})$ is satisfiable, and hence so is $\widetilde{\Phi}$. □

Proof of Theorem 1.1. Put $m = f(k)$ and the statement follows directly from Lemmas 2.2 and 2.3. □

Proof of Theorem 1.2. We will prove that (k,s)–SAT $\propto$ $(k,=s)$–SAT. Let a formula Ψ with a set C of clauses over a set X of variables be an instance of (k,s)–SAT. The purpose is to construct a $(k,=s)$–formula $\widetilde{\Psi}$ SAT-equivalent to Ψ. This can be done by adding new clauses to Ψ to complete the degrees of variables of Ψ, together with new variables that guarantee that the new clauses can always be satisfied. The explicit construction is as follows.

We begin with s disjoint copies $\Psi_1, \Psi_2, \ldots, \Psi_s$ of the formula Ψ. We use the following notation. Each formula Ψ_i has clause set $C_i = \{c_i | c \in C\}$ and variables $X_i = \{x_i | x \in X\}$, so that $x_i \in c_i$ if and only if $x \in c$ and $\neg x_i \in c_i$ if and only if $\neg x \in c$. For each $x \in X$, consider $s - deg_\Psi(x)$ formulas $\Phi_{x,1}, \Phi_{x,2}, \ldots, \Phi_{x,s-deg_\Psi(x)}$, where

$$\Phi_{x,i} = \bigwedge_{j=1}^{s} (x_j \vee y_{i,1} \vee y_{i,2} \vee \ldots \vee y_{i,k-1})$$

(the variables x_j are the same as those occurring in the formulas Ψ_j, while $y_{i,j}$ are new extra variables). Straightforwardly,

$$\widetilde{\Psi} = \bigwedge_{i=1}^{s} \Psi_i \wedge \bigwedge_{x \in X} \bigwedge_{i=1}^{s-deg_\Psi(x)} \Phi_{x,i}$$

is a $(k,=s)$–formula and it is satisfiable if and only if Ψ is satisfiable. □

3. The lower bound. In this section we prove that $f(k) \geq \lfloor \frac{2^k}{ek} \rfloor$ for all $k \geq 1$. Of course, this inequality is trivial for $k \leq 3$; and for $k \leq 7$, it follows from $f(k) \geq k$, which Tovey [T] proved. We will use the following variant of a lemma of Erdös and Lovász [EL] (frequently referred to as Lovász's Local Lemma).

LEMMA [S]. *Let $\xi_1, \xi_2, \ldots, \xi_n$ be events in a probability space* (Ω, Prob). *A graph Γ with vertex set $\xi_1, \xi_2, \ldots, \xi_n$ is called an event graph of $\xi_1, \xi_2, \ldots, \xi_n$ if its edges express dependencies between the events so that each event ξ_i is independent of any combination of events nonadjacent to ξ_i. Suppose there exist reals $\gamma_i, 0 < \gamma_i < 1, i = 1, 2, \ldots, n$ satisfying*

$$\text{Prob}(\xi_i) \leq \gamma_i \prod_{\xi_i \xi_j \in E(\Gamma)} (1 - \gamma_j)$$

for $i = 1, 2, \ldots, n$. Then

$$\text{Prob}(\neg\xi_1 \wedge \neg\xi_2 \wedge \ldots \wedge \neg\xi_n) > 0. \qquad \square$$

Set $g(k) = \lfloor \frac{2^k}{ek} \rfloor$ and consider any $(k, g(k))$–formula Φ with clauses $c_1, c_2, \ldots, c_m$ over variables $x_1, x_2, \ldots, x_n$. We prove that Φ is satisfiable.

Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ be a random truth assignment of the variables, i.e., $\mathbf{x}_i \in$ {TRUE,FALSE}, each $\mathbf{x}_i$ equals TRUE with probability $\frac{1}{2}$, and the random variables $\mathbf{x}_i$ are independent. Denote by ξ_j the event that c_j is not satisfied by $\mathbf{x}$. Then $\text{Prob}(\xi_j) = 2^{-k}$ for $j = 1, 2, \ldots, m$.

The graph $\Gamma = (\{\xi_i | 1 \leq i \leq m\}, \{\xi_i\xi_j | \xi_i$ and ξ_j share a variable$\})$ is an event graph for $\xi_i, 1 \leq i \leq m$. Since each variable occurs in at most $g(k)$ clauses, and each clause contains just k variables, every ξ_j has degree at most $k(g(k) - 1)$ in Γ. Put $\gamma_i = e\,\text{Prob}(\xi_i) = e2^{-k}$ for each i. By our assumptions,

$$\frac{\gamma_i}{\text{Prob}(\xi_i)} \prod_{\xi_i \xi_j \in E(\Gamma)} (1 - \gamma_j) \geq e(1 - \frac{e}{2^k})^{k(\lfloor \frac{2^k}{ek} \rfloor - 1)} > e(1 - \frac{e}{2^k})^{\frac{2^k}{e} - 1} > ee^{-1} = 1.$$

Thus

$$\text{Prob}(\neg\xi_1 \wedge \neg\xi_2 \wedge \ldots \wedge \neg\xi_n) > 0$$

follows by the local lemma. Consequently, $\Phi(\mathbf{x})$ =TRUE holds with a positive probability. This fact implies that Φ is satisfied by some suitably chosen truth assignment.

4. Nonsatisfiable formulas. In this section we prove lower bounds on two other parameters of nonsatisfiable formulas—the number of clauses and the number of variables with degree $f(k) + 1$. In the following proposition clauses are not necessarily of the same size.

PROPOSITION 4.1. *If $\Phi = \bigwedge_{i=1}^{n} c_i$, then at least $n - \sum_{i=1}^{n} 2^{-size(c_i)}$ clauses can be satisfied simultaneously (here $size(c_i)$ is the number of variables occurring in c_i).*

Proof. We use the random truth assignment as in §3 and we define random variables $\Xi_i, i = 1, 2, \ldots, n$ by

$$\Xi_i = \begin{cases} 1 & \text{if } c_i \text{ is satisfied by } \mathbf{x}, \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\text{E}\left(\sum_{i=1}^{n} \Xi_i\right) = \sum_{i=1}^{n} \text{Prob}(c_i \text{ is satisfied}) = n - \sum_{i=1}^{n} 2^{-size(c_i)},$$

which means that at least $n - \sum_{i=1}^{n} 2^{-size(c_i)}$, clauses are satisfied by a suitable truth assignment. □

COROLLARY 4.2. *If $\Phi = \bigwedge_{i=1}^{n} c_i$ and $\sum_{i=1}^{n} 2^{-size(c_i)} < 1$, then Φ is satisfiable.*

COROLLARY 4.3. *Every nonsatisfiable (k, ∞)–formula has at least 2^k clauses.*

We remark that Corollary 4.3 is the best possible, since for every k, the formula consisting of all the 2^k possible k–clauses over variables $x_1, x_2, \dots, x_k$ is nonsatisfiable. Proposition 4.1 was actually proved in [J] (though Johnson only states the result for the case of (k, ∞)–formulas). The proof presented there is algorithmic, and we decided to include our probabilistic argument, which we believe is more transparent.

PROPOSITION 4.4. *If Φ is a nonsatisfiable (k, ∞)–formula, then at least one of its clauses shares variables with at least $\lfloor \frac{2^k}{e} \rfloor$ other clauses.*

Proof. Follows from the local lemma and is similar to the proof in §3. □

By the definition of $f(k)$, we know that nonsatisfiable $(k, f(k) + 1)$–formulas exist. We show that such formulas are rich in variables occurring in $f(k) + 1$ clauses.

THEOREM 4.5. *Let Φ be a nonsatisfiable $(k, f(k) + 1)$–formula, $k \geq 4$. Then at least $\lfloor \log_2 f(k) \rfloor + 1$ of the variables of Φ occur in $f(k) + 1$ clauses.*

Proof. Let $x_1, x_2, \dots, x_t$ be the variables that occur each in $f(k) + 1$ clauses of a $(k, f(k) + 1)$–formula Φ. We prove that Φ is satisfiable, provided $t \leq \log_2 f(k)$. The idea is to construct a $(k, f(k))$–formula Ψ SAT-equivalent to Φ by taking 2^t disjoint (and slightly modified) copies of Φ. Then Ψ is satisfiable by the definition of $f(k)$ and the satisfiability of Φ follows.

Here is the explicit construction of Ψ. For each x_i, choose a clause, say c_i, of Φ which contains x_i (respectively, $\neg x_i$). Consider t new variables $y_1, y_2, \dots, y_t$ and 2^t disjoint copies Φ_u of Φ, indexed by Boolean vectors $u = (u_1, u_2, \dots, u_t), u_i \in \{0, 1\}$. For a variable x and a clause c of Φ, let their copies in Φ_u be denoted by x^u and c^u, respectively. For each $u \in \{0, 1\}^t$, let $\widetilde{\Phi}_u$ be the formula obtained from Φ_u by replacing the occurrence of x_i^u in c_i^u by y_i (respectively, $\neg y_i$) if $u_i = 0$ (respectively, $u_i = 1$). Finally, we put

$$\Psi = \bigwedge_{u \in \{0,1\}^t} \widetilde{\Phi}_u.$$

The degrees of the variables x_i^u in Ψ drop by one compared to their degrees in Φ_u, the degrees of other variables do not change, and the degrees of the variables y_i are equal to $2^t \leq f(k)$. Thus Ψ is a $(k, f(k))$–formula. Let ϕ : {variables of Ψ} $\to$ {TRUE,FALSE} be a truth assignment that satisfies Ψ. Consider $u_\phi = (u_1, u_2, \dots, u_t)$ with $u_i = 0$ (respectively, $u_i = 1$) if $\phi(y_i)$ =FALSE (respectively, $\phi(y_i)$ =TRUE). Then no clause $c_i^{u_\phi}$ in $\widetilde{\Phi}_{u_\phi}$ is satisfied by y_i , and hence all clauses of $\widetilde{\Phi}_{u_\phi}$ are satisfied by the variables x^{u_ϕ}. Therefore, Φ is satisfiable. □

The bound given by Theorem 4.5 is probably not the best possible. On the other hand, the result cannot be extended straightforwardly to (k, s)–formulas with $s > f(k)+1$, because of the following.

PROPOSITION 4.6. *For every k, there exists a nonsatisfiable (k, ∞)–formula that contains just one variable with degree greater than $f(k)$.*

Before proving this observation, we need one auxiliary result that is also interesting on its own.

LEMMA 4.7. *The function $f(k)$ is strictly increasing.*

Proof. We prove that for every k, there exists a nonsatisfiable $(k, f(k+1))$–formula. Let Φ be a nonsatisfiable $(k + 1, f(k + 1) + 1)$–formula. Obviously, every formula obtained from Φ by deleting some variables from some clauses (while retaining the occurrences in other clauses) is nonsatisfiable. It only remains to show that one may obtain a $(k, f(k+1))$–formula in this way.

Suppose the formula Φ has a set C of clauses over a set X of variables, and let $x_1, x_2, \ldots, x_n$ be the variables of Φ that occur in $f(k+1)+1$ clauses. For $i = 1, 2, \ldots, n$, we put $C_i = \{c | c \in C \text{ and } c \text{ contains } x_i \text{ or } \neg x_i\}$ and we consider the set system $\mathcal{C} = \{C_i | 1 \le i \le n\}$. For every $A \subset \{1, 2, \ldots, n\}$, we have

$$f((k+1)+1)|A| = |\{(x_i, c) | c \in C_i \text{ and } i \in A\}| \le (k+1)|\bigcup_{i \in A} C_i|,$$

and, hence,

$$|\bigcup_{i \in A} C_i| \ge \frac{f(k+1)+1}{k+1}|A| > |A|$$

(note that $f(k+1) \ge k+1$). Using Hall's theorem, it follows that there are *distinct* clauses $c_i, i = 1, 2, \ldots, n$ such that each c_i contains either x_i or $\neg x_i$.

Now we let Ψ be a formula obtained from Φ by deleting the occurrence of x_i from c_i for $i = 1, 2, \ldots, n$ and deleting arbitrary variables from the remaining clauses, one from each. Consequently, Ψ is a nonsatisfiable $(k, f(k+1))$–formula and $f(k) < f(k+1)$ follows. □

Proof of Proposition 4.6. Suppose $k \ge 2$ (for $k = 1$, $\Phi = (x) \wedge (\neg x)$ is a desirable formula). Let Φ be a nonsatisfiable $(k-1, f(k-1)+1)$–formula with a set C of clauses over a set X of variables, and consider two disjoint copies Φ_1, Φ_2 of Φ (Φ_i has clause set $C_i = \{c_i | c \in C\}$ and variable set $X_i = \{x_i | x \in X\}$). Let y be a new extra variable. Then

$$\Psi = \left(\bigwedge_{c \in C} (c_1 \vee y) \right) \vee \left(\bigwedge_{c \in C} (c_2 \vee \neg y) \right)$$

is a (k, ∞)–formula with just one variable (namely, y) occurring in more than $f(k)$ clauses (all other variables occur in at most $f(k-1)+1 \le f(k)$ clauses). Obviously, Ψ is nonsatisfiable, since y cannot satisfy simultaneously both Φ_1 and Φ_2. □

5. The upper bound. We first present a recursive inequality for $f(k)$.

LEMMA 5.1. *For every k, we have $f(k+1) \le 2f(k)+1$.*

Proof. Let Φ be a nonsatisfiable $(k, f(k)+1)$–formula with a set C of clauses over a set X of variables. For each $c \in C$, let x_c be a new extra variable. Consider

$$\Psi = \bigwedge_{c \in C} (c \vee x_c) \wedge (c \vee \neg x_c).$$

The degrees of the variables of Φ are doubled in Ψ, while the new variables $x_c, c \in C$ have degree 2. Thus, Ψ is a $(k+1, 2(f(k)+1))$–formula. To see that it is nonsatisfiable, recall that for each clause c of Φ, $(c \vee x_c) \wedge (c \vee \neg x_c)$ is SAT-equivalent to c.

Therefore, $f(k+1)+1 \le 2f(k)+2$ and the statement follows. □

PROPOSITION 5.2. *We have $f(4) \le 6$.*

Proof. Let a, b be variables. We first construct an auxiliary $(4, 7)$–formula $\Phi_{a,b}$ such that every truth assignment satisfying $\Phi_{a,b}$ satisfies $a \Rightarrow b$. Then we use disjoint copies of this formula to reduce the degrees of variables in a nonsatisfiable $(4, \infty)$–formula in analogy to the proof of Lemma 2.2.

We put

$$\begin{aligned}\Phi_{a,b} = {} & (u \vee x \vee y \vee z) \wedge (\neg u \vee \neg x \vee \neg y \vee \neg z) \wedge (u \vee \neg x \vee v \vee w)\\ & \wedge (x \vee \neg y \vee v \vee w) \wedge (y \vee \neg z \vee v \vee w) \wedge (z \vee \neg u \vee v \vee w)\\ & \wedge (\neg a \vee b \vee \neg v \vee w) \wedge (\neg a \vee b \vee \neg v \vee \neg w) \wedge (\neg a \vee b \vee v \vee \neg w).\end{aligned}$$

It is straightforward to check that $deg_{\Phi_{a,b}} u = deg_{\Phi_{a,b}} x = deg_{\Phi_{a,b}} y = deg_{\Phi_{a,b}} z = 4, deg_{\Phi_{a,b}} v = deg_{\Phi_{a,b}} w = 7$, and $deg_{\Phi_{a,b}} a = deg_{\Phi_{a,b}} b = 3$. One can also check easily that every truth assignment satisfying $\Phi_{a,b}$ satisfies $a \Rightarrow b$ as well, and on the other hand, every truth assignment to a, b that satisfies $a \Rightarrow b$ can be extended to a truth assignment satisfying $\Phi_{a,b}$.

In the forthcoming construction, we will use copies of this formula. Let the convention be such that $\Phi_{\alpha,\beta}$ is a copy of $\Phi_{a,b}$ with a (respectively, b) being renamed as α (respectively, β), the other variables u, x, y, z, v, w of $\Phi_{a,b}$ are renamed so that these new variables do not occur in other copies of $\Phi_{a,b}$.

Finally, let Ψ be a nonsatisfiable $(4, \infty)$–formula, and suppose a variable x occurs in $n \geq 8$ clauses, say, $c_1, c_2, \dots, c_n$. We replace x by n new variables $x_1, x_2, \dots, x_n$ (x_i replaces x in c_i), and we add formulas $\Phi_{x_i, x_{i+1}}, i = 1, 2, \dots, n$ $(x_{n+1} = x_1)$. The resulting formula Ψ' is again nonsatisfiable, since any satisfying truth assignment ϕ of Ψ' would satisfy $x_i \Rightarrow x_{i+1}, i = 1, 2, \dots, n$, and ϕ would induce a truth assignment satisfying Ψ in an obvious way. Note that each variable x_i occurs in 7 clauses of Ψ'. Applying this construction to all variables of degree greater than 7 in Ψ', we finally obtain a nonsatisfiable $(4, 7)$–formula $\widetilde{\Psi}$. □

The above construction can be straightforwardly generalized to $k > 4$. This leads, however, to the upper bound $f(k) \leq 2^{k-1} - 2$, which is worse than the one in Theorem 1.3.

Proof of the upper bound of Theorem 1.3. The inequality follows by an easy induction on k, using Proposition 5.2 and Lemma 5.1. □

6. Concluding remarks.

6.1. The behavior of $f(k)$. We have defined the function $f(k)$, showed that it is strictly increasing, and proved exponential bounds for it. Undoubtedly, it would be very interesting to find an exact formula for $f(k)$, but that might be a rather tough problem. We would be satisfied whether one can eliminate the factor $\frac{1}{k}$ in the lower bound in Theorem 1.3.

Let us define the function

$$g(k) = \log_2(f(k) + 1) - k.$$

Theorem 1.3 yields

$$-\log_2 e - \log_2 k < g(k) \leq \log_2 7 - 4$$

and Lemma 5.1 implies

$$\begin{aligned} g(k+1) &= \log_2(f(k+1) + 1) - (k+1) \leq \log_2(2(f(k) + 1)) - k - 1\\ &= 1 + \log_2(f(k) + 1) - k - 1 = g(k),\end{aligned}$$

and thus $g(k)$ is monotone. Hence, the limit $g = \lim_{k \to \infty} g(k)$ exists, and therefore, we propose the following open problem.

PROBLEM. *Determine whether $g > -\infty$. Find reasonable bounds for g in case of an affirmative answer, or determine the asymptotic behavior of $g(k)$ if $g = -\infty$.*

Note that if $g > -\infty$, then $f(k)$ behaves asymptotically as 2^{k+g}, whereas if $g = -\infty$, then for every arbitrarily large positive constant M, $f(k) < 2^{k-M}$ for every $k > k_M$.

6.2. Computation of $f(k)$. Another tough problem may be to determine the exact values of $f(k)$, even for small k. We only know $f(1) = 1, f(2) = 2$ and $f(3) = 3$, but already for $k = 4$, we know nothing better than $4 \leq f(k) \leq 6$. The problem is that no bound on the size of a minimum nonsatisfiable $(4, f(4)+1)$ (and in general $(k, f(k)+1)-$ for $k \geq 4$)–formula is known.

6.3. Decision versus search. We have called the $(k, s)-$SAT problem trivial when $s \leq f(k)$. One should keep in mind that this concerns decision problems. The companion search problem (i.e., given a (satisfiable) $(k, s)-$formula with $s \leq f(k)$, find a satisfying truth assignment) is much harder. For $s \leq k$, one can find a satisfying truth assignment via matching theory, but for $k < s \leq f(k)$, no polynomial algorithm is known. This is just another example of a problem where the existence of a solution is guaranteed by a nonconstructive argument, but it is not known how difficult it is to find one. (On the other hand, in Proposition 4.1 our probabilistic approach gives exactly the same result as the polynomial search algorithm derived in [J].)

Note added in proof. Recently we became aware of a paper of Dubois [D] who was apparently the first to disprove the conjecture of Tovey. By a rather technical construction, he proves that $f(4) \leq 5$ and $f(5) \leq 10$. It then follows from our Lemma 5.1 that $f(k) \leq 11 \cdot 2^{k-5} - 1 = 2^{k-1} - 2^{k-3} - 2^{k-5} - 1$, which is better than the upper bound in our Theorem 1.3. Dubois also raised a conjecture which, in our setting, is equivalent to $\lim_{k\to\infty} \frac{f(k+1)}{f(k)} = 2$. This is strongly supported by the lower bound in our Theorem 1.3, which directly yields $\lim\,\sup_{k\to\infty} \frac{f(k+1)}{f(k)} = 2$. More precisely, it follows that for every $\epsilon > 0$, the set $A_\epsilon = \{k | \frac{f(k+1)}{f(k)} < 2 - \epsilon\}$ has zero density, i.e., $\lim_{n\to\infty} \frac{|A_\epsilon \cap \{1,2,\ldots,n\}|}{n} = 0$ (and even $\frac{|A_\epsilon \cap \{1,2,\ldots,n\}|}{n} \leq O(\frac{\log_2 n}{n})$).

A recent paper of Beck [B] provides a significant breakthrough to the decision versus search metaquestion for applications of Lovász's Local Lemma. Beck gives a polynomial search algorithm which, however, does not apply to the full range of the Local Lemma. Namely, for the restricted SATISFIABILITY problem, his method enables to find a satisfying truth assignment to a given $(k, s)-$formula if $s \leq \frac{2^{\frac{k}{48}+1}}{k} + 1$. A parallelized version of the Beck's algorithm is presented in a paper of Alon [A].

REFERENCES

[A] N. ALON, *A parallel algorithmic version of the Local Lemma*, Random Str. Alg., 2(1991), pp. 367–378.

[B] J. BECK, *An algorithmic approach to the Lovász Local Lemma* I, Random Str. Alg., 2(1991), pp. 343–365.

[D] O. DUBOIS, *On the $r, s-$SAT satisfiability problem and a conjecture of Tovey*, Discr. Appl. Math., 26(1990), pp. 51–60.

[EL] P. ERDÖS AND L. LOVÁSZ, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, A. Hajnal, L. Lovász, and V. T. Sós, eds., Colloquia Mathematica Societatis János Bolyai 10, North-Holland, Amsterdam, 1974, pp. 609–627.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, Harry Freeman, San Francisco, 1979.

[J] D. S. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9(1974), pp. 256–278.

[S] J. SPENCER, *Asymptotic lower bounds for Ramsey functions*, Discrete Math., 20(1977), pp. 69–76.

[T] C. A. TOVEY, *A simplified satisfiability problem*, Discrete Appl. Math., 8(1984), pp. 85–89.

SIAM J. COMPUT.
Vol. 22, No. 1, pp. 211–219, February 1993

ROUNDS IN COMMUNICATION COMPLEXITY REVISITED*

NOAM NISAN† AND AVI WIGDERSON‡

Abstract. The k-round two-party communication complexity was studied in the deterministic model by [P. H. Papadimitriou and M. Sipser, *Proc. of the 14th STOC*, 1982, pp. 330–337] and [P. Duris, Z. Galil, and G. Schnitger, *Proc. of the 16th STOC*, 1984, pp. 81–91] and in the probabilistic model by [A. C. Yao, *Proc. of the 24th FOCS*, 1983, pp. 420–428] and [B. Halstenberg and R. Reischuk, *Proc. of the 20th STOC*, 1988, pp. 162–172]. This paper presents new lower bounds that give (1) randomization is more powerful than determinism in k-round protocols, and (2) an *explicit* function which exhibits an exponential gap between its k and $(k-1)$-round randomized complexity.

This paper also studies the three-party communication model, and exhibits an exponential gap in 3-round protocols that differ in the starting player.

Finally, this paper shows new connections of these questions to circuit complexity, that motivate further work in this direction.

Key words. communication complexity, information, rounds, monotone circuits

AMS(MOS) subject classifications. 68, 94, 94A05, 94A17, 94C99

1. Introduction.

1.1. The two-party model. Papadimitriou and Sipser [14] initiated the study of how Yao's model[1] [19] of communication complexity is affected by limiting the two players to only k rounds of messages. They considered the following natural problem g_k: each of the players A and B is given a list of n pointers (each of $\log n$ bits), each pointing to a pointer in the list of the other. Their task is to follow these pointers, starting at some fixed $v_0 \in A$, and find the kth pointer. This can easily be done in k rounds and complexity $O(k \log n)$: A starts and the players alternately send the value of the next pointer. It is not clear how to use less than $n \log n$ bits if only $(k-1)$ rounds are allowed or in fact with k-rounds but player B starts. Indeed, Papadimitriou and Sipser [14] conjectured that the complexity is exponentially higher (for fixed k), namely, that there is a strict hierarchy, and proved it for the case $k = 2$. The general case was resolved by Duris, Galil, and Schnitger [4], who gave an $\Omega(n/k^2)$ lower bound on the $(k-1)$ round complexity of g_k.

It is not difficult to see that allowing randomness g_k can be solved with high probability in $(k-1)$ rounds using only $O((n/k)\log n)$ communication bits. Another $\log n$ factor in the complexity can make this a Las Vegas (errorless) algorithm. This raises the question, *What is the relative power of randomness over determinism in k-round protocols*? Without limiting the number of rounds, Mehlhorn and Schmidt [12] showed a quadratic gap between Las Vegas and Determinism, and allowing error, the gap can be exponential.

We use simple information theoretic and probabilistic arguments to strengthen the lower bound of [4] in two ways. First we improve their $(k-1)$-round deterministic lower bound on g_k to $\Omega(n)$ (for $k < n/\log n$), thus showing that randomness can be cheaper by a factor of $k/\log^2 n$ for k-round protocols. This result also provides the largest gap

*Received by the editors February 21, 1991; accepted for publication November 25, 1991. This work was partially supported by the Wolfson Research Awards (both authors) and the American–Israeli Binational Science Foundation grant 89-00126 (first author), administered by the Israel Academy of Sciences and Humanities.

†Department of Computer Science, Hebrew University, Jerusalem, 91904 Israel.

‡Department of Computer Science, Princeton University, Princeton, New Jersey 08544-2087.

[1]In fact, they and [4] considered the stronger "arbitrary partition" model, but known simulation results of [4], [6], [9], [10] allow us to use Yao's standard "fixed partition" model without loss of generality.

known for $k > \log n$ in the deterministic model — the previous one was obtained in [4] via counting arguments. The fact that the simulation of McGeoch [10] is constructive gives the same gap in the arbitrary partition model for an explicit function, resolving an open question of Duris, Galil, and Schnitger [4].

Second, we prove that the probabilistic upper bound above is not very far from optimal — we give an $\Omega(n/k^2)$ lower bound, establishing an exponential gap in the probabilistic setting between k and $(k-1)$-round protocols for an explicitly given function. The existence of such functions (with somewhat larger gap) was proven by Halstenberg and Reischuk [6], via complicated counting arguments. The only previous exponential gap for an explicit function was shown for $k = 2$ by Yao [20]. We stress the simplicity of our proof technique, in contrast to that of Halstenberg and Reischuk [6]. We have recently learned that similar techniques were used by Smirnov [16] to prove an $\Omega(n/(k(\log n)^k))$ lower bound on g_k, which is much weaker than our bound, but establish an exponential gap as well.

Finally, we use the communication complexity characterization of circuit depth of Karchmer and Wigderson [8] to establish g_k as a complete problem for monotone depth-k Boolean circuits. (This result was independently discovered by Yannakakis [18].) Thus, a simple deterministic reduction enables us to derive the monotone constant-depth hierarchy of Klawe, Paul, Pippenger, and Yannakakis [7] from the constant-round hierarchy of Duris, Galil, and Schnitzer [4]. (The reverse direction was proven in [7].) We speculate that our new probabilistic lower bound may serve to extend the monotone circuit hierarchy result to depth above $\log n$, via probabilistic reductions (as was done by Raz and Wigderson [15]).

1.2. The multi-party model. Chandra, Furst, and Lipton [2] devised the multi-party communication complexity model. Here t players $P_1, P_2, \ldots, P_t$ are trying to compute a Boolean function $g(x_1, x_2, \ldots, x_t)$, where $x_i \in \{0,1\}^{n_i}$. (Until now all work in this model considered equal length inputs, i.e., $n_i = n$ for all i.) The twist is that every player P_i sees *all* values x_j for $j \neq i$. This model turns out to capture diverse computational models. Chandra, Furst, and Lipton [2] used it to prove that majority requires super-linear length constant width branching programs. Babai, Nisan, and Szegedy [1] gave $\Omega(n/2^t)$ lower bounds for explicit functions g, and used it for Turing machine, branching program, and formulae lower bounds, as well as efficient pseudorandom generators for small space. Recently, Håstad and Goldmann [5] used the results in [1] to prove lower bounds on constant-depth threshold circuits.

We consider only the 3-player model, and within it we allow three rounds of communciation: one per player. We exhibit a function u whose complexity is $\Omega(\sqrt{n})$ if P_3 is the first to speak, but $O(\log n)$ otherwise. The proof uses properties of universal hash functions developed in [13] and [11]. It is interesting that u acts on different size arguments; $u : \{0, 1\}^{2n} \times \{0,1\}^n \times \{0, 1\}^{\log n} \to \{0, 1\}$ so $n_1, n_2 = \Theta(n)$, but $n_3 = \log n$. The following connection to circuit complexity makes such functions important. We show that improving our lower bound to $\Omega(n)$ for some explicit function g of this form would give the following size-depth trade-off: the function $f : \{0, 1\}^{\Theta(n)} \to \{0, 1\}^n$ defined by $f(x_1, x_2)_{x_3} = g(x_1, x_2, x_3)$ cannot be computed by Boolean circuits of size $O(n)$ and depth $O(\log n)$ simultaneously. This result is obtained via Valiant's [17] method of depth reduction in circuits.

2. The two-party model.

The four subsections of this section give the definitions, results, technical lemmas, and some proofs, respectively, in the two-party communication complexity model.

2.1. Definitions. Let $g : X_A \times X_B \to \{0,1\}$ be a function. The players A, B receive, respectively, inputs $x_A \in X_A$, $x_B \in X_B$. A k-round protocol specifies for each input a sequence of k messages, $m_1, m_2, \ldots, m_k$ sent alternately between the players such that at the end both know $g(x_A, x_B)$. The cost of a k-round protocol is $\sum_{i=1}^{k} |m_i|$ (where $|m_i|$ is the binary length of m_i), maximized over all inputs (x_A, x_B). Denote by $C^{A,k}(g)$ (respectively, $C^{B,k}(g)$) the cost of the best protocol in which player A (respectively, B) sends the first message, and $C^k(g) = \min\{C^{A,k}(g), C^{B,k}(g)\}$.

Let $T : X_A \times X_B \to \{0, 1\}$ be the function computed by the two players following a protocol T. We introduce randomization by allowing T to be a random variable distributed over deterministic protocols. The cost is simply the expectation of the associated random variable. We say that randomized protocol makes ϵ-error if $\Pr[T(x_A, x_B) \neq g(x_A, x_B)] \leq \epsilon$ for every input $(x_A, x_B) \in X_A \times X_B$. Denote by $C_\epsilon^k(g)$ the cost of the best k-round ϵ-error protocol for g and similarly define $C_\epsilon^{A,k}$, $C_\epsilon^{B,k}$. The case $\epsilon = 0$ (e.g., $C_0^k(g)$) denotes Las Vegas (errorless) protocols.

Finally, if we leave T a deterministic protocol and choose the input uniformly at random, we can define the ϵ-error distributional complexity $D_\epsilon^k(g)$ to be the cost of the best k-round protocol for which $\Pr[T(x_A, x_B) \neq g(x_A, x_B)] \leq \epsilon$, under this distribution. The following lemmas are useful.

LEMMA 1 [20]. *For every* g, $\epsilon > 0$ $\; D_{2\epsilon}^k(g) \leq 2C_\epsilon^k(g)$.

LEMMA 2. *For all constants* $\frac{1}{3} \geq \epsilon > \epsilon' > 0 \quad C_{\epsilon'}^k(g) = O(C_\epsilon^k(g))$.

2.2. Results. Let V_A, V_B be two disjoint sets (of vertices) with $|V_A| = |V_B| = n$ and $V = V_A \cup V_B$. Let

$$F_A = \{f_A : V_A \to V_B\}, \qquad F_B = \{f_B : V_B \to V_A\}$$

and

$$f = (f_A, f_B) : V \to V$$

defined by

$$f(v) = \begin{cases} f_A(v) & v \in V_A \\ f_B(v) & v \in V_B \end{cases}.$$

For each $k \geq 0$ define $f^{(k)}(v)$ by $f^{(0)}(v) = v$, $f^{(k+1)}(v) = f(f^{(k)}(v))$.

Let $v_0 \in V_A$. The function we will be interested in computing is $g_k : F_A \times F_B \to V$ defined by $g_k(f_A, f_B) = f^{(k)}(v_0)$.

Remark. In the following theorems note that the number of input bits to each player is $n \log n$ and that they hold for every value of k. We also note that one can make g_k a Boolean function by taking the parity of the output vertex. All our upper and lower bounds apply to this Boolean function as well.

THEOREM 1 [14]. $C^{A,k}(g_k) = O(k \log n)$.

THEOREM 2. $C^{B,k}(g_k) = \Omega(n - k \log n)$.

THEOREM 3. $C_{1/3}^{B,k}(g_k) = O((n/k) \log n)$, $C_0^{B,k}(g_k) = O((n/k) \log^2 n)$.

THEOREM 4. $C_{1/3}^{B,k}(g_k) = \Omega(\frac{n}{k^2} - k \log n)$.

Remark. At the end of the proofs of Theorems 2 and 4 we explain why the "ugly" term $-k \log n$ in these lower bounds can be essentially ignored.

In the remainder we show the completeness of g_k for monotone depth k circuits. Let $g_k = g_{k,n}$ to stress that each player gets n vertices.

DEFINITION. For a Boolean function h define $L^d(h)$ to be the size of the minimal *monotone formula* of depth d and unbounded fanin that computes h. Define $LS^d(k, n)$ to be the maximum of $L^d(h)$ over all functions h that can be computed by monotone circuits of unbounded fanin depth k and total size n. Define $LF^d(k, n)$ to be the maximum of $L^d(h)$ over all functions h that can be computed by a formula of depth k and fanin n at each gate.

THEOREM 5.

$$\log LS^d(k,n) \le C^d(g_{k,n}) \le \log LF^d(k,n).$$

The left inequality was proven in [7], and allowed them to deduce a lower bound on g_k from their circuit lower bound. The right inequality was independently discovered by Yannakakis [18]. It allows us to recover the tight hierarchy theorem of [7] from the lower bound on g_k.

Let h_k be the complete function for depth k-circuits, i.e., an alternating and-or tree of depth k and fanin $n^{1/k}$ at each gate.

COROLLARY (see Klawe et al. [7]). Any monotone circuit of depth $k - 1$ for h_k requires size $2^{\Omega(n^{1/k}/k)}$.

2.3. Probability, measure, and information theory. Let Ω be a finite set (universe), $X \subseteq \Omega$. Denote by $\mu(X)$ the *density* of X in Ω, $\mu(X) = |X|/|\Omega|$. Let $P : X \to [0, 1]$ a probability distribution on X, and $x \in X$ a random variable distributed according to P. The probability of any event $Y \subseteq X$ is denoted $\Pr_P[Y]$, and the subscript P is usually omitted. For $y \in X$, we write $\Pr[\{y\}] = P_y$. Then the *entropy* $H(P) = H(x) = \sum_{y\in X} P_y \log P_y$. The *information* on X (relative to Ω), is $I(x) = \log|\Omega| - H(x)$. If P is the uniform distribution U on X, then $H(x) = \log|X|$, and $I(x) = -\log \mu(X)$.

The following lemmas will be useful to us.

LEMMA 3. *Let $x = (x_1, x_2, \dots, x_m)$ be a random variable (so $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_m$ and x_i distributed over Ω_i), then $I(x) \ge \sum_{i=1}^m I(x_i)$.*

The next lemma (from [15]) shows that if $I(x)$ is very small, one can get good bounds on the probability of any event under P in terms of its probability under the uniform distribution U.

LEMMA 4 [15]. *For $Y \subseteq X$, let $q = \Pr_U[Y]$. Assume*

$$\Delta = \sqrt{\frac{4I(x)}{q}} \le \frac{1}{10}.$$

Then

$$|\Pr_P[Y] - q| \le q\Delta.$$

LEMMA 5. *If*

$$X = \Omega = \{0,1\},\ I(x) \le \delta \le \frac{1}{4},$$

then

$$\left|P_0 - \frac{1}{2}\right|,\ \left|P_1 - \frac{1}{2}\right| \le 2\sqrt{\delta}.$$

2.4. Proofs.

Proof of Theorems 1 *and* 3. $C^{A,k}(g_k) \leq k \log n$ follows easily, since in round t the right player knows $f(v_{t-1}) = v_t$ and can send these $\log n$ bits to the second player.

The idea in beating the deterministic $\Omega(n)$ lower bound when the wrong player B starts is as follows: First B chooses a random subset $U \subseteq V_B$ with $|U| = 10n/k$, and sends to A $\{f_B(u) : u \in U\}$. Now it is A's turn and they start sending each other $v_1, v_2, \ldots$ as above, but lagging one round "behind schedule." However, with probability $\geq 2/3$, one of the v_i's will be in U, which allows them to save two rounds, and "finish on time." This gives $C_\epsilon^{B,k}(g_k) = O((k + n/k)\log n)$. This algorithm can be made Las Vegas with an extra factor of $O(\log n)$ in the complexity.

Proof of Theorems 2 *and* 4. Let $f = (f_A, f_B) \in F_A \times F_B$ be the input. Let T' be a deterministic k-round protocol for g_k in which B sends the first message. Note that at any round $t \geq 1$, if it is B's turn to speak, then $v_{t-1} = f^{(t-1)}(v_0) \in V_A$, and vice versa. It will be convenient to replace T' by a protocol T in which in any round $t \geq 1$, we replace the message m by the message (m, v_{t-1}). By induction on t, this is always possible for the player whose turn it is. In particular, it implies that $\geq \log n$ bits are sent per round. Thus, if T' uses C bits, T uses $\leq C + k \log n$ bits. We will assume T uses $\epsilon n/2$ bits, (ϵ will be chosen later) and obtain a contradiction.

Every node z of the protocol tree T can be labeled by the rectangle $F_A^z \times F_B^z$ of inputs arriving at z. By the structure of T, if z is at level $t \geq 1$ (the root is at level 0), then $v_0, v_1, \ldots, v_{t-1}$ are determined in $F_A^z \times F_B^z$.

We shall assume the input is chosen uniformly at random from $F_A \times F_B$, so in fact we shall bound from below the distributional complexity. Thus the probability of arriving at z is $\mu(F_A^z \times F_B^z)$, and given that the input arrived at z, it is uniformly distributed in $F_A^z \times F_B^z$. The main lemma below intuitively shows that if the input arrived at z and the rectangle at z has nice properties, then with high (enough) probability the input will proceed to a child w of z, which is equally nice. Nice means that both F_A^z, F_B^z are large enough and that the player *not* holding v_{t-1} has very little information on $v_t = f(v_{t-1})$.

Denote by c_z the total number of bits sent by the players before arriving at z. Assume without loss of generality that A speaks at z. Let f_A^z and f_B^z be random variables uniformly distributed over F_A^z and F_B^z, respectively. Recall that T uses $\leq \frac{\epsilon}{2} n$ bits, and let δ satisfy $\delta = \text{Max}\,\{4\sqrt{\epsilon}, 400\,\epsilon\}$. Define z to be *nice* if it satisfies:

1. $I(f_A^z) \leq 2c_z$,
2. $I(f_B^z) \leq 2c_z$,
3. $I(f_B^z(v_{t-1})) \leq \delta$.

MAIN LEMMA. If z is nice, and w a random child of z, then $\Pr[w \text{ not nice}] \leq 4\sqrt{\epsilon} + \frac{1}{n}$.

Proof. Assume A sends bits at z. Let a_w be the length of the message, which leads to the child w. Thus, $c_w = c_z + a_w$ for all children w of z. We will now give upper bounds separately on the probability of each of the three properties defining nice being false at a random child w.

Claim 1. $\Pr[I(f_B^w) > 2c_w] = 0$.

Proof. B sent nothing, so $\forall\, w$ $F_B^w = F_B^z$ and

$$I(f_B^w) = I(f_B^z) \leq 2c_z < 2c_w. \qquad \square$$

Claim 2. $\Pr[I(f_A^w) > 2c_w] \leq \frac{1}{n}$.

Proof. A child w is chosen with probability $\mu(F_A^w)/\mu(F_A^z)$. Also note that the for every w $a_w \geq \log n$ (by our assumption on the protocol) and they satisfy Kraft's inequality $\sum_w 2^{-a_w} \leq 1$ (where the sum is over all children of z). Thus we have

$$\Pr[I(f_A^w) > 2c_w] \leq \Pr[\mu(F_A^w) < 2^{-2c_w}]$$
$$\leq \Pr[\mu(F_A^w)\big/\mu(F_A^z) < 2^{-2a_w}] \leq \tfrac{1}{n}\sum_w 2^{-a_w} \leq \frac{1}{n}. \qquad \square$$

Claim 3. $\Pr[I(f_A^w(v_t)) > \delta] \leq 4\sqrt{\epsilon}$.

Proof. We may assume now that $I(f_A^w) \leq 2c_w \leq \epsilon n$. The random variable f_A^w is a vector of random variables $f_A^w(v)$ for all $v \in V_A$. Thus, by Lemma 3, $\sum_{v\in V} I(f_A^w(v)) \leq I(f_A^w) \leq \epsilon n$. So if v_t was chosen *uniformly* from V_A, $\Pr_U[I(f_A^w(v_t)) > \delta] \leq \frac{\epsilon}{\delta}$ by Markov's inequality. But $v_t = f_B^z(v_{t-1})$, so v_t is distributed with $I(v_t) = I(f_B^z(v_{t-1})) \leq \delta$ as we assumed z was nice. By Lemma 4 (and our choice of δ),

$$\Pr[I(f_A^w(v_t)) > \delta] \leq \frac{\epsilon}{\delta}\left(1 + \sqrt{\frac{4\delta}{\epsilon/\delta}}\right) \leq 4\sqrt{\epsilon}. \qquad \square$$

Now we can conclude the proofs of Theorems 2 and 4 from the main lemma. Consider any *nice* leaf ℓ of the protocol tree T, labeled by an answer (0 or 1). Say A spoke on the last round k. Then $I(v_k) = I(f_B^\ell(v_{k-1})) \leq \delta$. So by Lemma 5, even if the algorithm gives one bit (say, parity) of the answer, it is correct with probability $\leq \frac{1}{2} + 2\sqrt{\delta}$.

Conclusion of Theorem 2. Take $\epsilon = 10^{-4}$. The root of T is nice, so by the main lemma and induction we have a positive probability ($\geq 2^{-k}$) of reaching a nice leaf, contradicting the fact that the protocol never errs. This proves only $C^{B,k}(g_k) = \Omega(n - k\log n)$, since we augmented an arbitrary T' to a nice protocol T.

Getting rid of the $-k\log n$ term, achieving the lower bound $C^{B,k}(g_k) = \Omega(n)$ (which is stronger when $k \geq \frac{n}{\log n}$) requires a more delicate argument that we sketch below. The idea is to follow the same steps of the proof with the following changes.

(1) We stay with the original protocol T', as we cannot afford the players sending $\log n$ bits per round as in the nice protocol T.

(2) We still fix the vertex v_{t-1} by the player sending the message at round t, but avoid paying $\log n$ bits for this information by removing this vertex from our universe. Thus, the information I is measured relative to a smaller set of pointers at every round.

(3) We prove a weaker main lemma, which is clearly sufficient in the deterministic case, namely, that every nice node z has at least one nice child w. The details are left to the interested reader.

Conclusion of Theorem 4. Pick $\epsilon = 10^{-4}\cdot k^{-2}$. Thus the probability of not reaching a nice leaf is $\leq k\frac{1}{25k} = \frac{1}{25}$, and the probability that the protocol answers correctly is less than $\frac{1}{25} + (\frac{1}{2} + \frac{2}{5\sqrt{k}}) < 0.95$. Thus we get from Lemmas 1 and 2 that $D_{1/20}^{B,k}(g_k) = \Omega((n/k^2) - k\log n)$.

This bound is $\Omega(n/k^2)$ for all $k < (n/\log n)^{1/3}$. For larger k one can use the trivial lower bound k, which applies to every k-round protocol. Note that when $k \geq n^{1/3}$ this trivial bound is larger than n/k^2.

Proof of Theorem 5. As mentioned above, the left inequality was proven in [7], so we prove only the right inequality. The proof is based on the Karchmer and Wigderson characterization of circuit depth in terms of communication complexity, which can be stated as follows. For every monotone function h on n variables with minterms $\text{Min}(h)$ and maxterms $\text{Max}(h)$ define a communication search problem $R_h^m \subset \text{Min}(h) \times \text{Max}(h) \times [n]$ in which player A gets a minterm $S \in \text{Min}(h)$, player B gets a maxterm $T \in \text{Max}(h)$, and their task is to find an element in $S \cap T$. Then monotone formulae for h and protocols for R_h^m are in 1-1 correspondence via the simple syntactic identification of $\vee$ gates with player A's moves and $\wedge$ gates with player B's moves. In particular, depth corresponds to the number of rounds and logarithm of the size to the communication complexity.

In view of the above, all we need to give now is a reduction from computing $g_{k,n}$ to the computation of R_h^m for some function h, which has a depth k formula of fanin n at each gate. Once this is done, the players can solve R_h^m and hence, $g_{k,n}$ in d rounds and $\log LF^d(k,n)$ communication by simulating the guaranteed depth d circuit for h.

Let h be defined by a formula that is a complete n-ary tree of depth k, alternating levels of $\vee$ and $\wedge$ gates (say, with $\vee$ at the root), and distinct n^k variables at the leaves. The players agree on a fixed labeling of the nodes of this tree in which the root is labeled v_0, the children of every $\vee$ gates labeled by V_B, and children of every $\wedge$ gate labeled by V_A. Let f_A and f_B be the inputs to players A, B, respectively. Player A constructs sets S_i of nodes from the ith level inductively as follows. S_0 contains the root. If level i contains $\vee$ gates, then for every gate in S_i labeled v he adds to S_{i+1} the unique child of this gate labeled $f_A(v)$. If level i contains $\wedge$ gates, then for every gate in S_i he adds all its children to S_{i+1}. In a similar way (exchanging the roles of gates) player B constructs his sets T_i. It is easy to verify that S_k is a minterm of h, T_k is a maxterm of h, and that they intersect at a unique leaf whose label is $f^{(k)}(v_0)$. This completes the reduction, and hence the proof. $\square$

3. The three-party model. Let $g : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^{n_3} \to \{0,1\}$ be a function. Players P_1, P_2, P_3 are given (x_2, x_3), (x_1, x_3), (x_1, x_2), respectively, with $x_i \in \{0,1\}^{n_i}$ and compute g from this information by exchanging messages according to a predetermined protocol. We consider only 3-round protocols in which each player speaks once. Let $M^i(g)$ denote the communication complexity when player P_i speaks first (and then the other two in arbitrary order), and $M^s(g)$ the complexity when they all speak simultaneously (an oblivious protocol). Clearly, for all $i \in \{1,2,3\}$ $M^i(g) \leq M^s(g)$.

Let $u : \{0,1\}^{2n} \times \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}$ be the following function. Interpret the first string x_1 as a 2-universal hash function (see Carter and Wegman [3]) h, mapping $\{0,1\}^n$ to itself, the second string x_2 as an argument y to h, and the third x_3 as an index $j \in [n]$. Then, $u(h,y,j) = h(y)_j$. The next two theorems exhibit an exponential gap between 3-round protocols that differ in the order in which players speak.

THEOREM 6. $M^1(u) = M^2(u) = O(\log n)$.

THEOREM 7. $M^3(u) = \Omega(\sqrt{n})$.

Proof of Theorem 7. We first recall a fundamental lemma from [11] regarding the distribution of hash values given little information on the hash function and the argument.

LEMMA 6 [11]. *Let $H = \{h : I \to O\}$ be a collection of universal hash functions from domain I into range O. Let $A \subset I$, $B \subset O$, $C \subset H$ and $p = |B|/|O|$. Then*

$$|\Pr[h(x) \in B \mid x \in A, h \in C] - p| \leq \sqrt{p|H|/(|A||C|)}.$$

Restrict the value of j to be $j \in [\sqrt{n}]$. Thus, we consider $h : \{0,1\}^n \to \{0,1\}^{\sqrt{n}}$, which is still a universal hash function. Assume $M^3(u) \leq \sqrt{n}/5$. This means that there is a new protocol (independent of j) to compute $z = h(y)$ in which P_3 sends $\sqrt{n}/5$ bits, and then players P_1 and P_2 can compute each bit of z separately, using altogether $n/5$ bits.

Let (h, y) be chosen uniformly at random. Simple averaging shows that there are messages m_1, m_2, m_3 of P_1, P_2, P_3, respectively, which under this distribution, satisfy $\Pr[m_1] \geq 2^{-n/4}$, $\Pr[m_2] \geq 2^{-n/4}$, and $\Pr[m_3] \geq 2^{-\sqrt{n}/4}$. As m_1 corresponds to a subset C of all hash functions (inputs to P_1), and m_2 corresponds to a subset A of all inputs to P_2, we can use Lemma 6 with $|H|/|C| \leq 2^{n/4}$, $1/|A| \leq 2^{-3n/4}$, $B = \{z\}$ and $p = 2^{-\sqrt{n}}$ to obtain

$$\Pr[h(y) = z \mid m_1, m_2, m_3] \leq 2^{\sqrt{n}/4} \Pr[h(y)$$

$$= z \mid m_1, m_2] \leq 2^{\sqrt{n}/4} \cdot 2^{-\sqrt{n}/2} = 2^{-\sqrt{n}/4} < 1.$$

Let $f : \{0,1\}^m \to \{0,1\}^n$ be an arbitrary function, and for any $m' < m$ define

$$g_f : \{0,1\}^{m'} \times \{0,1\}^{m-m'} \times \{0,1\}^{\log n} \to \{0,1\} \text{by} g_f(x_1, x_2, x_3)$$

$$= f(x_1 \circ x_2)_{x_3},$$

where $\circ$ denotes concatenation. The next theorem gives the relationship of size-depth trade-offs in circuits to 3-round oblivious protocols.

THEOREM 8. *If f above can be computed by a circuit of fan-in 2, size $O(n)$, and depth $O(\log n)$, then $M^s(g_f) = O(n/\log\log n)$.*

Proof of Theorem 8. Let $f : \{0,1\}^m \to \{0,1\}^n$ be computed by a circuit C of size $O(n)$ and depth $O(\log n)$. By a result of Valiant [17], there are $s = O(n/\log\log n)$ wires in C, $e_1, e_2, \ldots, e_s$ with the following property. For every input $x \in \{0,1\}^m$, and every $j \in [n]$, $f(x)_j$ is determined by the values $e_1(x), \ldots, e_s(x)$ on these wires, together with the values of x_i, $i \in S_j$ with $|S_j| \leq \sqrt{n}$. To compute g_f, note that P_3 has access to $x = (x_1 \circ x_2)$ (which is the input to f) and, therefore, can compute the values on the wires. P_2 and P_3, now knowing $j = x_3$, exchange the necessary bits in S_j to complete the computation of $f(x)_j$.

Acknowledgments. We are grateful to the referees for careful reading and many valuable suggestions.

REFERENCES

[1] L. BABAI, N. NISAN, AND M. SZEGEDY, *Multiparty protocols and logspace-hard pseudorandom sequences*, Proc. of the 21st STOC, 1989, pp. 1-11.

[2] A. CHANDRA, M. FURST, AND R. LIPTON, *Multi-party protocols*, Proc. of the 15th STOC, 1983, pp. 94–99.

[3] L. CARTER AND M. WEGMAN, *Universal hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[4] P. DURIS, Z. GALIL, AND G. SCHNITGER, *Lower bounds of communication complexity*, Proc. of the 16th STOC, 1984, pp. 81–91.

[5] J. HÅSTAD AND M. GOLDMANN, *On the power of small depth threshold circuits*, Proc. of the 31st FOCS, 1990, pp. 610-618.

[6] B. HALSTENBERG AND R. REISCHUK, *On different modes of communication*, Proc. of the 20th STOC, 1988, pp. 162-172.

[7] M. KLAWE, W. J. PAUL, N. PIPPENGER, AND M. YANNAKAKIS, *On monotone formulae with restricted depth*, Proc. of the 16th STOC, 1984, pp. 480–487.

[8] M. KARCHMER AND A. WIGDERSON, *Monotone circuits for connectivity require super-logarithmic depth*, Proc. of the 20th STOC, 1988, pp. 539–550.

[9] T. LAM AND L. RUZZO, *Results on communication complexity classes*, Proc. of the 4th Structures in Complexity Theory Conference, 1989, pp. 148–157.

[10] L. A. McGeoch, *A strong separation between k and $k-1$ round communication complexity for a constructive language*, Tech. Report CMU-CS-86-157, Carnegie Mellon University, Pittsburgh, PA, 1986.

[11] Y. Mansour, N. Nisan, and P. Tiwary, *The computational complexity of universal hashing*, Proc. of the 22nd STOC, 1990, pp. 235–243.

[12] K. Mehlhorn and E. Schmidt, *Las Vegas is better than determinism in VLSI and distributed computing*, Proc. of the 14th STOC, 1982, pp. 330–337.

[13] N. Nisan, *Pseudorandom generators for space bounded computation*, Proc. of the 22nd STOC, 1990, pp. 204–212.

[14] P. H. Papadimitriou and M. Sipser, *Communication complexity*, Proc. of the 14th STOC, 1982, pp. 330–337.

[15] R. Raz and A. Wigderson, *Probabilistic communication complexity of Boolean relations*, Proc. of the 30th FOCS, 1989, pp. 562–567.

[16] D. V. Smirnov, *Shannon's information methods for lower bounds for probabilistic communication complexity*, manuscript 1989. (In Russian.)

[17] L. Valiant, *Graph theoretic arguments in low-level complexity*, Tech. Report CS 13-77, University of Edinburgh, Edinburgh, UK, 1977.

[18] M. Yannakakis, private communication.

[19] A. C.-C. Yao, *Some complexity questions related to distributive computing*, Proc. of the 11th STOC, 1979, pp. 209–213.

[20] ———, *Lower bounds by probabilistic arguments*, Proc. of the 24th FOCS, 1983, pp. 420–428.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 221–242, April 1993

RECURSIVE STAR-TREE PARALLEL DATA STRUCTURE*

OMER BERKMAN† AND UZI VISHKIN†‡

Abstract. This paper introduces a novel parallel data structure called the recursive star-tree (denoted "*-tree"). For its definition a generalization of the $*$ functional is used (where for a function $f * f(n) = \min\{i | f^{(i)}(n) \leq 1\}$ and $f^{(i)}$ is the ith iterate of f). Recursive *-trees are derived by using recursion in the spirit of the inverse Ackermann function.

The recursive *-tree data structure leads to a new design paradigm for parallel algorithms. This paradigm allows for extremely fast parallel computations, specifically, $O(\alpha(n))$ time (where $\alpha(n)$ is the inverse of the Ackermann function), using an optimal number of processors on the (weakest) concurrent-read, concurrent-write parallel random-access machine (CRCW PRAM).

These computations need only constant time, and use an optimal number of processors if the following nonstandard assumption about the model of parallel computation is added to the CRCW PRAM: an extremely small number of processors each can write simultaneously into different bits of the same word.

Applications include finding lowest common ancestors in trees by a new algorithm that is considerably simpler than the known algorithms for the problem, restricted domain merging, parentheses matching, and a new parallel reducibility.

Key words. parallel algorithms, parallel data structures, lowest common ancestors

C.R. subject classification. F. 2. 2

1. Introduction.

The model of parallel computation used in this paper is the concurrent-read, concurrent-write (CRCW) parallel random-access machine (PRAM). We assume that several processors may attempt to write at the same memory location only if they are seeking to write the same value (the so-called Common CRCW PRAM). We use the weakest Common CRCW PRAM model, in which only concurrent writes of the value one are allowed. Given two parallel algorithms for the same problem, one is *more efficient* than the other if (1) primarily, its time–processor product is smaller and (2) secondarily (but important), its parallel time is smaller. An *optimal* parallel algorithm is an algorithm whose time–processor product matches the sequential complexity of the problem (which in this paper is always linear). A *fully parallel* algorithm is a parallel algorithm that runs in constant time and uses an optimal number of processors. An *almost fully parallel* algorithm is a parallel algorithm that runs in time $\alpha(n)$ (the inverse of the Ackermann function) while using an optimal number of processors.

The notion of a fully parallel algorithm represents an ultimate theoretical goal for designers of parallel algorithms. Research on lower bounds for parallel computation (see references later) indicates that for nearly any interesting problem this goal is not achievable. These same results also preclude almost fully parallel algorithms for the same problems. Therefore, any result that approaches this goal is somewhat surprising.

*Received by the editors April 16, 1990; accepted for publication (in revised form) January 8, 1992. Based on "Recursive *-Tree Parallel Data-Structure" by O. Berkman and U. Vishkin which appeared in the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, October 30–November 1, 1989, pp. 196–202. ©1989 IEEE.

†Department of Computing, King's College London, The Strand, London WC2R 2LS, England. Part of this work was carried out while the author was at the University of Maryland Institute for Advanced Computer Studies (UMIACS), College Park, Maryland 20742; and the Department of Computer Science, Tel Aviv University, Tel Aviv, Israel 69978. The work of this author was partially supported by National Science Foundation grant CCR-8906949.

‡University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, Maryland 20742; and Department of Electrical Engineering, University of Maryland, College Park, Maryland 20742; and the Department of Computer Science, Tel Aviv University, Tel Aviv, Israel 69978. The work of this author was partially supported by National Science Foundation grants CCR-8906949 and CCR-9111348.

The class of doubly logarithmic optimal parallel algorithms and the challenge of designing such algorithms is discussed in [BBG$^+$89]. The class of almost fully parallel algorithms represents an even stricter demand.

There is a remarkably small number of problems for which there exist optimal parallel algorithms that run in $o(\log\log n)$ time. These problems include (a) OR and AND of n bits; (b) finding the minimum among n elements, where the input consists of integers in the domain $[1, \cdots, n^c]$ for a constant c (see [FRW84]); (c) $\log^{(k)} n$ coloring of a cycle, where $\log^{(k)}$ is the kth iterate of the log function and k is constant [CV86b]; (d) some probabilistic computational geometry problems [Sto88]; (e) matching a pattern string in a text string, following a preprocessing stage in which a table based on the pattern is built [Vis91].

Not only is the number of such upper bounds small, there is evidence that for almost any interesting problem an $o(\log\log n)$-time optimal upper bound is impossible. We mention time lower bounds for a few very simple problems. Clearly, these lower bounds apply to more involved problems. For brevity, only lower bounds for optimal speed-up algorithms are stated: (a) parity of n bits, for which the time lower bound is $\Omega(\log n/\log\log n)$, following from the lower bound of [Has86] for circuits in conjunction with the general simulation result of [SV84] or from [BH87]; (b) finding the minimum among n elements, for which the lower bound is $\Omega(\log\log n)$ on a comparison model [Val75]; (c) merging two sorted arrays of numbers, for which the lower bound is similar to (b) [BH85].

The main contribution of this paper is a parallel data structure called a *recursive *-tree*. This data structure also provides a new paradigm for parallel algorithms. There are two known examples for which tree-based data structures provide a "skeleton" for parallel algorithms: (1) balanced binary trees, where the depth of such a tree with n leaves is $\log n$, and (2) "doubly logarithmic" balanced trees, where the depth of such a tree with n leaves is $\log\log n$ (each node of a doubly logarithmic tree, whose rooted subtree has x leaves, has $\sqrt{x}$ children). Balanced binary trees are used in the prefix-sums algorithm of [LF80] (perhaps the most heavily used routine in parallel computation) and in many other logarithmic-time algorithms. [BBG$^+$89] shows how to apply doubly logarithmic trees for guiding the flow of the computation in several doubly logarithmic algorithms (including some previously known algorithms). Similarly, the recursive *-tree data structure provides a new pattern for almost fully parallel algorithms.

To be able to list results obtained by application of *-trees, we define the following family of extremely slow-growing functions. Our definition is direct. A subsequent comment explains how this definition leads to an alternative definition of the inverse Ackermann function. For a more standard definition see [Tar75]. We note that such a direct definition is implicit in several "inverse-Ackermann-related" serial algorithms (e.g., [HS86]).

The inverse-Ackermann function. Consider a real function f. Let $f^{(i)}$ denote the ith iterate of f. (Formally, we denote $f^{(1)}(n) = f(n)$ and $f^{(i)}(n) = f(f^{(i-1)}(n))$ for $i > 1$.) Next, we define the $*$ (pronounced "star") functional that maps the function f into another function $*f$: $*f(n) = \min\{i \mid f^{(i)}(n) \leq 1\}$ (we consider only functions for which this minimum is well defined). (The function $\log^*$ will thus be denoted $*\log$.)

We define inductively a series I_k of slow-growing functions from the set of integers that are larger than 2 into the set of positive integers: (i) $I_1(n) = \lceil n/2 \rceil$ and (ii) $I_k = *I_{k-1}$ for $k \geq 2$. The first three in this series are familiar functions: $I_1(n) = \lceil n/2 \rceil$, $I_2(n) = \lceil \log n \rceil$, and $I_3(n) = *\log n$. The inverse Ackermann function is $\alpha(n) = \min\{i \mid I_i(n) \leq i\}$.

Comment. Ackermann's function is defined following [HS86] as follows: $A_1(n) = 2n$ and $A_k(n) = A_{k-1}^{(n)}(1)$ for $k \geq 2$.

Note that I_k is actually the inverse of the kth recursion level of A, the Ackermann function, namely, $I_k(n) = \min\{i \mid A_k(i) \geq n\}$ or $I_k(A_k(n)) = n$. The definition of $\alpha(n)$ is equivalent to the more often used (but perhaps less intuitive) definition: $\min\{i \mid A_i(i) \geq n\}$.

*Applications of recursive *-trees*.

1. *The lowest-common-ancestor* (LCA) *problem*. Suppose a rooted tree T is given for preprocessing. The preprocessing should enable a single processor to process quickly queries of the following form: Given two vertices u and v, find their lowest common ancestor in T.

Results. (i) Preprocessing is in $I_m(n)$ time and uses an optimal number of processors. (The space complexity here and throughout the paper is linear unless otherwise specified.) Queries will be processed in $O(m)$ time, that is, $O(1)$ time for constant m. A more specific result is (ii) almost fully parallel preprocessing and $O(\alpha(n))$ time for processing a query. These results assume that the Euler tour of the tree and the level of each vertex in the tree are given. Without this assumption the time for preprocessing is $O(\log n)$ with an optimal number of processors, and each query can be processed in constant time. For a serial implementation the preprocessing time is linear and a query can be processed in constant time.

Our algorithm for the LCA problem is new and is based on an approach that is completely different from the serial algorithm of Harel and Tarjan [HT84] and the simplified and parallelizable algorithm of Schieber and Vishkin [SV88]. Its serial version is considerably simpler than these two algorithms. Specifically, consider the Euler tour of the tree and replace each vertex in the tour by its level. This gives a sequence of integers. Unlike previous approaches, the new LCA algorithm is based only on an analysis of this sequence of integers. In particular, answering an LCA query converts to finding the minimum over some range $[i, i+1, \cdots, j]$ of the sequence of levels. (We define the range-minima problem formally and discuss previous results for it in §3.) This provides another interesting example of the quest for parallel algorithms also enriching the field of serial algorithms. Algorithms for quite a few problems use an LCA algorithm as a subroutine. We mention some: (1) strong orientation [Vis85]; (2) computing an open ear decomposition and st-numbering of a biconnected graph [MSV86] (also, [FRT89] and [RR89] use as a subroutine an algorithm for st-numbering and thus also the LCA algorithm); (3) approximate string matching on strings [LV89] and on trees [SZ89] and retrieving information on strings from their suffix trees [AIL$^+$88].

2. *The all-nearest-zero-bit problem*. Let $A = (a_1, a_2, \cdots, a_n)$ be an array of bits. For each bit a_i find the nearest-zero bit both to its left and right.

Result. The algorithm is almost fully parallel.

A similar problem is considered in [CFL83], where the motivation is circuits.

3. *The parentheses-matching problem*. Suppose a legal sequence of parentheses is given. For each parenthesis find its mate.

Result. Assuming the level of nesting of each parenthesis is given, we have an almost fully parallel algorithm. Without this assumption $T = O(\log n / \log\log n)$ if an optimal number of processors is used.

Parentheses matching in parallel is considered in [AMW89], [BSV88], [BV85], and [DS83]. We are putting the algorithm for parentheses matching into a later paper [BV91] to keep the present paper to a reasonable length.

4. *Restricted-domain merging*. Let $A = (a_1, \cdots, a_n)$ and $B = (b_1, \cdots, b_n)$ be two

nondecreasing lists whose elements are integers drawn from the domain $[1, \cdots, n]$. The problem is to merge them into a sorted list.

Result. The algorithm is almost fully parallel.

Merging in parallel is considered in [Van89], [BH85], [Kru83], [SV81], and [Val75]. We presented the merging algorithm in another paper [BV90], where it is implemented on the less powerful concurrent-read, exclusive-write (CREW PRAM) model with the same bounds, and the restricted-domain limitation is somewhat relaxed by using unrelated techniques.

5. *Almost fully parallel reducibility*. Let A and B be two problems. Suppose that any input of size n for problem A can be mapped into an input of size $O(n)$ for problem B. Such a mapping from A to B is an *almost fully parallel reducibility* if it can be realized by an almost fully parallel algorithm.

Given a convex polygon, the *all-nearest-neighbors* (*ANN*) problem is to find for each vertex of the polygon its nearest (Euclidean) neighbor. Using almost fully parallel reducibilities, we prove the following lower bound for the ANN problem: Any CRCW PRAM algorithm for the ANN problem that uses $O(n \log^c n)$ (for any constant c) processors needs $\Omega(\log \log n)$ time. We note that this lower bound is proved in [SV90] by using a considerably more involved technique.

Fully parallel results. For our fully parallel results we introduce the CRCW-bit PRAM model of computation. In addition to the above definition of the CRCW PRAM, we assume that a few processors can each write simultaneously into different bits of the same word. Specifically, in our algorithms this number of processors is very small and never exceeds $O(I_d(n))$, where d is a constant. Therefore, the assumption strikes us as quite reasonable from the architectural point of view. We believe that the cost for implementing a step of a PRAM on a feasible machine is likely to absorb implementation of this assumption at no extra cost. Nevertheless, we do not advocate adopting the CRCW-bit PRAM as a theoretical substitute for the CRCW PRAM.

Specific fully parallel results.

1. *The lowest-common-ancestor problem*. The preprocessing algorithm is fully parallel if we assume that the Euler tour of the tree and the level of each vertex in the tree are given. A query can be processed in constant time.

2. *The all-nearest-zero-bit problem*. The algorithm is fully parallel.

3. *The parentheses-matching problem*. If we assume that the level of nesting of each parenthesis is given, the algorithm is fully parallel.

4. *Restricted-domain merging*. The algorithm is fully parallel.

Results 3 and 4 are not explicitly given in [BV91] and [BV90]. However, they can be derived from the series of fast algorithms in these papers in a fashion similar to the way the fully parallel algorithm of §4.4 in the current paper is derived from the series of algorithms of Lemma 4.2.1 below.

We elaborate on where our fully parallel algorithms use the CRCW-bit new assumption. The algorithms work by mapping the input of size n into n bits. Then, given any constant d, we derive a value $x = O(I_d(n))$. The algorithms proceed by forming n/x groups of x bits each. Informally, our problem is then to pack all x bits of the same group into a single word and solve the original problem with respect to an input of size x in constant time. This packing is exactly where our almost fully parallel CRCW PRAM algorithms fail to become fully parallel and the CRCW-bit assumption is used. We believe that it is of theoretical interest to determine ways of avoiding such packing and thereby get fully parallel algorithms on a CRCW PRAM without the CRCW-bit assumption and suggest this as open problem.

A repeating motif in the present paper is putting restrictions on the domain of problems. This is the case for the all-nearest-zero-bit problem and the restricted-domain merging problem. Perhaps our more interesting applications concern problems whose input domain is *not explicitly restricted*. This is the case for the LCA problem and the parentheses-matching problem. However, as part of the design of our algorithms for these respective problems, we identified a few subproblems whose input domains are restricted: The level of vertices in the Euler tour is the restricted input for the LCA subproblem, and the level of nesting of each parenthesis is the input for the parentheses-matching subproblem. The Euler tour and prefix sums (by which the level of nesting is computed) are basic techniques for trees and arrays, respectively. A lower bound on their running times is $\Omega(\log n/\log\log n)$. One of the aims of this paper is to determine the added difficulty of each of the two problems, LCA and parentheses matching, beyond the use of these basic (and simple) techniques.

The rest of this paper is organized as follows. Section 2 describes the recursive *-tree data structure, and §3 recalls a few basic problems and algorithms. In §4 the parallel algorithms for LCA and the all-nearest-zero bit are presented. A sequential version of the LCA algorithm, which is considerably simpler than the parallel version, is also outlined. The almost fully parallel reducibility is presented in §5, and §6 discusses how to efficiently compute the functions used in this paper.

2. The recursive *-tree data structure. Let n be a positive integer. We define inductively a series of $\alpha(n) - 1$ trees. For each m, $2 \leq m \leq \alpha(n)$, a balanced tree with n leaves, denoted $BT(m)$, is defined. For a given m, $BT(m)$ is a recursive tree in the sense that each of its nodes holds a tree of the form $BT(m-1)$.

The base of the inductive definition (*see Fig.* 1). We start with the definition of the *-tree $BT(2)$. $BT(2)$ is simply a complete binary tree with n leaves.

Number of children per node	Number of leaves per node	Level of the tree
$I_1^{(0)}(n)/I_1^{(1)}(n) = 2$	$I_1^{(0)}(n) = n$	1
$I_1^{(1)}(n)/I_1^{(2)}(n) = 2$	$I_1^{(1)}(n) = \frac{n}{2}$	2
$I_1^{(2)}(n)/I_1^{(3)}(n) = 2$	$I_1^{(2)}(n) = \frac{n}{4}$	3
0	1	$*I_1(n) + 1$ $= I_2(n) + 1$ $= \log n + 1$

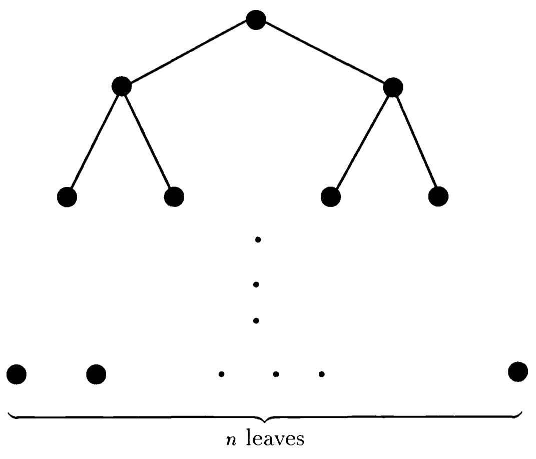


FIG. 1. *A BT (2) tree.*

The inductive step (*see Fig.* 2). For $m, 3 \leq m \leq \alpha(n)$, we define $BT(m)$ as follows. $BT(m)$ has n leaves. The number of levels in $BT(m)$ is $*I_{m-1}(n) + 1$ ($= I_m(n) + 1$). The root is at level 1, and the leaves are at level $*I_{m-1}(n) + 1$. Consider a node v at level $1 \leq l \leq *I_{m-1}(n)$ of the tree. Node v has $I^{(l-1)}_{m-1}(n)/I^{(l)}_{m-1}(n)$ children (we define $I^{(0)}_{m-1}(n)$ to be n). The total number of leaves in the subtree rooted at node v is $I^{(l-1)}_{m-1}(n)$. We refer to the part of the $BT(m)$ tree described so far as the *top recursion level* of $BT(m)$ (denoted $TRL-BT(m)$ for brevity). In addition, node v recursively contains a $BT(m-1)$ tree. The number of leaves in this tree is exactly the number of children of node v in $BT(m)$.

Number of children per node	Number of leaves per node	Level of the tree
$I^{(0)}_{m-1}(n)/I^{(1)}_{m-1}(n)$	$I^{(0)}_{m-1}(n) = n$	1
$I^{(1)}_{m-1}(n)/I^{(2)}_{m-1}(n)$	$I^{(1)}_{m-1}(n)$	2
$I^{(2)}_{m-1}(n)/I^{(3)}_{m-1}(n)$	$I^{(2)}_{m-1}(n)$	3
0	1	$*I_{m-1}(n) + 1$ $= I_m(n) + 1$

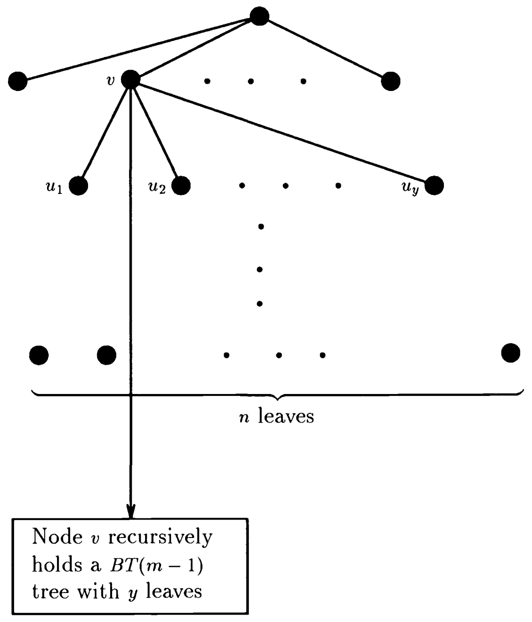


FIG. 2. *$BT(m)$—mth recursive $*$—tree.*

There is one key idea that enables our algorithms to be as fast as they are. When the mth tree $BT(m)$ is used to guide the computation, *we invest $O(1)$ time on the top recursion level for $BT(m)$*! Since $BT(m)$ has m levels of recursion, this leads to a total of $O(m)$ time.

Similar computational structures have appeared in a few contexts. See [AS87] and [Yao82] for generalized range-minima computations, [HS86] for Davenport–Schinzel sequences, and [CFL83] for circuits.

3. Basics. We need the following problems and algorithms.

The Euler-tour technique. Consider a tree $T = (V, E)$ rooted at some vertex r. The Euler-tour technique enables us to compute several problems on trees in logarithmic time and optimal speedup (see also [TV85] and [Vis85]). The technique is summarized below.

Step 1. For each edge $(v \to u)$ in T we add its antiparallel edge $(u \to v)$. Let H denote the new graph. Since the in-degree and out-degree of each vertex in H are the same, H has an Euler path that starts and ends in the root r of T. Step 2 computes this path into a vector of pointers D of size $2|E|$, where for each edge e of H, $D(e)$ gives the successor edge of e in the Euler path.

Step 2. For each vertex v of H we do the following. Let the outgoing edges of v be $(v \to u_0), \cdots, (v \to u_{d-1})$. Then $D(u_i \to v) := (v \to u_{(i+1) \bmod d})$ for $i = 0, \cdots, d-1$. Vector D now represents an Euler circuit of H. The "correction" $D(u_{d-1} \to r) :=$ end-of-list (where the out-degree of r is d) gives an Euler path that starts and ends in r.

Step 3. In this step we apply list ranking to the Euler path. This results in ranking the edges so that the tour can be stored in an array. Similarly, we can find for each vertex in the tree its distance from the root. This distance is called the *level* of vertex v. These and other applications of list ranking appear in [TV85]. This list ranking can be performed in logarithmic time with an optimal number of processors by the method of [AM88], [CV86a], or [CV89].

Comments.

1. In §4 we assume that the Euler tour is given in an already ranked form. There we systematically replace each edge (u, v) in the Euler tour by the vertex v. We then add the root of the tree to the beginning of this new array. Suppose a vertex v has l children. Then v appears $l + 1$ times in our array.

2. We note that while advancing from a vertex to its successor in the Euler tour, the level may either increase by one or decrease by one.

Finding the minimum for restricted-domain inputs. Given an array $A = (a_1, a_2, \cdots, a_n)$ where each a_i is an integer between 1 and n, find the minimum value in A.

Fich, Ragde, and Wigderson [FRW84] gave the following parallel algorithm for the restricted-domain minimum-finding problem. It runs in $O(1)$ time with n processors. We use an auxiliary vector B of size n, which is all zero initially. Processor $i, 1 \leq i \leq n$, writes one into location $B(a_i)$. Now the problem is to find the leftmost one in B. Partition B into $\sqrt{n}$ equal-size subarrays. For each such subarray find in $O(1)$ time, using $\sqrt{n}$ processors, whether the subarray contains a one. Apply the $O(1)$ time algorithm of Shiloach and Vishkin [SV81] for finding the leftmost subarray of size $\sqrt{n}$ containing a one, using n processors. Finally, reapply this latter algorithm for finding the index of the leftmost one in this subarray.

Remark 3.1. This algorithm can be readily generalized to yield $O(1)$ time for inputs between 1 and p^c, where $c > 1$ is a constant, as long as $p \geq n$ processors are used.

Range-minima problem. Given an array $A = (a_1, a_2, \cdots, a_n)$ of n real numbers, preprocess the array so that for any interval $[a_i, a_{i+1}, \cdots, a_j]$ the minimum over the interval can be retrieved in constant time by using a single processor.

We show how to preprocess A in constant (to be precise, $O(1/\epsilon)$) time with $n^{1+\epsilon}$ processors and $O(n^{1+\epsilon})$ space for any constant ϵ. The preprocessing algorithm uses the following naive parallel algorithm for the range-minima problem. Allocate n^2 processors to each interval $[a_i, a_{i+1}, \cdots, a_j]$, and find the minimum over the interval in constant time, as in [SV81]. This naive algorithm runs in constant time and uses n^4 processors and n^2 space.

The preprocessing algorithm. Suppose some constant ϵ is given. The output of the preprocessing algorithm can be described by means of a balanced tree T with n leaves, where each internal node has $n^{\epsilon/3}$ children. The root of the tree is at level one, and the leaves are at level $3/\epsilon+1$. Let v be an internal node, and, let $u_1, \cdots, u_{n^{\epsilon/3}}$ be its children. Each internal node v has the following data:

1. $M(v)$, the minimum over its leaves (i.e., the leaves of its subtree);
2. For each $1 \leq i < j \leq n^{\epsilon/3}$ the minimum over the range $\{M(u_i), M(u_{i+1}), \cdots, M(u_j)\}$.

This requires $O(n^{2\epsilon/3})$ space per internal node and a total of $O(n^{1+\epsilon/3})$ space.

The preprocessing algorithm advances through the levels of the tree in $3/\epsilon$ steps, starting from level $3/\epsilon$. At each level each node computes its data by using the naive range-minima algorithm. Each internal node uses $n^{4\epsilon/3}$ processors and $O(n^{2\epsilon/3})$ space, and the whole algorithm uses $n^{1+\epsilon}$ processors, $O(n^{1+\epsilon/3})$ space, and $O(3/\epsilon)$ time.

Retrieval. Suppose we wish to find the minimum, MIN(i,j), over an interval $[a_i, a_{i+1}, \cdots, a_j]$. Let LCA$(a_i, a_j)$ be the lowest common ancestor of leaf a_i and leaf a_j in the tree T. There are two possibilities.

(i) LCA(a_i, a_j) is at level $3/\epsilon$. The data at LCA(a_i, a_j) give MIN(i,j).

(ii) LCA(a_i, a_j) is at level $< 3/\epsilon$. Let x be the number of edges in the path from a_i (or a_j) to LCA(a_i, a_j) (i.e., LCA(a_i, a_j) is at level $3/\epsilon + 1 - x$). Using the tree T, we can represent interval $[i,j]$ as a disjoint union of $2x - 1$ intervals whose minima were previously computed. Formally, let $r(i)$ denote the rightmost leaf of the parent of a_i in T, and, let $l(j)$ denote the leftmost leaf of the parent of a_j in T. MIN$[i,j]$ is the minimum among three numbers.

1. MIN$[i, r(i)]$. The data at the parent of a_i give this information.
2. MIN$[l(j), j]$. The data at the parent of a_j give this information.
3. MIN$[r(i)+1, l(j)-1]$. Advance to level $3/\epsilon$ of the tree to get this information recursively.

Complexity. Retrieval of MIN(i,j) takes constant time: The first and second numbers can be looked up in $O(1)$ time. Retrieval of the third number takes $O(3/\epsilon - 1)$ time.

Below we review previous results for the range-minima problem. This review is appropriate since the main technical effort of §4 is a parallel algorithm for an instance of the range-minima problem. [GBT84] gives a linear time algorithm for preprocessing that allows constant-time query retrieval. We discuss these results further in Remark 4.1. An $O(\log\log n)$-time optimal parallel preprocessing algorithm for the range-minima problem is given in [BSV88]. The following generalization of the range-minima problem is considered in [Yao82] and [AS87]. The input array A contains semigroup elements, and a query over the range $[i,j]$ requests the product $a_i \cdot a_{i+1} \cdot \cdots \cdot a_j$, where "$\cdot$" is the semigroup product operator. Both papers give a sequential algorithm for the problem that runs in $O(n\alpha(n))$ time and allows $O(\alpha(n))$ time for query retrieval. In addition, they show that $\Omega(\alpha(n))$ is needed if only linear time is allowed for preprocessing. [AS87] also gives a tradeoff between the sequential time needed for preprocessing and the time for answering a query. The algorithm we give for our instance of the range-minima problem (given in §4) benefits from this approach. However, a serial implementation of our preprocessing algorithm needs only linear time.

4. LCA algorithm.

The input to this problem is a rooted tree $T = (V, E)$. Let $n = 2|V| - 1$. We assume that we are given a sequence of n vertices $A = [a_1, \cdots, a_n]$, which is the Euler tour of the input tree, and that we know for each vertex v its level, LEVEL(v), in the tree.

Recall the range-minima problem defined in §3. Below we give a simple reduction from the LCA problem to a restricted-domain range-minima problem, which is an instance of the range-minima problem in which *the difference between each two successive* *numbers for the range-minima problem is exactly one*. The reduction takes $O(1)$ time and uses n processors. An algorithm for the restricted-domain range-minima problem is given later, implying an algorithm for the LCA problem.

4.1. Reducing the LCA problem to a restricted-domain range-minima problem. Let v be a vertex in T. Denote by $l(v)$ the index of the leftmost appearance of v in A and by $r(v)$ the index of its rightmost appearance. For each vertex v in T it is easy to find $l(v)$ and $r(v)$ in $O(1)$ time with n processors by using the following (trivial) observations: $l(v)$ is where $a_{l(v)} = v$ and $\text{LEVEL}(a_{l(v)-1}) = \text{LEVEL}(v) - 1$; $r(v)$ is where $a_{r(v)} = v$ and $\text{LEVEL}(a_{r(v)+1}) = \text{LEVEL}(v) - 1$. The claims and corollaries below provide guidelines for the reduction.

CLAIM 1. *Vertex u is an ancestor of vertex v if and only if $l(u) < l(v) < r(u)$.*

COROLLARY 1. *Given two vertices u and v, a single processor can find in constant time whether u is an ancestor of v.*

COROLLARY 2. *Vertices u and v are unrelated (namely, neither is u an ancestor of v nor is v an ancestor of u) if and only if either $r(u) < l(v)$ or $r(v) < l(u)$.*

CLAIM 2. *Let u and v be two unrelated vertices. (By Corollary 2 we may assume without loss of generality that $r(u) < l(v)$.) Then the* LCA *of u and v is the vertex whose level is minimal over the interval $[r(u), l(v)]$ in A.*

The reduction. Let $\text{LEVEL}(A) = [\text{LEVEL}(a_1), \text{LEVEL}(a_2), \cdots, \text{LEVEL}(a_n)]$. Claim 2 shows that after we perform the range-minima preprocessing algorithm with respect to $\text{LEVEL}(A)$, a query of the form $\text{LCA}(u, v)$ becomes a range-minimum query. Observe that the difference between the level of each pair of successive vertices in the Euler tour (and thus each pair of successive entries in $\text{LEVEL}(A)$) is exactly one and therefore the reduction is to the restricted-domain range-minima problem as required.

Remark 4.1. In [GBT84] it is observed that the problem of preprocessing an array so that each range-minimum query can be answered in constant time (this is the range-minima problem defined in §3) is equivalent to the LCA problem. It gives a linear-time algorithm for the former problem by using an algorithm for the latter. This does not look very helpful: We know to solve the range-minima problem on the basis of the LCA problem, and, conversely, we know to solve the LCA problem on the basis of the range-minima problem. Nevertheless, using the restricted-domain properties of our range-minima problem, we show that this cyclic relationship between the two problems can be broken and thereby lead to a new algorithm.

4.2. The restricted-domain range-minima algorithm. Below we define a restricted-domain range-minima problem that is slightly more general than the problem for LEVEL (A). The more general definition enables recursion in the algorithm below. The rest of this section shows how to solve this problem.

The restricted-domain range-minima problem. We are given an integer k and an array $A = (a_i, a_2, \cdots, a_n)$ of integers, such that $|a_i - a_{i+1}| \leq k$. In words, the difference between each $a_i, 1 \leq i < n$, and its successor a_{i+1} is at most k. The parameter k need not be a constant. Preprocess the array $A = (a_1, a_2, \cdots, a_n)$ so that any query $\text{MIN}[i, j]$, $1 \leq i < j \leq n$, requesting the minimum element over the interval $[a_i, \cdots, a_j]$, can be processed quickly with a single processor.

Comment 1. We make the simplifying assumption that $\sqrt{k}$ and all other quantities needed in the paper are always integers. For example, $\log n$ and $n/\log n$ are assumed to be integers. This implies that the root of $BT(3)$ with n leaves has $n/\log n$ children, each with *exactly* $\log n$ leaves in its rooted subtree. To demonstrate that handling the general case is only a matter of technicality let us consider as an example the *-tree

$BT(3)$. When $BT(3)$ is used in the algorithm below with, say, r leaves, we need to store at each internal node an array that contains all leaves of the node. The space is thus $O(r)$ per level and $O(r \log^* r)$ overall. We sketch how to treat general r: The number of children of the root of $BT(3)$ will be $\lceil r/\log r \rceil$, each with $\lceil \log r \rceil$ leaves. This implies that additional $O(r/\log r)$ space is needed for all nodes at level 2 of the tree. At level 3 the additional space needed is $O(\lceil r/\log r \rceil \cdot \lceil \log \log r \rceil)$. Overall, the additional space required is linear. To facilitate memory access we will allocate to each level of $BT(3)$ an additional $O(r)$ space, which is more than is actually needed. This will double the space and number of processors needed by our algorithms but will not change the complexity in terms of "big Oh."

Comment 2. In case the minimum in the interval is not unique, find the minimum element in $[a_i, \cdots, a_j]$ whose index is smallest ("the leftmost minimum element"). Throughout this section, whenever we refer to a minimum over an interval we always mean the leftmost minimum element. Finding the leftmost minimum element (and not just the minimum value) will serve us later.

We start by constructing recursively a series of $\alpha(n) - 1$ parallel preprocessing algorithms for our range-minima problem.

LEMMA 4.2.1. *The algorithm for $2 \leq m \leq \alpha(n)$ runs in cm time, for some constant c, using $nI_m(n) + \sqrt{k}n$ processors and $o(nI_m(n))$ space. The preprocessing algorithm results in a table. With this table any range-minimum query can be processed in cm time with one processor. In addition, the preprocessing algorithm finds explicitly all prefix minima and suffix minima. Therefore, there is no need to do any processing for prefix-minima or suffix-minima queries.*

Our optimal algorithms, whose efficiencies are given in Theorem 4.3.1, are derived from this series of algorithms. We describe the series of preprocessing algorithms. We give first the base of the recursive construction and later the recursive step.

The base of the recursive construction (*the algorithm for $m = 2$*). To provide intuition for the description of the preprocessing algorithm for $m = 2$ we present first its output and how the output can be used for processing a range-minimum query.

Output of the preprocessing algorithm for $m = 2$:

(1) For each consecutive subarray $a_{j \log^3 n+1}, \cdots, a_{(j+1)\log^3 n}$, $0 \leq j \leq n/\log^3 n - 1$, we keep a table. The table enables constant-time retrieval of any range-minimum query within the subarray.

(2) We have array $B = b_1, \cdots, b_{n/\log^3 n}$, consisting of the minimum in each subarray.

(3) We have a complete binary tree $BT(2)$, whose leaves are $b_1, \cdots, b_{n/\log^3 n}$. Each internal node v of the tree holds an array P_v with an entry for each leaf of v. Consider prefixes that span between $l(v)$, the leftmost leaf of v, and a leaf of v. Array P_v has the minima over all these prefixes. Node v also holds a similar array S_v. For each suffix that spans between a leaf v and $r(v)$, the rightmost leaf of v, array S_v has its minimum.

(4) We have two arrays of size n each; one contains all prefix minima and the other contains all suffix minima with respect to A.

Remark. The reason behind the (seemingly arbitrary) grouping of array A into subarrays of $\log^3 n$ elements each is as follows: The main part of the preprocessing algorithm for $m = 2$ (which is given below) deals with constructing output element (3). To maintain processor count within our desired bounds, the input to this part (which is array B) must be of size $O(n/\log^3 n)$.

LEMMA 4.2.2. *Let m be 2. Then $I_m(n) = \log n$. The preprocessing algorithm runs in 2c time for some constant c using $n \log n + \sqrt{k}n$ processors and $o(n \log n)$ space. Retrieval of a query* MIN$[i, j]$ *takes 2c time with one processor.*

How does one retrieve a query MIN$[i,j]$ in constant time? There are two possibilities.

(i) a_i and a_j belong to the same subarray (of size $\log^3 n$). MIN(i,j) is computed in $O(1)$ time by using the table that belongs to the subarray.

(ii) a_i and a_j belong to different subarrays. Let right(i) denote the rightmost element in the subarray of a_i and let left(j) denote the leftmost element in the subarray of a_j. MIN$[i,j]$ is the minimum among three numbers:

1. MIN$[i$,right$(i)]$, the minimum over the suffix of a_i in its subarray;
2. MIN[left$(j),j]$, the minimum over the prefix of a_j in its subarray;
3. MIN[right$(i)+1$, left$(j)-1$].

The retrieval of the first and second numbers is similar to possibility (i) above. Denote $i_1 = \lceil i/\log^3 n\rceil + 1$ and denote $j_1 = \lceil j/\log^3 n\rceil - 1$. Retrieval of the third number is equivalent to finding the minimum over interval $[b_{i_1}, \cdots, b_{j_1}]$ in B, which is denoted MIN$_B[i_1, j_1]$.

Let x be the lowest common ancestor of b_{i_1} and b_{j_1}, let x_1 be the child of x that is an ancestor of b_{i_1}, and let x_2 be the child of x that is an ancestor of b_{j_1}. MIN$_B[i_1, j_1]$ is the minimum of two numbers:

1. MIN$_B[i_1, r(x_1)]$, the minimum over the suffix of b_{i_1} in x_1 (we obtain this from S_{x_1});
2. MIN$_B[l(x_2), j_1]$, the minimum over the prefix of b_{j_1} in x_2 (we obtain this from P_{x_2}).

How to find x, x_1, and x_2 in constant time remains to be shown. It is observed in [HT84] that the lowest common ancestor of two leaves in a complete binary tree can be found in constant time with one processor. (The idea is to number the leaves from 0 to $n-1$. Given two leaves i_1 and j_1, it suffices to find the most significant bit in which the binary representation of i_1 and j_1 are different in order to get their lowest common ancestor.) Thus x (and thereby also x_1 and x_2) can be found in constant time. Constant-time retrieval of the MIN(i,j) query follows.

The preprocessing algorithm for $m = 2$.

Step 1. Partition A into subarrays of $\log^3 n$ elements each. Allocate $\log^4 n$ processors to each subarray, and apply the preprocessing algorithm for range minima given in §3 (for $\epsilon = \frac{1}{3}$). This uses $\log^4 n$ processors and $O(\log^3 n \log^{1/3} n)$ space per subarray and $n \log n$ processors and $o(n \log n)$ space overall.

Step 2. Take the minimum in each subarray to build array B of size $n/\log^3 n$. The difference between two successive elements in B is at most $k \log^3 n$.

Step 3. Build $BT(2)$, a complete binary tree whose leaves are the elements of B. For each internal node v of $BT(2)$ we keep an array. The array consists of the values of all leaves in the subtree rooted at v. So the space needed is $O(n/\log^3 n)$ per level and $O(n/\log^2 n)$ for all levels of the tree. We allocate to each leaf at each level $\sqrt{k}\log^2 n$ processors, and the total number of processors used is thus $\sqrt{k}n$.

Step 4. For each internal node find the minimum element over its array. If the minimum element is not unique, the leftmost one is found. We apply the constant- time algorithm mentioned in Remark 3.1. Consider an internal node of size r. After subtracting the first element of the array from each of its elements, we get an array whose elements range between $-kr\log^3 n$ and $kr \log^3 n$. The size of the range, which is $2kr\log^3 n + 1$, does not exceed the square of the number of processors, which is $r\sqrt{k}\log^2 n$, and the algorithm of Remark 3.1 can be applied.

Step 5. For each internal node v we compute P_v (S_v is computed similarly). That is, for each leaf b_i of v we need to find MIN$_B[l(v), i]$ (the minimum over the prefix of b_i

in v). For this, the minimum among the following list of (at most) $\log n + 1$ numbers is computed. Denote the level of v in the binary tree by level (v). Each level l, level $(v) < l \leq \log n + 1$, of the tree contributes (at most) one number. Let u denote the ancestor at level $l-1$ of b_i. Let u_1 and u_2 denote the left and right children of u, respectively. If b_i belongs to (the subtree rooted at) u_2, then level l contributes the minimum over u_1. If b_i belongs to u_1, then level l does not contribute anything (actually, level l contributes a large default value so that the minimum computation is not affected). Finally, b_i is also included in the list. This minimum computation can be done in constant time with $\log^2 n$ processors by the algorithm of [SV81]. Note that all prefix minima and all suffix minima of B are computed (in the root) in this step. We note that since $BT(2)$ is a complete binary tree, node u (the ancestor at level $l-1$ of b_i) can be easily found in constant time with one processor ([HT84]).

Step 6. For each a_i we find its prefix minimum and its suffix minimum with respect to A in constant time by using one processor. Let b_j be the minimum representing the subarray of size $\log^3 n$ containing a_i. The minimum over the prefix of a_i with respect to A is the minimum between the prefix of b_{j-1} with respect to B and the minimum over the prefix of a_i with respect to its subarray.

This completes the description of the recursion base: Items (1), (2), (3), and (4) of the output were computed (respectively) in steps 1, 2, 5, and 6 above. The complexity of the recursion base is specified by $O(1)$ time by using $n \log n + \sqrt{k}n$ processors and $o(n \log n)$ space. Lemma 4.2.2 follows.

The recursion step. The algorithm is presented in a way similar to that of the recursion base.

Output of the mth preprocessing algorithm.

(1) For each consecutive subarray $a_{jI^3_m(n)+1}, \cdots, a_{(j+1)I^3_m(n)}$, $0 \leq j \leq n/I^3_m(n) - 1$, we keep a table. The table enables constant-time retrieval of any range-minimum query within the subarray. (The notation $I^3_m(n)$ means $(I_m(n))^3$, where $I_m(n)$ is defined earlier.)

(2) We have array $B = b_1, \cdots, b_{n/I^3_m(n)}$, consisting of the minimum in each subarray.

(3) We have $TRL\text{–}BT(m)$, the top recursion level of (the recursive *-tree) $BT(m)$, whose leaves are $b_1, \cdots, b_{n/I^3_m(n)}$. Each internal node v of $TRL-BT(m)$ holds an array P_v and array S_v with an entry for each leaf of v. These arrays hold (as in the binary tree for $m = 2$) prefix minima and suffix minima with respect to the leaves of v.

(4) Let $u_1, \cdots, u_y$ be the children of v, an internal node of $TRL-BT(m)$. Denote by MIN(u_i) the minimum over the leaves of node u_i, $1 \leq i \leq y$. Each such node v has recursively the output of the $(m-1)$th preprocessing algorithm with respect to the input MIN$(u_1), \cdots$, MIN(u_y).

(5) We have two arrays of size n each; one contains all prefix minima and the other contains all suffix minima with respect to A.

How can we retrieve a query MIN$[i, j]$ in cm time? We distinguish two possibilities.

(i) a_i and a_j belong to the same subarray (of size $I^3_m(n)$). MIN(i, j) is computed in $O(1)$ time by using the table that belongs to the subarray.

(ii) a_i and a_j belong to different subarrays. Again, let right(i) denote the rightmost element in the subarray of a_i, and let left(j) denote the leftmost element in the subarray of a_j. MIN$[i, j]$ is the minimum among three numbers:

1. MIN$[i$,right$(i)]$;
2. MIN$[$left$(j), j]$;
3. MIN$[$right$(i) + 1$,left$(j) - 1]$.

The retrieval of the first and second numbers is similar to possibility (i) above. Denote $i_1 = \lceil i/I_m^3(n) \rceil + 1$, and denote $j_1 = \lceil j/I_m^3(n) \rceil - 1$. Retrieval of the third number is equivalent to finding the minimum over interval $[b_{i_1}, \cdots, b_{j_1}]$ in B, which is denoted $\text{MIN}_B[i_1, j_1]$.

Let x be the lowest common ancestor of b_{i_1} and b_{j_1} in $TRL-BT(m)$, let $x_{\beta(i_1)}$ be the child of x that is an ancestor of b_{i_1}, and let $x_{\beta(j_1)}$ be the child of x that is an ancestor of b_{j_1}. $\text{MIN}_B[i_1, j_1]$ is (recursively) the minimum among three numbers:

1. $\text{MIN}_B[i_1, r(x_{\beta(i_1)})]$, the minimum over the suffix of b_{i_1} in $x_{\beta(i_1)}$ (we obtain this from $S_{x_{\beta(i_1)}}$);

2. $\text{MIN}_B[l(x_{\beta(j_1)}), j_1]$, the minimum over the prefix of b_{j_1} in $x_{\beta(j_1)}$ (we obtain this from $P_{x_{\beta(j_1)}}$);

3. $\text{MIN}_B[r(x_{\beta(i_1)}) + 1, l(x_{\beta(j_1)}) - 1]$ (this will be recursively derived from the data at node x).

The first two numbers are precomputed in $TRL–BT(m)$. The recursive definition of the third number implies that $\text{MIN}_B[i_1, j_1]$ is actually the minimum among $4(m-1)-2$ precomputed numbers and can thus be done in $\overline{c_1}m$ time for some constant $\overline{c_1}$. Below we show how to find the nodes x, $x_{\beta(i_1)}$, and $x_{\beta(j_1)}$ in constant time with one processor. This (together with the recursive definition of $BT(m)$) implies that over the retrieval procedure finding these nodes takes $\overline{c_2}m$ time for some constant $\overline{c_2}$ when one processor is used . We choose $c > \overline{c_1} + \overline{c_2}$, and the retrieval time is then cm, as required.

We first note that for each leaf of $TRL-BT(m)$, finding the child of the root that is its ancestor needs constant time if one processor is used: Recall that the number of leaves of $TRL-BT(m)$ is $n/I_m^3(n)$, its root has $(n/I_m^3(n))/I_{m-1}(n/I_m^3(n))$ children, and each of these children has $I_{m-1}(n/I_m^3(n))$ leaves. Thus the ancestor of a leaf b_i, $1 \le i \le n/I_m^3(n)$, of $TRL–BT(m)$ is the $\lceil i/I_{m-1}(n/I_m^3(n)) \rceil$th child of the root. Given two leaves of $TRL–BT(m)$, consider their ancestors among the children of the root. If these ancestors are different, we are done. Suppose these ancestors are the same.

Observe that for $TRL–BT(m)$ the same subtree structure is replicated at each child of the root. As part of the preprocessing algorithm we build one table with respect to this generic subtree structure. For each pair of leaves u and v of the generic subtree structure, we compute three items into a table: (1) their lowest common ancestor w; (2) the child f of w that is an ancestor of u; (3) the child g of w that is an ancestor of v. Since each child of the root has $I_{m-1}(n/I_m^3(n)) \le I_{m-1}(n)$ leaves, the size of the table is only $O(I_{m-1}^2(n))$.

How the table is computed remains to be shown. Consider an internal node w of the generic subtree structure, and suppose that its rooted subtree has r leaves. At node w each pair of leaves u, v is allocated to a processor. The processor determines in constant time if w is the LCA of u and v. This is done by finding whether the child of w that is an ancestor of u, denoted f, and the child of w that is an ancestor of v are different. If they are different, then w, f, and g are as required for the table. The number of internal nodes of the generic subtree is $O(I_{m-1}(n))$, and each has at most $I_{m-1}(n)$ leaves. Thus the number of processors needed for computing the table is $O(I_{m-1}^3(n))$.

The preprocessing algorithm for m. Inductively, we assume that we have an algorithm that preprocesses the array $A = (a_1, a_2, \cdots, a_n)$ for the range-minima problem in $c(m-1)$ time, using $nI_{m-1}(n) + \sqrt{k}n$ processors and $o(nI_{m-1}(n))$ space, where c is a constant, and that after this preprocessing any $\text{MIN}[i, j]$ query can be answered in $c(m-1)$ time. We construct an algorithm that solves the range-minima problem in $c_1 + c(m-1)$ time for some constant c_1, using $nI_m(n) + \sqrt{k}n$ processors and $o(nI_m(n))$ space. We have already shown that a query can be answered in $c_2 m$ time with one processor for some constant c_2. Selecting initially $c > c_1$ and $c > c_2$ implies that the algorithm runs in cm

time, using $nI_m(n) + \sqrt{k}n$ processors and $o(nI_m(n))$ space, and that a query can be answered in cm time.

Step 1. Partition A into subarrays of $I_m^3(n)$ elements each. Allocate $I_m^4(n)$ processors to each subarray, and apply the preprocessing algorithm for range minima given in §3 (for $\epsilon = \frac{1}{3}$). This uses $I_m^4(n)$ processors and $O(I_m^3(n)I_m^{1/3}(n))$ space per subarray and $nI_m(n)$ processors and $o(nI_m(n))$ space overall.

Step 2. Take the minimum in each subarray to build array B of size $n/I_m^3(n)$. The difference between two successive elements in B is at most $kI_m^3(n)$.

Step 3. Build $TRL-BT(m)$, the upper level of a $BT(m)$ tree whose leaves are the elements of B. The definition of $TRL-BT(m)$ implies that each internal node of $TRL-BT(m)$, whose rooted tree has r leaves, has $r/I_{m-1}(r)$ children. For each such internal node v of $TRL–BT(m)$ we keep an array. The array consists of the values of the r leaves of the subtree rooted at v. $TRL–BT(m)$ has $*I_{m-1}(n/I_m^3(n)) + 1 \leq I_m(n) + 1$ levels and thus at most $I_m(n)$ internal levels (where internal levels exclude the leaves but include the root). So the space needed is $O(n/I_m^3(n))$ per level and $O(n/I_m^2(n))$ for all levels of $TRL-BT(m)$. We allocate to each leaf at each internal level $1 + \sqrt{k}I_m^2(n)$ processors, and the total number of processors used is thus $n/I_m^2(n) + \sqrt{k}n$ (which is less than $nI_m(n) + \sqrt{k}n$).

Step 4.1. For each internal node of $TRL-BT(m)$ find the minimum over its array. The difference between the minimum value and the maximum value in an array never exceeds the square of its number of processors, and we apply the constant-time algorithm mentioned in Remark 3.1 as in Step 4 of the recursion-base algorithm.

Step 4.2. We focus on internal node v having r leaves in $TRL-BT(m)$. Each of its $r/I_{m-1}(r)$ children contributes its minimum, and we preprocess these minima by using the assumed algorithm for $m - 1$. The difference between adjacent elements is at most $kI_{m-1}(r)I_m^3(n)$. Thus this computation takes $c(m - 1)$ time using $r + \sqrt{kI_m^3(n)}\,r$ processors. (To see this, simplify $(r/I_{m-1}(r))I_{m-1}(r) + \sqrt{kI_{m-1}(r)I_m^3(n)} \cdot (r/I_{m-1}(r))$, the processor count term for this problem, into $r + \sqrt{kI_m^3(n)}\,r/\sqrt{I_{m-1}(r)}$, which is less than $r + \sqrt{kI_m^3(n)}\,r$ processors.) This amounts to $n/I_m^3(n) + \sqrt{kI_m^3(n)}\,n/I_m^3(n)$ per level or a total of $n/I_m^2(n) + \sqrt{k}n/\sqrt{I_m(n)}$ processors, which is less than $n/I_m^2(n) + \sqrt{k}n$.

Step 5. For each internal node v we compute P_v (S_v is computed similarly). That is, for each leaf b_i of v we need to find $\text{MIN}_B[l(v), i]$, the minimum over the prefix of b_i with respect to the leaves of node v. For this, the minimum among the following list of at most $*I_{m-1}(n) + 1 = I_m(n) + 1$ numbers is computed: Each level l, level$(v) < l \leq I_m(n) + 1$, of the tree contributes (at most) one number. Let u denote the ancestor at level $l - 1$ of b_i, and let $u_1, \cdots, u_y$ denote its children, which are at level l. Suppose $u_j, j > 1$ is an ancestor of b_i. We take the prefix minimum over the leaves of $u_1, \cdots, u_{j-1}$. This prefix minimum is computed in the previous step (by the assumed algorithm for $m - 1$). If u_1 is the ancestor of b_i, then level l contributes a large default value (as in Step 5 of the recursion-base algorithm). Finally, b_i is also added to the list. This minimum computation can be done in constant time by using $I_m^2(n)$ processors (by the algorithm of [SV81]). Note that (1) all prefix minima and all suffix minima with respect to B are computed (in the root) in this step, and (2) given a leaf b_i in $TRL-BT(m)$, finding a node w that is both a child of the root and an ancestor of b_i in constant time was solved in the context of the query retrieval. Node u is found similarly.

Step 6. For each a_i we find its prefix minimum and its suffix minimum with respect to A by using one processor in constant time. This is similar to Step 6 of the recursion base.

This completes the description of the recursive step: Items (1), (2), (3), (4), and (5) of the output were computed (respectively) in steps 1, 2, 5, 4.2, and 6 above.

Complexity of the recursive step. In addition to application of the inductively assumed algorithm, Steps 1 through 6 take constant time and use $nI_m(n) + \sqrt{k}n$ processors and $o(nI_m(n))$ space. This totals cm time if $nI_m(n) + \sqrt{k}n$ processors and $o(nI_m(n))$ space are used. Together with Lemma 4.2.2, Lemma 4.2.1 follows.

From recursion to algorithm. The recursive procedure in Lemma 4.2.1 translates easily into a constructive parallel algorithm in which the instructions for each processor at each time unit are available. For such translation, issues such as processor allocation and computation of certain functions need to be taken into account. Since $TRL-BT(m)$ is balanced, allocating processors in the algorithm above can be done in constant time if the following functions are precomputed: (a) $I_m(x)$ for $1 \leq x \leq n$ and (b) $I^{(i)}_{m-1}(x)$ for $1 \leq x \leq n$ and $1 \leq i \leq I_m(x)$. Let us illustrate how processor allocation is done in Step 3 above. We use $n/I^2_m(n) + \sqrt{k}n$ processors. The processors are partitioned into $I_m(n)$ groups of $n/I^3_m(n) + \sqrt{k}n/I_m(n)$ processors each. Each group is allocated to one level. Within a level, the processors in its group are partitioned equally among the nodes of the level. This will provide a sufficient number of processors to each node. Some processors are superfluous and simply remain idle. These same functions suffice for all other computations above. The functions are computed and stored in a table at the beginning of the algorithm. Section 6 discusses their computation.

4.3. The optimal parallel algorithms. In this subsection we show how to derive a series of optimal parallel algorithms from the series of algorithms described in Lemma 4.2.1. Theorem 4.3.1 gives a general tradeoff result between the running time of the preprocessing algorithm and the retrieval time. Corollary 4.3.1 emphasizes results in cases for which the retrieval time for a query is constant. Corollary 4.3.2 points at an interesting tradeoff instance in which the retrieval time bound is increased to $O(\alpha(n))$ and the preprocessing algorithm runs in $O(\alpha(n))$ (i.e., it become almost fully parallel).

THEOREM 4.3.1. *Consider the range-minima problem where k, the bound on the difference between two successive elements in A, is constant. For each $2 \leq m \leq \alpha(n)$ we present a parallel preprocessing algorithm that runs in time $O(I_m(n))$ and uses an optimal number of processors and optimal linear space. Query retrieval time is $O(m)$ if one processor is used.*

COROLLARY 4.3.1. *When m is constant the preprocessing algorithm runs in $O(I_m(n))$ time if $n/I_m(n)$ processors are used. Retrieval time is $O(1)$ if one processor is used.*

COROLLARY 4.3.2. *When $m = \alpha(n)$ the preprocessing algorithm runs in $O(\alpha(n))$ time if $n/\alpha(n)$ processors are used. Retrieval time is $O(\alpha(n))$ if one processor is used.*

We describe below the optimal preprocessing algorithm for m as per Theorem 4.3.1.

Step 1. Partition A into subarrays of $I^2_m(n)$ elements each, allocate $I_m(n)$ processors to each subarray, and find the minimum in the subarray. This can be done in $O(I_m(n))$ time. Put the $n/I^2_m(n)$ minima into an array B.

Step 2. Out of the series of preprocessing algorithms of Lemma 4.2.1, apply the algorithm for m to B, where k', the difference between two successive elements of B, is $O(I^2_m(n))$. This takes $O(m)$ time if $\sqrt{k'}\,n/I^2_m(n) + n/I_m(n)$ processors and $o(n/I_m(n))$ space are used. This can be simulated in $O(m)$ time by using $n/I_m(n)$ processors and $o(n/I_m(n))$ space.

Step 3. Preprocess each subarray of $I^2_m(n)$ elements so that a range-minimum query within the subarray can be retrieved in $O(1)$ time. This is done by using the following parallel variant of the range-minima algorithm of Gabow, Bentley, and Tarjan (GBT) [GBT84].

Range minimum: a parallel variant of the GBT *algorithm*. Consider the general range-minima problem as defined in §3, with respect to an input array $C = (c_1, c_2, \cdots, c_n)$. We overview a preprocessing algorithm that runs in $O(\sqrt{n})$ time using $\sqrt{n}$ processors, so that a range-minimum query can be processed in constant time.

(1) Partition array C into $\sqrt{n}$ subarrays $C_1, \cdots, C_{\sqrt{n}}$, each with $\sqrt{n}$ elements.

(2) Apply the GBT linear time serial algorithm separately to each C_i, taking $O(\sqrt{n})$ time and using $\sqrt{n}$ processors.

(3) Let $\overline{c_i}$ be the minimum over C_i. Apply the GBT algorithm to $\overline{C} = (\overline{c_1}, \cdots, \overline{c_{\sqrt{n}}})$ in $O(\sqrt{n})$ time, using a single processor.

It should be clear that any range-minimum query with respect to C can be retrieved in constant time by at most three queries with respect to the tables built by the above applications of the GBT algorithm. The complexity of the preprocessing algorithm is specified by $O(I_m(n))$ time if $n/I_m(n)$ processors are used. Retrieval of a range-minimum query takes $O(m+1)$ time, which is $O(m)$ time if one processor is used. Theorem 4.3.1 follows.

4.4. The fully parallel algorithms. Consider the restricted-domain range-minima problem in which k, the bound on the difference between adjacent elements, is constant. In this subsection we present a fully parallel preprocessing algorithm for the problem on a CRCW-bit PRAM that provides for constant time processing of a query. Theorem 4.4.1 gives the general result achieved in this subsection, including tradeoff among parameters. Corollary 4.4.1 summarizes the fully parallel result.

Let d be an integer $2 < d \leq \alpha(n)$. The model of parallel computation is the CRCW-bit PRAM with the assumption that up to $I_d(n)$ processors may write simultaneously into different bits of the same memory word.

THEOREM 4.4.1. *The preprocessing algorithm takes $O(d)$ time if n processors and linear space are used. Retrieval of a query* MIN(i, j) *takes $O(d)$ time if one processor is used.*

Remark. Theorem 4.4.1 represents a tradeoff between the time for the preprocessing algorithm and query retrieval, on one hand, and the number of processors that may write simultaneously into different bits of the same memory word, on the other.

COROLLARY 4.4.1. *For a constant d the algorithm is fully parallel and query retrieval can be done in constant time with one processor.*

Step 1. Partition A into $n/I_d(n)$ subarrays of $I_d(n)$ elements each. For each subarray find the minimum in $O(1)$ time and $I_d(n)$ processors. For this we apply the constant-time algorithm mentioned in Remark 3.1 as in Step 4 of the recursion-base algorithm. Put the $n/I_d(n)$ minima into an array B. The difference between two successive elements in B is at most $kI_d(n)$.

Step 2. Out of the series of preprocessing algorithms of Lemma 4.2.1 apply the algorithm for d to B, where k', the difference between two successive elements of B, is $O(I_d(n))$. This takes $O(d)$ time when $\sqrt{k'}\, n/I_d(n) + n$ processors and $o(n)$ space are used and can be simulated in $O(d)$ time by using n processors and $o(n)$ space.

Suppose we know to retrieve a range-minimum query within each of the subarrays of size $I_d(n)$ in constant time. It should be clear how a query MIN(i, j) can then be retrieved in $O(d)$ time. Theorem 4.4.1 would follow. Thus it remains to show how to preprocess the subarrays of size $I_d(n)$ in constant time such that a range-minimum query within a subarray can be retrieved in constant time. These subarrays are preprocessed in Steps 3.1, 3.2, and 3.3.

Step 3.1. For each subarray subtract the value of its first element from each element of the subarray. Observe that after this subtraction the value of the first element is zero and the difference between each pair of successive elements remains at most k. Step 3.2

constructs a table with the following information: For any $I_d(n)$-tuple $(c_1, \cdots, c_{I_d(n)})$, where $c_1 = 0$ and the difference between each pair of successive c_i values is at most k, the table has an entry. This entry gives all $I_d(n)(I_d(n) - 1)/2$ range minima with respect to this $I_d(n)$-tuple.

Step 3.2. All n processors together build a table. Each entry of the table corresponds to one possible allocation of values to the $I_d(n)$-tuple. The entry provides all $I_d(n)(I_d(n) - 1)/2$ range minima for this allocation. Observe that the number of possible allocations is $(2k(I_d(n) - 1) + 1)^{I_d(n)-1}$. To see this we note that each element in an $I_d(n)$-tuple assumes an integer value in the range $[-k(I_d(n) - 1) \cdots k(I_d(n) - 1)]$. Our table will have $y \geq (2k(I_d(n) - 1) + 1)^{I_d(n)-1}$ entries. It will be more convenient to defer the statement of the exact value for y to the comment below. By using n processors (or even fewer) the table can be built in $O(1)$ time. We sketch below how the table is built. Consider any integer x in the range $[1, y]$, and suppose x is given in binary representation. We break the binary (i.e., bit) representation of x into $I_d(n) - 1$ bit strings of equal length. Each of these bit strings should be long enough to represent integers in the range $[-k(I_d(n) - 1), k(I_d(n) - 1)]$ (namely, a range of $2k(I_d(n) - 1) + 1$ integers). (It is enough to have $\lceil \log(2k(I_d(n) - 1) + 1) \rceil$ bits in each bit string and therefore a total of $(I_d(n) - 1)(\lceil \log(2k(I_d(n) - 1) + 1) \rceil)$ bits for representing the original value of x. So $y = 2^{(I_d(n)-1)(\lceil \log(2k(I_d(n)-1)+1) \rceil)}$.) These integers are then put in an array X (of size $I_d(n) - 1$). It is easy to see that for any $I_d(n)$-tuple (whose first entry is fixed to zero) for which the difference between any two consecutive locations is at most k, there exists an integer x in the range $[1, y]$ whose array X is equal to locations 2 to $I_d(n)$ in the $I_d(n)$-tuple. For each entry x, $1 \leq x \leq y$, in our table we compute separately each range minimum with respect to array X of x. Given a range $[i, j]$ the minimum over $[X_i \cdots X_j]$ is done in constant time with $(j - i)^2$ processors by the algorithm of [SV81]. Since $I_d(n) \leq \log^* n$, the n available processors are more than enough for the necessary computations above.

Step 3.3. The only difficulty is to identify the table entry for our $I_d(n)$-tuple, $c_1, \cdots, c_{I_d(n)}$, since once we reach the entry the table already provides the desired range minimum. We allocate to each subarray $I_d(n)$ processors. For each subarray we have a word in our shared memory with $(I_d(n) - 1)\lceil \log(2k(I_d(n) - 1) + 1) \rceil$ bits. Processor $i, 1 < i \leq I_d(n)$, writes c_i (which is an integer from $[-k(I_d(n) - 1) \cdots k(I_d(n) - 1)]$) starting in bit number $(i - 2)\lceil \log(2k(I_d(n) - 1) + 1) \rceil$ of the word belonging to its subarray (bit zero is the least significant). As a result, this word has a sequence of integers from the range $[-k(I_d(n) - 1), k(I_d(n) - 1)]$ that yields the desired entry in our table. Note that exactly $I_d(n)$ processors write to different bits of the same memory word. Theorem 4.4.1 follows.

4.5. The all-nearest-zero-bit problem. The following corollary of Theorems 4.3.1 and 4.4.1 is needed for §5.

COROLLARY 4.5.1. *The all-nearest-zero-bit problem is almost fully parallel. On the* CRCW-*bit* PRAM *the all-nearest-zero-bit problem is fully parallel.*

Proof. Recall that the algorithm for the restricted-domain range-minima problem computes all suffix minima. Recall also that if the minimum over an interval is not unique, the leftmost minimum is found. Thus if we apply the restricted-domain range-minima algorithm (with the difference between successive elements at most one) with respect to A, then the minimum over the suffix of entry $i + 1$ gives the nearest zero to the right of entry i. Thus the all-nearest-zero bit is actually an instance of the restricted-domain range-minima problem (with the difference between successive elements at most one). It follows that the almost fully parallel and the fully parallel algorithms for the latter apply for the all-nearest-zero-bit problem as well. □

4.6. A simple sequential LCA algorithm. In this subsection we outline a sequential variant of the restricted-domain range-minima problem for which k, the difference between adjacent elements, is one. Together with the reduction of §4.1, this gives a sequential algorithm for the LCA problem.

We first describe two preprocessing procedures for the range-minima problem: (i) Procedure I takes $O(n \log n)$ time for an input array of length n. No assumptions are needed regarding the difference between adjacent elements. (A similar procedure is used in [AS87].) (ii) Procedure II takes exponential time. After each of these preprocessing procedures, query retrieval takes constant time. Second, the sequential linear-time range-minima algorithm is described. Finally, we show how to retrieve a range-minimum query in constant time.

Procedure I. Build a complete binary tree whose leaves are the elements of the input array A. Compute (and keep) for each internal node all prefix minima and all suffix minima with respect to its leaves.

Procedure I clearly runs in $O(n \log n)$ time. Given any range $[i, j]$, the range-minimum query with respect to $[i, j]$ can be processed in constant time, as follows. (1) Find the lowest node u of the binary tree such that the range $[i, j]$ falls within its leaves. This range is the union of a suffix of the left child of u and a prefix of the right child of u. The minima over the suffix and prefix were computed by Procedure I. (2) The answer to the query is the minimum among these two minima.

Procedure II. We use the assumption that the difference between any two adjacent elements of the input array A is exactly one. A table similar to the table built in Step 3 of §4.4 is built as follows. We assume without loss of generality that the value of the first element of A is zero (since otherwise we can subtract from every element in A the value of the first element without affecting the answers to range-minima queries). Then the number of different possible input arrays A is 2^{n-1}. The table will have a subtable for each of these 2^{n-1} possible arrays. For each possible array the subtable will store the answer to each of the $n(n-1)/2$ possible range queries. The time to build the table is $O(2^n n^2)$, and $O(2^n n^2)$ space is needed.

The linear-time range-minima preprocessing algorithm follows.

• For each of the subsets $a_{i \log n+1}, \cdots, a_{(i+1)\log n}$ for $0 \leq i \leq n/\log n - 1$, find its minimum and apply Procedure I to an array of these $n/\log n$ minima.

• Separately, for each of the subsets $a_{i \log n+1}, \cdots, a_{(i+1)\log n}$ for $0 \leq i \leq n/\log n - 1$, do the following. Partition each such subset into smaller subsets of size $\log\log n$, and find the minimum in each smaller subset; apply Procedure I to these $\log n/\log\log n$ minima.

• Run Procedure II to build the table required for an (any) array of size $\log\log n$. For each of the subsets $a_{i \log\log n+1}, \cdots, a_{(i+1)\log\log n}$ for $0 \leq i \leq n/\log\log n - 1$, identify its subtable.

The time (and space) for each step of the preprocessing algorithm is $O(n)$.

Consider a query requesting the minimum over a range $[i, j]$. We show how to process it in constant time. The range $[i, j]$ can easily be presented as the union of the following (at most) five ranges: $[i, x_1]$, $[x_1+1, y_1]$, $[y_1+1, y_2]$, $[y_2+1, x_2]$, and $[x_2+1, j]$, where (1) $[i, x_1]$ (and $[x_2+1, j]$) falls within a single subset of size $\log\log n$—its minimum is available in its subtable, (2) $[x_1+1, y_1]$ (and $[y_2+1, x_2]$) is the union of subsets of size $\log\log n$ and falls within a single subset of size $\log n$—its minimum is available from the application of Procedure I to the subset of size $\log n$, and (3) $[y_1+1, y_2]$ is the union of subsets of size $\log n$—its minimum is available from the first application of Procedure I. So the minimum over range $[i, j]$ is simply the minimum of these five minima.

5. Almost fully parallel reducibility. We demonstrate how to use the $*$-tree data structure for reducing a problem A to another problem B by an almost fully parallel algorithm. We apply this reduction for deriving a parallel lower bound for problem A from a known parallel lower bound for problem B.

Given a convex polygon with n vertices, the *all-nearest-neighbors* (*ANN*) problem is to find for each vertex of the polygon its nearest (Euclidean) neighbor.

THEOREM 5.1. *Any* CRCW PRAM *algorithm for the* ANN *problem that uses* $O(n \log^c n)$ (*for any constant* c) *processors needs* $\Omega(\log \log n)$ time.

Proof. We give below an almost fully parallel reduction from the problem of merging two sorted lists of length n each to the ANN problem with $O(n)$ vertices. This reduction together with the following lemma imply the theorem.

LEMMA 5.1. *Merging two sorted lists of length n each by using* $O(n \log^c n)$ (*for any constant* c) *processors on a* CRCW PRAM *requires* $\Omega(\log \log n)$ time.

A remark in [SV90] implies that Borodin and Hopcroft's [BH85] lower bound for merging in a parallel comparisons model can be extended to yield the lemma.

Proof of Theorem 5.1 (*continued*).

The reduction (*see Fig*. 3). Let $A = (a_1, a_2, \cdots, a_n)$ and $B = (b_1, b_2, \cdots, b_n)$ be two increasing lists of numbers that we wish to merge. Assume, without loss of generality, that the numbers are integers and that $a_1 = b_1$, $a_n = b_n$. (The lower bound for merging assumes that the numbers are integers.) Consider the following *auxiliary* problem: For each $1 \leq i \leq n$ find the minimum index j such that $b_j > a_i$. The position of a_i in the merged list is $i + j - 1$, and therefore an algorithm for the auxiliary problem (together with a similar algorithm for finding the positions of the b_i numbers in the merged list) suffices for the merging problem.

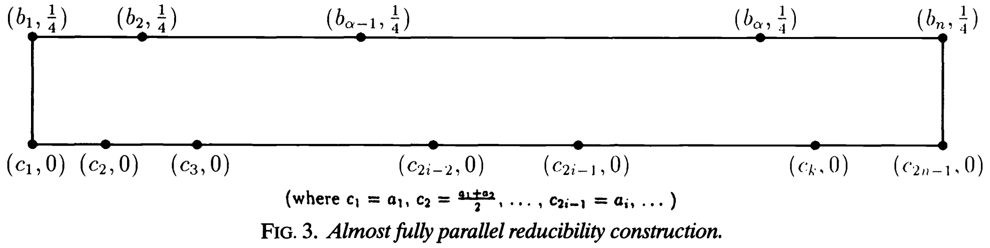


FIG. 3. *Almost fully parallel reducibility construction.*

We give an almost fully parallel reduction from the auxiliary problem to the ANN problem with respect to the following convex polygon. Let $(c_1, c_2, \cdots, c_{2n-1}) = (a_1, (a_1 + a_2)/2, a_2, (a_2 + a_3)/2, \cdots, a_{n-1}, (a_{n-1} + a_n)/2, a_n)$. The numbers $c_1, c_2, \cdots, c_{2n-1}$ form an increasing list. The convex polygon is $(c_1, 0), (c_2, 0), \cdots, (c_{2n-1}, 0), (b_n, \frac{1}{4}), (b_{n-1}, \frac{1}{4}), \cdots, (b_1, \frac{1}{4})$.

In [SV90] a similar construction is given, and the lower-bound proof then follows by (nontrivial) Ramsey theoretic arguments.

Let $D[1, \cdots, 2n-1]$ be a binary vector. Each vertex $(c_l, 0)$ finds its nearest neighbor with respect to the convex polygon (by using a "supposedly existing" algorithm for the ANN problem) and assigns the following into vector D.

```
If the nearest vertex is of the form$(b_k, \frac{1}{4})$
Then $D(l) := 0$
Else $D(l) := 1$
```

Next we apply to vector D the almost fully parallel algorithm for the nearest-zero-bit problem of §4.5. Finally, we show how to solve our auxiliary problem with respect to every element a_i. We break into two cases concerning the nearest neighbor of $(a_i, 0)$ $(= (c_{2i-1}, 0))$.

Case (i). The nearest neighbor of $(a_i, 0)$ is a vertex $(b_\alpha, \frac{1}{4})$. Then the minimum index j such that $b_j > a_i$ is either α or $\alpha + 1$. A single processor can determine the correct value of j in $O(1)$ time.

Case (ii). Otherwise, then $D(2i - 1) = 1$. The nearest-zero computation gives the smallest index $k > 2i - 1$ for which $D(k) = 0$. Let the nearest neighbor of $(c_k, 0)$ be $(b_\alpha, \frac{1}{4})$. Then $j = \alpha$ is the minimum index for which $b_j > a_i$. $\square$

6. Computing various functions. We need to compute certain functions in our algorithms. For each $2 < m \leq \alpha(n)$ we need $I_{m-1}^{(i)}(n)$ for all $1 \leq i \leq I_m(n)$. Fortunately, the function parameters that we are actually concerned with are small relative to n. For instance, to compute $I_3(n) = \log^* n$, it is enough to compute $\log^* (\log\log n)$ since $\log^* n = \log^* (\log\log n) + 2$.

We show only how to compute $I_m(n)$ for each $2 \leq m \leq \alpha(n)$. Computation of $I_{m-1}^{(i)}(n)$ for $2 < m \leq \alpha(n)$ and all $1 \leq i \leq I_m(n)$ is similar. Our computation works by induction on m.

The inductive hypothesis. Let $x = \log\log n$. We know how to compute the following values in $O(m - 1)$ time by using $o(n/\alpha(n))$ processors: (1) $I_{m-1}(n)$ and (2) $I_{m-1}(y)$ for all $1 \leq y \leq \log\log n$.

We show the inductive claim (the claim itself should be clear) by assuming the inductive hypothesis. The inductive base is given later. First, we describe (informally) the computation of $I_m(x)$ in $O(1)$ (additional) time using $o(n/\alpha(n))$ processors. Consider all permutations of the numbers $1, \cdots, x$. The number of these permutations is (much) less than n. The idea is to identify a permutation that provides the sequence $[x, I_{m-1}(x), I_{m-1}^{(2)}(x), \cdots, I_{m-1}^{(k)}(x) = 1, \cdots]$. So if $I_{m-1}(p_i) = p_{i+1}$ for all $0 \leq i \leq k - 1$, we conclude that $I_m(x) = k$. We can check this condition in $O(1)$ time with x processors per permutation by using the ability of the CRCW PRAM to find the AND of x bits in $O(1)$ time. The total number of processors is $o(n/\alpha(n))$. We make two remarks: (1) Computing $I_m(y)$ for all $1 \leq y \leq \log\log n$, the rest of the inductive claim, in $O(1)$ time with $o(n/\alpha(n))$ processors is similar; (2) there are easy ways for finding all permutations in $O(1)$ time by using the number of available processors.

We finish by showing the inductive base. We compute $\log n$ in $O(1)$ time and with $o(n/\alpha(n))$ processors as follows. If n is given in a binary representation, then the index of the leftmost one is $\log n$. Following [FRW84], this can be computed in $O(1)$ time by using as many processors as the number of bits of a number. By iterating this we obtain $\log^{(2)} n$. Finally, we find $\log y$ for all $1 \leq y \leq \log\log n$. The number of processors used for this computation is $o(n/\alpha(n))$.

Acknowledgments. We are grateful to Pilar de la Torre and to Baruch Schieber for fruitful discussions and helpful comments.

REFERENCES

[AIL$^+$88] A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin, *Parallel construction of a suffix tree with applications*, Algorithmica, 3 (1988), pp. 347–365.

[AM88] R. J. Anderson and G. L. Miller, *Deterministic parallel list ranking*, in Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988, pp. 81–90.

[AMW89] R. J. Anderson, E. W. Mayr, and M. K. Warmuth, *Parallel approximation algorithms for bin packing*, Inform. and Comput., 82 (1989), pp. 262–277.

[AS87] N. Alon and B. Schieber, *Optimal preprocessing for answering on-line product queries*, Tech. Report TR 71/87, Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University, Tel Aviv, Israel, 1987.

[BBG+89] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, *Highly-parallelizable problems*, in Proc. 21st Annual ACM Symposium on Theory of Computing, 1989, pp. 309–319.

[BH85] A. Borodin and J. E. Hopcroft, *Routing, merging, and sorting on parallel models of computation*, J. Comput. System Sci., 30 (1985), pp. 130–145.

[BH87] P. Beame and J. Hastad, *Optimal bounds for decision problems on the* CRCW PRAM, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 83–93.

[BSV88] O. Berkman, B. Schieber, and U. Vishkin, *Some doubly logarithmic optimal parallel algorithms based on finding nearest smaller values*, Tech. Report UMIACS-TR-88-79, University of Maryland Institute for Advanced Computer Studies, College Park, MD, 1988; also IBM Res. Report, Computer Sciences, RC 14128 (#63291); J. Algorithms, to appear.

[BV85] I. Bar-On and U. Vishkin, *Optimal parallel generation of a computation tree form*, ACM Trans. Prog. Lang. and Systems, 7 (1985), pp. 348–357.

[BV90] O. Berkman and U. Vishkin, *On parallel integer merging*, Tech. Report UMIACS-TR-90-15.1 (revised version), University of Maryland Institute for Advanced Computer Studies, College Park, MD, 1990; Inform. and Comput., to appear.

[BV91] O. Berkman and U. Vishkin, *Almost fully parallel parentheses matching*, Tech. Report UMIACS-TR-91-103, University of Maryland Institute for Advanced Computer Studies, College Park, MD, 1991.

[CFL83] A. K. Chandra, S. Fortune, and R. J. Lipton, *Unbounded fan-in circuits and associative functions*, in Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 52–60.

[CV86a] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proc. 27th IEEE Annual Symposium on Foundations of Computer Science, 1986, pp. 478–491.

[CV86b] ———, *Deterministic coin tossing with applications to optimal parallel list ranking*, Inform. and Control, 70 (1986), pp. 32–53.

[CV89] ———, *Faster optimal parallel prefix sums and list ranking*, Inform. and Comput., 81 (1989), pp. 334–352.

[DS83] E. Dekel and S. Sahni, *Parallel generation of postfix and tree forms*, ACM Trans. on Prog. Lang. and Systems, 5 (1983), pp. 300–317.

[FRT89] D. Fussell, V. Ramachandran, and R. Thurimella, *Finding triconnected components by local replacements*, Lecture Notes in Computer Science 372, Springer-Verlag, Berlin, New York, 1989.

[FRW84] F. E. Fich, P. L. Ragde, and A. Wigderson, *Relations between concurrent-write models of parallel computation* (preliminary version), in Proc. 3rd ACM Symposium on Principles of Distributed Computing, 1984, pp. 179–189; also SIAM J. Comput., 17 (1988), pp. 606–627.

[GBT84] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, *Scaling and related techniques for geometry problems*, in Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 135–143.

[Has86] J. Hastad, *Almost optimal lower bounds for small depth circuits*, in Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 6–20.

[HS86] S. Hart and M. Sharir, *Nonlinearity of Davenport–Schinzel sequences and generalized path compression schemes*, Combinatorica, 6 (1986), pp. 151–177.

[HT84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[Kru83] C. P. Kruskal, *Searching, merging, and sorting in parallel computation*, IEEE Trans. Comput., C-32 (1983), pp. 942–946.

[LF80] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980) pp. 831–838.

[LV89] G. M. Landau and U. Vishkin, *Efficient parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 157–169.

[MSV86] Y. Maon, B. Schieber, and U. Vishkin, *Parallel ear decomposition search* (*EDS*) *and st-numbering in graphs*, Theoret. Comput. Sci., 47 (1986), pp. 277–298.

[RR89] V. Ramachandran and J. H. Reif, *An optimal parallel algorithm for graph planarity*, in Proc. 30th IEEE Annual Symposium on Foundations of Computer Science, 1989, pp. 282–287.

[Sto88] Q. F. STOUT, *Constant-time geometry on PRAMs*, in Proc. International Conference on Parallel Processing, 1988, pp. 104–107.

[SV81] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging, and sorting in a parallel computation model*, J. Algorithms, 2 (1981), pp. 88–102.

[SV84] L. STOCKMEYER AND U. VISHKIN, *Simulation of parallel random access machines by circuits*, SIAM J. Comput., 13 (1984), pp. 409–422.

[SV88] B. SCHIEBER AND U. VISHKIN, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[SV90] ———, *Finding all-nearest neighbors for convex polygons in parallel: A new lower bound technique and a matching algorithm*, Discrete Appl. Math., 29 (1990), pp. 97–111.

[SZ89] D. SHASHA AND K. ZHANG, *New algorithms for the editing distance between trees*, in Proc. 1st ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 117–126, to appear in J. Algorithms as *Fast algorithms for the unit cost editing distance between trees*.

[Tar75] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.

[TV85] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithms*, SIAM J. Comput., 14 (1985), pp. 862–874.

[Val75] L. G. VALIANT, *Parallelism in comparison problems*, SIAM J. Comput., 4 (1975), pp. 348–355.

[Van89] A. VAN GELDER, *PRAM processor allocation: A hidden bottleneck in sublogarithmic algorithms*, IEEE Trans. Comput., 38 (1989), pp. 289–292.

[Vis85] U. VISHKIN, *On efficient parallel strong orientation*, Inform. Process. Lett., 20 (1985), pp. 235–240.

[Vis91] ———, *Deterministic sampling–a new technique for fast pattern matching*, SIAM J. Comput., 20 (1991), pp. 22–40.

[Yao82] A. C. YAO, *Space–time tradeoff for answering range queries*, in Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 128–136.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 243–260, April 1993

APPROXIMATE BOYER–MOORE STRING MATCHING*

JORMA TARHIO† AND ESKO UKKONEN†

Abstract. The Boyer–Moore idea applied in exact string matching is generalized to approximate string matching. Two versions of the problem are considered. The k mismatches problem is to find all approximate occurrences of a pattern string (length m) in a text string (length n) with at most k mismatches. The generalized Boyer–Moore algorithm is shown (under a mild independence assumption) to solve the problem in expected time $O(kn(1/(m-k)+(k/c)))$, where c is the size of the alphabet. A related algorithm is developed for the k differences problem, where the task is to find all approximate occurrences of a pattern in a text with $\leq k$ differences (insertions, deletions, changes). Experimental evaluation of the algorithms is reported, showing that the new algorithms are often significantly faster than the old ones. Both algorithms are functionally equivalent with the Horspool version of the Boyer–Moore algorithm when $k = 0$.

Key words. string matching, edit distance, Boyer–Moore algorithm, k mismatches problem, k differences problem

AMS(MOS) subject classifications. 68C05, 68C25, 68H05

1. Introduction.

The fastest known exact string matching algorithms are based on the Boyer–Moore idea [BoM77], [KMP77]. Such algorithms are "sublinear" on the average in the sense that it is not necessary to check every symbol in the text. The larger the alphabet and the longer the pattern, the faster the algorithm works. In this paper we generalize this idea to approximate string matching. Again the approach leads to algorithms that are significantly faster than the previous solutions of the problem.

We consider two important versions of the approximate string matching problem. In both, we are given two strings, the *text* $T = t_1t_2 \ldots t_n$ and the *pattern* $P = p_1p_2 \ldots p_m$ in some alphabet Σ, and an integer k. In the first variant, called the *k mismatches problem*, the task is to find all occurrences of P in T with at most k mismatches, that is, all j such that $p_i = t_{j-m+i}$ for $i = 1, \ldots, m$ except for at most k indexes i.

In the second variant, called the *k differences problem*, the task is to find (the endpoints of) all substrings P' of T with the edit distance at most k from P. The edit distance means the minimum number of editing operations (the differences) needed to convert P' to P. An editing operation is either an insertion, a deletion, or a change of a character. The k mismatches problem is a special case with the change as the only editing operation.

There are several algorithms proposed for these two problems; see, e.g., the survey [GaG88]. Both can be solved in time $O(mn)$ by dynamic programming [Sel80], [Ukk85b]. A very simple improvement giving $O(kn)$ expected time solution for random strings is described in [Ukk85b]. Later, Landau and Viskin [LaV88], [LaV89], Galil and Park [GaP89], Ukkonen and Wood [UkW90] have given different algorithms that consist of preprocessing the pattern in time $O(m^2)$ (or $O(m)$) and scanning the text in worst-case time $O(kn)$. For the k differences problem, $O(kn)$ is the best bound currently known if the preprocessing is allowed to be at most $O(m^2)$. For the k mismatches problem Kosaraju [Kos88] gives an $O(n\sqrt{m}\ \text{polylog}(m))$ algorithm. Also see [GaG86], [GrL89].

We develop a new approximate string matching algorithm of Boyer–Moore type for the k mismatches problem and show, under a mild independence assumption, that it processes a random text in expected time $O(kn(1/(m-k) + (k/c)))$ where c de-

*Received by the editors April 11, 1990; accepted for publication (in revised form) October 22, 1991.
†Department of Computer Science, University of Helsinki, Teollisuuskatu 23, SF-00510 Helsinki, Finland.

notes the size of the alphabet. A related but different method is (independently) developed and analyzed in [Bae89a]. We also give an algorithm for the k differences problem and show in a special case that its expected processing time for a random text is $O(c/(c-2k)kn(k/(c+2k^2)+(1/m)))$. The preprocessing of the pattern needs time $O(m+kc)$ and $O((k+c)m)$, respectively. We have also performed extensive experimental comparison of the new methods with the old ones, showing that Boyer–Moore algorithms are significantly faster, for large m and c in particular.

Our algorithms can be considered as generalizations of the Boyer–Moore algorithm for exact string matching because they are functionally identical with the Horspool version [Hor80] of the Boyer–Moore algorithm when $k = 0$. The algorithm of [Bae89a] generalizes the original Boyer–Moore idea for the k mismatches problem.

All these algorithms are "sublinear" in the sense that it is not necessary to examine every text symbol. Another approximate string matching method of this type (based on totally different ideas) has recently been given in [ChL90].

The paper is organized as follows. We first consider the k mismatches problem for which we give and analyze the Boyer–Moore solution in §2. Section 3 develops an extension to the k differences problem and outlines an analysis. Section 4 reports our experiments.

2. The k mismatches problem.

2.1. Boyer–Moore–Horspool algorithm. The characteristic feature of the Boyer–Moore algorithm [BoM77] for exact matching of string patterns is the right-to-left scan over the pattern. At each alignment of the pattern with the text, characters of the text below the pattern are examined from right to left, starting by comparing the rightmost character of the pattern with the character in the text currently below it. Between alignments, the pattern is shifted from left to right along the text.

In the original algorithm the shift is computed using two heuristics: the match heuristic and the occurrence heuristic. The *match* heuristic implements the requirement that after a shift, the pattern has to match all the text characters that were found to match at the previous alignment. The *occurrence* heuristic implements the requirement that we must align the rightmost character in the text that caused the mismatch with the rightmost character of the pattern that matches it. After each mismatch, the algorithm chooses the larger shift given by the two heuristics.

Since the patterns are not periodic on the average, the match heuristic is not very useful. A simplified version of the method can be obtained by using the occurrence heuristic only. Then we may observe that it is not necessary to base the shift on the text symbol that caused the mismatch. Any other text character below the current pattern position will do as well. Then the natural choice is the text character corresponding to the rightmost character of the pattern as it potentially leads to the longest shifts. This simplification was noted by Horspool [Hor80]. We call this method the Boyer–Moore–Horspool or the BMH algorithm.

The BMH algorithm has a simple code and is in practice better than the original Boyer–Moore algorithm. In the preprocessing phase the algorithm computes from the pattern $P = p_1p_2\dots p_m$ the shift table d, defined for each symbol a in alphabet Σ as

$$d[a] = \min\{s \mid s = m \text{ or } (1 \leq s < m \text{ and } p_{m-s} = a)\}.$$

For a text symbol a below p_m, the table d shifts the pattern right until the rightmost a in $p_1 \dots p_{m-1}$ becomes above the a in the text. Table d can be computed in time $O(m+c)$, where $c = |\Sigma|$, by the following algorithm.

ALGORITHM 1. BMH-preprocessing.

```
for $a$ in $\Sigma$ do $d[a] := m$;
for $i := 1, \ldots, m-1$ do $d[p_i] := m - i$.
```

The total BMH method [Hor80] including the scanning of the text $T = t_1 t_2 \ldots t_n$ is given below.

ALGORITHM 2. The BMH method for exact string matching.

```
call Algorithm 1;
$j := m$;                                   {pattern ends at text position $j$}
while $j \le n$ do begin
    $h := j; i := m$;                       {$h$ scans the text, $i$ the pattern}
    while $i > 0$ and $t_h = p_i$ do begin
        $i := i - 1; h := h - 1$ end;       {proceed to the left}
    if $i = 0$ then report match at position $j$;
    $j := j + d[t_j]$ end                   {shift to the right}
```

2.2. Generalized BMH algorithm. The generalization of the BMH algorithm for the k mismatches problem will be very natural: for $k = 0$ the generalized algorithm is exactly as Algorithm 2. Recall that the k mismatches problem asks for finding all occurrences of P in T such that in at most k positions of P, T and P have different characters.

We have to generalize both the right-to-left scanning of the pattern and the computation of the shift. The former is very simple; we just scan the pattern to the left until we have found $k+1$ mismatches (unsuccessful search) or the pattern ends (successful search).

To understand the generalized shift it may be helpful to look at the k mismatches problem in a tabular form. Let M be a $m \times n$ table such that for $1 \le i \le m, 1 \le j \le n$,

$$M[i,j] = \begin{cases} 0 & \text{if } p_i = t_j, \\ 1 & \text{if } p_i \ne t_j. \end{cases}$$

There is an exact match ending at position r of T if $M[i, r-m+i] = 0$ for $i = 1, \ldots, m$, that is, there is a whole diagonal of 0's in M ending at $M[m,r]$. Similarly, there is an approximate match with $\le k$ mismatches if the diagonal contains at most k 1's. This implies that any successive $k+1$ entries of such a diagonal have to contain at least one 0.

Assume then that the pattern is ending at text position j and we have to compute the next shift. We consider the last $k+1$ text characters below the pattern, the characters $t_{j-k}, t_{j-k+1}, \ldots, t_j$. Then, suggested by the above observation, we glide the pattern to the right until there is at least one match in $t_{j-k}, t_{j-k+1}, \ldots, t_j$. The maximum shift is $m-k$. Clearly this is a correct heuristic: A smaller shift would give an unsuccessful alignment because there are at least $k+1$ mismatches, and a shift larger than $m-k$ would skip over a potential match.

Let $d(t_{j-k}, t_{j-k+1}, \ldots, t_j)$ denote the length of the shift. The values of $d(t_{j-k}, \ldots, t_j)$ could be precomputed and tabulated. This would lead to quite heavy preprocessing of at least time $\Theta(c^k)$. Instead, we apply a simpler preprocessing that makes it possible to compute the shift on-the-fly with small overhead while scanning.

In terms of M the shifting means finding the first diagonal above the current diagonal such that the new diagonal has at least one 0 for $t_{j-k}, t_{j-k+1}, \ldots, t_j$.

For example, consider table M in Fig. 1, where we assume that $k = 1$. We may shift from the diagonal of $M[1, 1]$ directly to the diagonal of $M[1, 3]$, as this diagonal contains the first 0 for characters $t_3 = a, t_4 = a$. Hence $d(a, a) = 2$ for the pattern *abbb*. Also note that t_4 alone would give a shift of 3 and t_3 a shift of 2, and $d(t_3, t_4)$ is the minimum over these component shifts.

$$
\begin{array}{cc|cccccccccccc|}
 & & \multicolumn{12}{c}{T} \\
 & & \mathtt{a} & \mathtt{b} & \mathtt{a} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{b} & \mathtt{b} & \mathtt{b} & \mathtt{a} \\
\hline
 & \mathtt{a} & \textcircled{0} & 1 & \textcircled{0} & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
P & \mathtt{b} & 1 & \textcircled{0} & 1 & \textcircled{1} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 & \mathtt{b} & 1 & 0 & \textcircled{1} & 1 & \textcircled{1} & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 & \mathtt{b} & 1 & 0 & 1 & \textcircled{1} & 1 & \textcircled{0} & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
$$

$$\longrightarrow$$

FIG. 1. *Determining of shift* ($k = 1$).

In general, we compute $d(t_{j-k}, \ldots, t_j)$ as the minimum of the component shifts for each $t_{j-k}, \ldots, t_j$. The component shift for t_h depends both on the character t_h itself and on its position below the pattern. Possible positions are $m - k, m - k + 1, \ldots, m$. Hence we need a $(k + 1) \times c$ table d_k defined for each $i = m - k, \ldots, m$, and for each a in Σ, as

$$d_k[i, a] = \min \{s | s = m \text{ or } (1 \leq s < m \text{ and } p_{i-s} = a)\}.$$

Here the values greater than $m - k$ are not actually relevant. Table d_k is presented in this form, because the same table is used in the algorithm solving the k differences problem.

Table d_k can be computed in time $O((m + c)k)$ by a straightforward generalization of the BMH-preprocessing, which scans $k + 1$ times over P, and each scanning creates a new row of d_k.

A more efficient method needs only one scan, from right to left, over P. For each symbol p_i encountered, the corresponding updates are made to d_k. To keep track of the updates already made, we use a table *ready*$[a]$, a in Σ, such that *ready*$[a] = j$ if $d_k[i, a]$ already has its final value for $i = m, m - 1, \ldots, j$. Initially, *ready*$[a] = m + 1$ for all a, and $d_k[i, a] = m$ for all i, a. The algorithm is as follows.

ALGORITHM 3. Computation of table d_k.

```
1.    for $a$ in $\Sigma$ do ready$[a] := m + 1$;
2.    for $a$ in $\Sigma$ do
3.        for $i := m$ downto $m - k$ do
4.            $d_k[i, a] := m$;
5.    for $i := m - 1$ downto 1 do begin
6.        for $j :=$ ready$[p_i] - 1$ downto $\max(i, m - k)$ do
7.            $d_k[j, p_i] := j - i$;
8.        ready$[p_i] := \max(i, m - k)$ end
```

The initializations in steps 1–4 take time $O(kc)$. Steps 5–8 scan over P in time $O(m)$ plus the time of the updates of d_k in step 7. This takes time $O(kc)$ as each $d_k[j, p_i]$ is updated at most once. Hence Algorithm 3 runs in time $O(m + kc)$.

We now have the following total method for the k mismatches problem.

ALGORITHM 4. Approximate string matching with k mismatches.

```
1.  compute table $d_k$ from $P$ with Algorithm 3;
2.  $j := m$;                                   {pattern ends at text position $j$}
3.  while $j \le n + k$ do begin
4.      $h := j; i := m; neq := 0$;             {$h$ scans the text, $i$ the pattern}
5.      $d := m - k$;                           {initial value of the shift}
6.      while $i > 0$ and $neq \le k$ do begin
7.          if $i \ge m - k$ then $d := \min(d, d_k[i, t_h])$;
                                        {minimize over the component shifts}
8.          if $t_h \ne p_i$ then $neq := neq + 1$;
9.          $i := i - 1; h := h - 1$ end;       {proceed to the left}
10.     if $neq \le k$ then report match at position $j$;
11.     $j := j + d$ end                        {shift to the right}
```

2.3. Analysis. First recall that the preprocessing of P by Algorithm 3 takes time $O(m + kc)$ and space $O(kc)$. The scanning of T by Algorithm 4 obviously needs $O(mn)$ time in the worst case. The bound is strict for example for $T = a^n, P = a^m$.

Next we analyze the scanning time in the average case. The analysis will be done under the random string assumption which says that individual characters in P and T are chosen independently and uniformly from Σ. The time requirement is proportional to the number of the text-pattern comparisons in step 8 of Algorithm 4. Let $C_{loc}(P)$ be a random variable denoting, for some fixed c and k, the number of such comparison for some alignment of pattern P between two successive shifts, and let $\bar{C}_{loc}(P)$ be its expected value.

LEMMA 1.

$$\bar{C}_{loc}(P) < \left(\frac{c}{c-1} + 1\right)(k+1).$$

Proof. The distribution of $C_{loc}(P) - (k+1)$ converges to the negative binomial distribution (the Pascal distribution) with parameters $(k+1, 1-(1/c))$ when $m \to \infty$, because $C_{loc}(P) - (k+1)$ is the number of matches until we find the $k+1$st mismatch; the probability of the mismatch is $1 - (1/c)$. As the expected value of $C_{loc}(P)$ increases with m, the expected value $(k+1)/(c-1)$ of this negative binomial distribution (see, e.g., [Fel65]) would be an upper bound (and the limit as $m \to \infty$) of $\bar{C}_{loc}(P) - (k+1)$. This, however, ignores the effect of the fact that after a shift of length $d < m - k$ we know that at least one and at most $k+1$ of characters $P_{m-d-k}, \ldots, P_{m-d}$ will match. Hence to bound $\bar{C}_{loc}(P) - (k+1)$ properly, it surely suffices to add $k+1$ to the above bound that gives

$$\bar{C}_{loc}(P) - (k+1) < \frac{k+1}{c-1} + k + 1,$$

and the lemma follows. □

Let $S(P)$ be a random variable denoting the length of the shift in Algorithm 4 for pattern P and for some fixed k and c when scanning a random T. Moreover, let P_0 be a pattern that repeatedly contains all characters in Σ in some fixed order until the length of P_0 equals m. Then it is not difficult to see that P_0 gives on the average the minimal shift, that is, the expected values satisfy $\bar{S}(P_0) \leq \bar{S}(P)$ for all P of length m. Hence a lower bound for $\bar{S}(P_0)$ gives a lower bound for the expected shift over all patterns of length m (cf. [Bae89b]).

LEMMA 2. $\bar{S}(P_0) \geq \frac{1}{2}\min(c/(k+1), m-k)$. *Moreover,* $\bar{S}(P_0) \geq 1$.

Proof. Let $t = \min(c-1, m-k-1)$. Then the possible lengths of a shift are $1, 2, \ldots, t+1$. Therefore,

$$\bar{S}(P_0) = \sum_{i=0}^{t} Pr(S(P_0) > i),$$

where $Pr(A)$ denotes the probability of event A. Then

$$Pr(S(P_0) > i) = \left(\frac{c-i}{c}\right)^{k+1}$$

because for each of the $k+1$ text symbols that are compared with the pattern to determine the shift (step 8 of Algorithm 4), there are i characters not allowed to occur as text symbols. Otherwise the shift would not be $> i$. Hence

$$\bar{S}(P_0) = \sum_{i=0}^{t} \left(1 - \frac{i}{c}\right)^{k+1},$$

which is clearly ≥ 1, because $t \geq 0$ as we may assume that $c \geq 2$ and that $k < m$.

We divide the rest of the proof into two cases.

Case 1. $m - k < c/(k+1)$. Then $t = m - k - 1$, and we have

$$\begin{aligned}\bar{S}(P_0) &\geq \sum_{i=0}^{m-k-1} \left(1 - \frac{k+1}{c} \cdot i\right) \\ &= m - k - \frac{k+1}{c} \cdot \frac{(m-k-1)(m-k)}{2} \\ &\geq (m-k)\left(1 - \frac{k+1}{c} \cdot \frac{m-k}{c}\right) \geq \frac{1}{2}(m-k).\end{aligned}$$

Case 2. $m - k \geq c/(k+1)$. Then $t \geq \lceil c/(k+1) \rceil - 1$, and we have

$$\begin{aligned}\bar{S}(P_0) &\geq \sum_{i=0}^{\lceil \frac{c}{k+1} \rceil - 1} \left(1 - \frac{i}{c}\right)^{k+1} \geq \sum_{i=0}^{\lceil \frac{c}{k+1} \rceil - 1} \left(1 - \frac{k+1}{c} \cdot i\right) \\ &= \left\lceil \frac{c}{k+1} \right\rceil - \frac{k+1}{c} \cdot \frac{1}{2} \cdot \left\lceil \frac{c}{k+1} \right\rceil \left(\left\lceil \frac{c}{k+1} \right\rceil - 1\right) \\ &\geq \left\lceil \frac{c}{k+1} \right\rceil \left(1 - \frac{1}{2} \cdot \frac{k+1}{c} \cdot \frac{c}{k+1}\right) = \frac{1}{2}\left\lceil \frac{c}{k+1} \right\rceil. \qquad \square\end{aligned}$$

Consider finally the total expected number $\bar{C}(P)$ of character comparisons when Algorithm 4 scans a random T with pattern P. Let $f(P)$ be the random variable denoting the number of shifts taken during the execution, and let $\bar{f}(P)$ be its expected value. Then we have

$$\bar{C}(P) = \bar{f}(P) \cdot \bar{C}_{loc}(P).$$

To estimate $\bar{f}(P)$, we let S_i be a random variable denoting the length of ith shift. At the start of Algorithm 4, P is aligned with T such that its first symbol corresponds to the text position 1, and at the end P is aligned such that its first symbol corresponds to some text position $\leq n-m+k+1$, but the next shift would lead to a position $> n-m+k+1$. Hence new shifts are taken until the total length of the shifts exceeds $n-m+k$. This implies that $f(P)$ equals the largest index Φ such that

$$\sum_{i=1}^{\Phi} S_i \leq n-m+k.$$

Assume now that the different variables S_i are independent, that is, the shift lengths are independent; note that this simplification is not true for two successive shifts such that the first one is shorter than $k+1$. Then all variables S_i have a common distribution with expected value $\bar{S}(P) \geq \bar{S}(P_0)$. Under this assumption,

$$\left\{ \sum_{i=1}^{\Phi} S_i \right\}$$

is, in fact, a *pure renewal process* within interval $[0, n-m+k]$ in the terminology of [Fel66, Chap. XI]. Then the expected value of Φ is $(n-m+k)/\bar{S}(P)$ for large $n-m+k$ (see [Fel66, p. 359]). Hence

$$\bar{f}(P) = O\left(\frac{n-m+k}{\bar{S}(P_0)}\right),$$

and by Lemma 2,

$$\bar{f} = O\left(\max\left(\frac{k+1}{c}, \frac{1}{m-k}\right) \cdot (n-m+k)\right).$$

Recalling finally that $\bar{C}(P) = \bar{f}(P) \cdot \bar{C}_{loc}(P)$ and applying Lemma 1, we obtain that

$$\bar{C}(P) \leq O\left(\max\left(\frac{k+1}{c}, \frac{1}{m-k}\right)(n-m+k)\left(\frac{c}{c-1}+1\right)(k+1)\right),$$

which is

$$O\left(\frac{nk^2}{c} + \frac{nk}{m-k}\right) \quad \text{as } n >> m.$$

Hence we have the following.

THEOREM 1. *The expected running time of Algorithm* 4 *is* $O(nk((k/c)+1/(m-k)))$, *if the lengths of different shifts are mutually independent. The preprocessing time is* $O(m+kc)$, *and the working space is* $O(kc)$.

Removing the independence assumption from Theorem 1 remains open.

3. The k differences problem.

3.1. Basic solution by dynamic programming.

The *edit distance* [WaF75], [Ukk85a] between two strings, A and B, can be defined as the minimum number of editing steps needed to convert A to B. Each editing step is a rewriting step of the form $a \rightarrow \epsilon$ (a deletion), $\epsilon \rightarrow b$ (an insertion), or $a \rightarrow b$ (a change) where a, b are in Σ and ϵ is the empty string.

The *k differences problem* is, given pattern $P = p_1 p_2 \ldots p_m$ and text $T = t_1 t_2 \ldots t_n$ and an integer k, to find all such j that the edit distance (i.e., the number of differences) between P and some substring of T ending at t_j is at most k. The basic solution of the problem is by the following dynamic programming method [Sel80], [Ukk85b]: Let D be an $m+1$ by $n+1$ table such that $D(i,j)$ is the minimum edit distance between $p_1 p_2 \ldots p_i$ and any substring of T ending at t_j. Then

$$D(0,j) = 0, \quad 0 \le j \le n;$$
$$D(i,j) = \min \begin{cases} D(i-1,j)+1, \\ D(i-1,j-1) + \textbf{if } p_i = t_j \textbf{ then } 0 \textbf{ else } 1, \\ D(i,j-1)+1. \end{cases}$$

Table D can be evaluated column-by-column in time $O(mn)$. Whenever $D(m,j)$ is found to be $\le k$ for some j, there is an approximate occurrence of P ending at t_j with edit distance $D(m,j) \le k$. Hence j is a solution to the k differences problem.

3.2. Boyer–Moore approach. Our algorithm contains two main phases: the scanning and the checking. The scanning phase scans over the text and marks the parts that contain all the approximate occurrences of P. This is done by marking some entries $D(0,j)$ on the first row D. The checking phase then evaluates all diagonals of D whose first entries are marked. This is done by the basic dynamic programming restricted to the marked diagonals. Whenever the dynamic programming refers to an entry outside the diagonals, the entry can be taken to be ∞. Because this is quite straightforward we do not describe it in detail. Rather, we concentrate on the scanning part.

The scanning phase repeatedly applies two operations: mark and shift. The shift operation is based on a Boyer–Moore idea. The mark operation decides whether or not the current alignment of the pattern with the text needs accurate checking by dynamic programming and in the positive case marks certain diagonals. To understand the operations we need the concept of a minimizing path in table D.

For every $D(i,j)$, there is a *minimizing arc* from $D(i-1,j)$ to $D(i,j)$ if $D(i,j) = D(i-1,j)+1$, from $D(i,j-1)$ to $D(i,j)$ if $D(i,j) = D(i,j-1)+1$, and from $D(i-1,j-1)$ to $D(i,j)$ if $D(i,j) = D(i-1,j-1)$ when $p_i = t_j$ or if $D(i,j) = D(i-1,j-1)+1$ when $p_i \ne t_j$. The costs of the arcs are 1, 1, 0, and 1, respectively. The minimizing arcs show the actual dependencies between the values in table D. A *minimizing path* is any path that consists of minimizing arcs and leads from an entry $D(0,j)$ on the first row of D to an entry $D(m,h)$ on the last row of D. Note that $D(m,h)$ equals the sum of the costs of the arcs on the path. A minimizing path is *successful* if it leads to an entry $D(m,h) \le k$.

A *diagonal* h of D for $h = -m, \ldots, n$, consists of all $D(i,j)$ such that $j - i = h$. As any vertical or horizontal minimizing arc adds 1 to the value of the entry, the next lemma easily follows.

LEMMA 3. *The entries on a successful minimizing path are contained in* $\le k+1$ *successive diagonals of D.*

Our marking method is based on the following lemma. For each $i = 1, \ldots, m$, let the *k environment* of the pattern symbol p_i be the string $C_i = p_{i-k} \ldots p_{i+k}$, where $p_j = \epsilon$ for $j < 1$ and $j > m$.

LEMMA 4. *Let a successful minimizing path go through some entry on a diagonal h of D. Then for at most k indexes,* $i, 1 \le i \le m$, *character* t_{h+i} *does not occur in k environment* C_i.

Proof. Column $j, h+1 \le j \le h+m$, of D is called *bad* if t_j does not appear in C_{j-h}. The lemma claims that the number of the bad columns is $\le k$. Let M be the path in the lemma. Let R be the set of indexes, $j, h+1 \leq j \leq h+m$, such that path M contains at least one entry $D(i,j)$ on column j of D. If M starts or ends outside diagonal h, then the size of R can be $< m$. Then, however, M must have at least one vertical arc for each index j missing in R because M crosses diagonal h. Therefore, $\mathit{vert}(M) \geq m - \mathit{size}(R)$, where $\mathit{vert}(M)$ is the number of vertical arcs of M.

By Lemma 3, M must be contained in diagonals $h-k, h-k+1, \ldots, h+k$ of D. Hence for each j in R, path M must enter some entry on column j restricted to diagonals $h-k, \ldots, h+k$, that is, some entry $D(i-k,j), \ldots, D(i+k,j)$. Then if j is bad, the first arc in M that enters column j must add 1 to the total cost of M. Because such an arc enters a new column, it must be either a diagonal or a horizontal arc; note that with no restriction on generality we may assume that the very first arc of M is not a vertical one. Hence the number of bad columns in R is $\leq \mathit{cost}(M) - \mathit{vert}(M)$, where $\mathit{cost}(M)$ is the value of the final entry of M.

Moreover, there can be $m - \mathit{size}(R)$ additional bad columns as every column outside R can be bad. The total number of the bad columns is, therefore, at most $m - \mathit{size}(R) + \mathit{cost}(M) - \mathit{vert}(M) \leq \mathit{cost}(M) \leq k$. □

Lemma 4 suggests the following marking method. For diagonal h, check for $i = m, m-1, \ldots, k+1$ if t_{h+i} is in C_i until $k+1$ bad columns are found. Note that to get minimum shift $k+1$ (see Fig. 2) we stop already at $i = k+1$ instead of at $i = 1$. If the number of bad columns is $\leq k$, then mark diagonals $h-k, \ldots, h+k$, that is, mark entries $D(0, h-k), \ldots, D(0, h+k)$.

For finding the bad columns fast we need a precomputed table $\mathit{Bad}(i,a), 1 \leq i \leq m, a \in \Sigma$, such that

$$\mathit{Bad}(i,a) = \textbf{true} \quad \text{if and only if } a \text{ does not appear in } k \text{ environment } C_i.$$

Clearly the table can be computed by a simple scanning of P in time $O((c+k)m)$.

After marking we have to determine the length of shift; that is, what is the next diagonal after h around which the marking should eventually be done? The marking of heuristic ensures that all successful minimizing paths that are properly before diagonal $h+k+1$ are already marked. Hence we can safely make at least a shift of $k+1$ to diagonal $h+k+1$.

This can be combined with the shift heuristics of Algorithm 4 of §2 based on table d_k. So we determine the first diagonal after h, say, $h+d$, where at least one of the characters $t_{h+m}, t_{h+m-1}, \ldots, t_{h+m-k}$ matches with the corresponding character of P. This is correct because then there can be a successful minimizing path that goes through diagonal $h+d$. The value of d is evaluated as in Algorithm 4, using exactly the same precomputed table d_k. Note that unlike in the case of Algorithm 4, the maximum allowed value of d is now m, not $m-k$, as the marking starts from diagonal $h-k$, not from h. Finally, the maximum of $k+1$ and d is the length of the shift.

In practice, the marking and the computation of the shift can be merged if we start searching for the bad columns from the end of the pattern.

Figure 2 illustrates marking and shifting. For $r = h+m, h+m-1, \ldots, h+k+1$ we check whether or not t_r appears among the pattern symbols corresponding to the shaded block 1 (the k environment). If $k+1$ symbols t_r that do not appear are found, entries $D(0, h-k), \ldots, D(0, h+k)$ are marked. Simultaneously we check what is the next diagonal after h containing a match between P and $t_{h+m-k}, \ldots, t_{h+m}$ (shaded block 2). The next shift is to this diagonal but at least to diagonal $h+k+1$.

We get the following algorithm for the scanning phase.

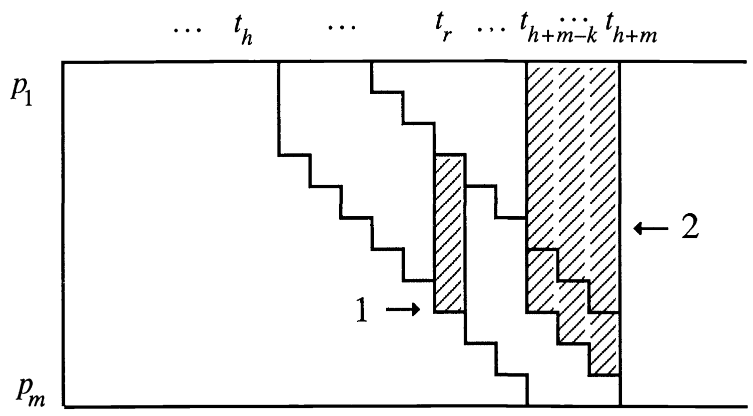

$\ldots\ t_h\ \ldots\ t_r\ \ldots\ t_{h+m-k}\ \ldots\ t_{h+m}$
p_1
p_m
$\leftarrow 2$
$1 \rightarrow$

FIG. 2. *Mark and shift* ($k = 2$).

ALGORITHM 5. The scanning phase for the k differences problem.

```
1.  compute table Bad and, by Algorithm 3, table $d_k$ from $P$;
2.  $j := m$;
3.  while $j \leq n + k$ do begin
4.      $r := j; i := m$;
5.      $bad := 0$;                    {bad counts the bad indexes}
6.      $d := m$;                      {initial value of shift}
7.      while $i > k$ and $bad \leq k$ do begin
8.          if $i \geq m - k$ then $d :=\min(d, d_k[i, t_r])$;
9.          if $Bad(i, t_r)$ then $bad := bad + 1$;
10.         $i := i - 1; r := r - 1$ end;
11.     if $bad \leq k$ then
12.         mark entries $D(0, j - m - k), \ldots, D(0, j - m + k)$;
13.     $j := j + \max(k + 1, d)$ end
```

The loop in steps 7–9 can be slightly optimized by splitting it into two parts such that the first one handles $k + 1$ text characters and computes the length of shift, and the latter goes on counting bad indexes (a similar optimization also applies to Algorithm 4).

Algorithm 5 has the drawback that it marks all diagonals when $k > (m - 1)/2$, and hence it is useful only for $k \leq (m - 1)/2$. Its marking accuracy can be improved for larger k at cost of losing the minimum shift of $k + 1$. This is achieved by replacing line 7 by

```
7'.         while $i > 0$ and $bad \leq k$ do begin
```

and lines 11–13 by

```
11'.    if $bad \leq k$ then begin
12'.        mark entries $D(0, j - m - k), \ldots, D(0, j - m + k)$;
13'.    $j := j + \max(k + 1, d)$ end
14'.    else $j := j + \max(\min(k + 1, i + 1), d)$ end
```

3.3. Analysis. The preprocessing of P requires $O((k+c)m)$ for computing table *Bad* and $O(m+kc)$ for computing table d_k. As $k<m$, the total time is $O((k+c)m)$. The working space is $O(cm)$.

The marking and shifting by Algorithm 5 takes time $O(mn/k)$ in the worst case. The analysis of the average case is similar to the analysis of Algorithm 4 in §2. Let $B_{loc}(P)$ be a random variable denoting, for some fixed c and k, the number of the columns examined (step 9 of Algorithm 5) until $k+1$ bad columns are found and the next shift will be taken.

Obviously, $B_{loc}(P)$ corresponds to $C_{loc}(P)$ of Lemma 1. For the expected value $\bar{B}_{loc}(P)$ we show the following rough bound.

LEMMA 5. *Let* $2k+1<c$. *Then*

$$\bar{B}_{loc}(P) \le \left(\frac{c}{c-2k-1}+1\right)(k+1).$$

Proof. The expected value of $B_{loc}(P)-(k+1)$ can be bounded from above by the expected value of the negative binomial distribution with parameters $(k+1,q)$, where q is a lower bound for the probability that a column is bad. Recall that column j is called bad if text symbol t_j does not occur in the corresponding k environment. As the k environment is a substring of P of length at most $2k+1$, it can have at most $2k+1$ different symbols. Therefore, the probability that a random t_j does not belong to the symbols of a k environment is at least $(c-(2k+1))/c$. Hence we can choose $q=(c-(2k+1))/c$.

The negative binomial distribution would then give for $\bar{B}_{loc}(P)-(k+1)$ an upper bound $(2k+1)(k+1)/(c-(2k+1))$. However, the shift heuristic implies that after a shift of length $<m$ we know that at least one and at most $k+1$ columns will not be bad. Hence to bound $\bar{B}_{loc}(P)-(k+1)$ properly, we have to add $k+1$ to the above bound, which gives

$$\bar{B}_{loc}(P)-(k+1) \le \frac{(2k+1)}{c-(2k+1)}(k+1)+k+1,$$

and the lemma follows. □

Let $S'(P)$ be a random variable denoting the length of the shift in Algorithm 5 for pattern P and for some fixed k and c. When scanning a random T, the special pattern P_0 again gives the shortest expected shift, that is, $\bar{S}'(P_0) \le \bar{S}'(P)$ for all P of length m. Lemma 6 gives a bound for $\bar{S}'(P_0)$.

LEMMA 6. $\bar{S}'(P_0) \ge \frac{1}{2}\min(c/(k+1), m)$.

Proof. Let $t=\min(c-1, m-1)$. Then the possible lengths of a shift are $1,2,\ldots,t+1$; note that a shift actually is always $\ge k+1$ according to our heuristic, but the heuristic can be ignored here, as our goal is to prove a lower bound. Therefore,

$$\bar{S}'(P_0)=\sum_{i=0}^{t} Pr(S'(P_0)>i).$$

If $0 \le i \le m-k-1$, then

$$Pr(S'(P_0)>i)=\left(\frac{c-i}{c}\right)^{k+1}$$

because for each of the $k+1$ text symbols that are compared with the pattern to determine the shift (step 8 of Algorithm 5), there are i characters not allowed to occur as the text symbols. This is exactly as in the proof of Lemma 2. A slight difference arises when $m - k \leq i \leq m - 1$. Then

$$Pr(S'(P_0) > i) = \left(\frac{c-1}{c}\right)^{m-i} \cdot \frac{c-i+1}{c} \cdot \frac{c-i+2}{c} \cdot \ldots \cdot \frac{c-m+k+1}{c}$$

because now the number of forbidden characters is i for the $m - i$ last text symbols and $i - 1, i - 2, \ldots, i - (m - k - 1)$ for the remaining $k + 1 - (m - i)$ text symbols, listed from right to left. But also in this case,

$$Pr(S'(P_0) > i) \geq \left(\frac{c-i}{c}\right)^{k+1}.$$

Hence

$$\bar{S}'(P_0) \geq \sum_{i=0}^{t} \left(1 - \frac{i}{c}\right)^{k+1}.$$

The rest of the proof is divided into two cases which are so similar to the cases in the proof of Lemma 2 that we do not repeat the details. If $m < c/(k+1)$, then $\bar{S}'(P_0) \geq \frac{1}{2}m$. If $m \geq c/(k+1)$, then $\bar{S}'(P_0) \geq \frac{1}{2}\lceil c/(k+1)\rceil$. □

As the length of a shift is always $\geq k + 1$, we get, from Lemma 6,

$$\begin{aligned} \bar{S}'(P) &\geq \bar{S}'(P_0) \\ &\geq \max\left(k+1, \min\left(\frac{c}{2(k+1)}, \frac{m}{2}\right)\right) \\ &= \min\left(\max\left(k+1, \frac{c}{2(k+1)}\right), \max\left(k+1, \frac{m}{2}\right)\right) \\ &\geq \frac{1}{2}\min\left(k+1+\frac{c}{2(k+1)}, \frac{m}{2}\right). \end{aligned}$$

The number of text positions at which a right-to-left scanning of P is performed between two shifts is again

$$O\left(\frac{n-m}{\bar{S}'(P)}\right) = O\left(\frac{n-m}{\bar{S}'(P_0)}\right).$$

This can be shown as in the analysis of Algorithm 4. Note that for Algorithm 5 we need not assume explicitly that the lengths of different shifts are independent. They are independent as the length of the minimum shift is $k + 1$.

Hence the expected scanning time of Algorithm 5 for pattern P is

$$O\left(\bar{B}_{loc}(P) \cdot \frac{n-m}{\bar{S}'(P)}\right).$$

When we apply here the upper bound for $\bar{B}_{loc}(P)$ from Lemma 5 and the above lower bound for $\bar{S}'(P)$, and simplify, we obtain our final result.

THEOREM 2. *Let* $2k + 1 < c$. *Then the expected scanning time of Algorithm* 5 *is*

$$O\left(\left(\frac{c}{c-2k}\right) \cdot kn \cdot \left(\frac{k}{c+2k^2} + \frac{1}{m}\right)\right)$$

The preprocessing time is $O((k+c)m)$, and the working space $O(cm)$.

The checking of the marked diagonals can be done after Algorithm 5 or in cascade with it in which case a buffer of length $2m$ is enough for saving the relevant part of text T. The latter approach is presented in Algorithm 6, which contains a modification of Algorithm 5 as its subroutine, function *NPO*.

ALGORITHM 6. The total algorithm for the k differences problem.

```
 1.        function NPO; begin                 {the next possible occurrence}
 2.        while $j \le n + k$ do begin
 3.            $r := j; i := m; bad := 0; d := m;$
 4.            while $i > k$ and $bad \le k$ do begin
 5.                if $i \ge m - k$ then $d := \min(d, d_k[i, t_r]);$
 6.                if $Bad(i, t_r)$ then $bad := bad + 1;$
 7.                $i := i - 1; r := r - 1$ end;
 8.            if $bad \le k$ then goto out;
 9.            $j := j + \max(k + 1, d)$ end;
10.  out: if $j \le n + k$ then begin
11.            $NPO := j - m - k;$
12.            $j := j + \max(k + 1, d)$ end
13.        else $NPO := n + 1$ end;

14.       compute tables $Bad$ and $d_k$;
15.       $j := m;$
16.       for $i := 0$ to $m$ do $H_0[i] := i;$
17.       $H := H_0;$
18.       $top := \min(k + 1, m);$             {$top - 1$ is the last row with the value $\le k$}
19.       $col := NPO;$
20.       $lastcol := col + m + 2k - 1;$
21.       while $col \le n$ do
22.           for $r := col$ to $lastcol$ do begin
23.               $c := 0;$
24.               for $i := 1$ to $top$ do begin
25.                   if $p_i = t_r$ then $d := c;$
26.                   else $d := \min(H[i-1], H[i], c) + 1;$
27.                   $c := H[i]; H[i] := d$ end;
28.               while $H(top) > k$ do $top := top - 1;$
29.               if $top = m$ then report match at $j$;
30.               else $top := top + 1$ end;
31.           $next := NPO;$
32.           if $next > lastcol + 1$ then begin
33.               $H := H_0;$
34.               $top := \min(k + 1, m);$
```

```
35.             col := next end
36.         else col := lastcol +1;
37.         lastcol := next +m + 2k − 1 end
```

The checking phase of Algorithm 6 evaluates a part of D by dynamic programming (see §3.1). Because entries on every diagonal are monotonically increasing [Ukk85a], the computation along a marked diagonal can be stopped, when the threshold value of $k+1$ is reached, because the rest of the entries on that diagonal will be greater than k. Algorithm 6 implements this idea in a slightly streamlined way. Instead of restricting the evaluation of D exactly on the marked diagonals, (which could be done, of course, but leads to more complicated code), we evaluate each *column* of D that intersects some marked diagonal. Each such column is evaluated from its first entry to the last one that could be $\leq k$. This can be easily decided using the diagonalwise monotonicity of D [Ukk85b]. The evaluation of each separate block of columns can start from a column identical to the first column of D (H_0 in Algorithm 6; H stores the previous as well as the current column under evaluation). For random strings, this method spends expected time of $O(k)$ on each column (this conjecture of [Ukk85b] has recently been proved in [ChL92]). Hence the total expected time of the checking phase remains $O(kn)$.

Asymptotically, steps 22–37 of Algorithm 6 are executed very seldom. Hence except for small patterns, small alphabets and large k's, the expected time for the checking phase tends to be small, in which case the time bound of Theorem 2 is valid for our entire algorithm.

3.4. Variations. Each marking operation before the next shift takes time $O(m)$ in the worst case. At the cost of decreased accuracy of marking we can reduce this by limiting the number of the columns whose badness is examined. The time reduces to $O(k)$ when we examine only at most ak columns for some constant $a > 1$. If there are not more than k bad columns among them, then the diagonals are marked. This variation appealingly has the feature that the total time of marking and shifting reduces to $O(n)$ in the worst case. Of course, the gain may be lost in the checking phase, as more diagonals will be marked.

On the other hand, the accuracy of the marking heuristic, which quite often conservatively marks too many diagonals in its present form, can be improved by a more careful analysis of whether or not a column is bad. Such an analysis can be based, at the cost of longer preprocessing, on the observation that two matches on successive columns of D can occur in the same minimizing path only if they are on the same diagonal.

In Algorithm 6, the width of the band of columns inspected is $m+2k$. The algorithm works better for small alphabets and short patterns, if a wider width is used, because that will reduce reinspection of text positions during the scanning phase. If the width is at least $2m+k$, then we can in the case of a potential match make a shift of $m+1$, which guarantees that no text position is reinspected in that situation.

4. Experiments and conclusions. We have tested extensively our algorithms and compared them with other methods. We will present results of a comparison with the $O(kn)$ expected time dynamic programming method [Ukk85b] which we found to be the best in practice among the old algorithms we have tested [JTU90].

Table 1 shows total execution times of Algorithm 4 and 6 and the corresponding dynamic programming algorithms $DP1$ (the k mismatches problem) and $DP2$ (the k differences problem). Preprocessing, scanning, and checking times are specified for Algorithm 6, as well as preprocessing times for Algorithm 4. In our tests, we used random

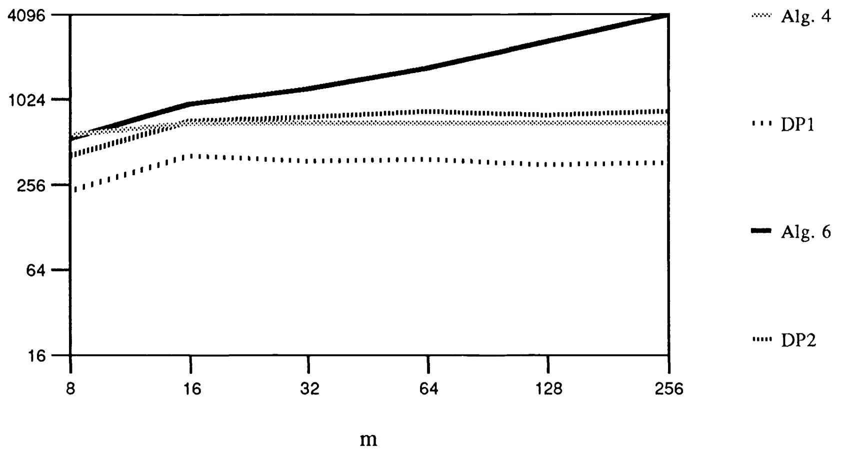


FIG. 3. *Total times for $k = 4$ and $c = 2$.*

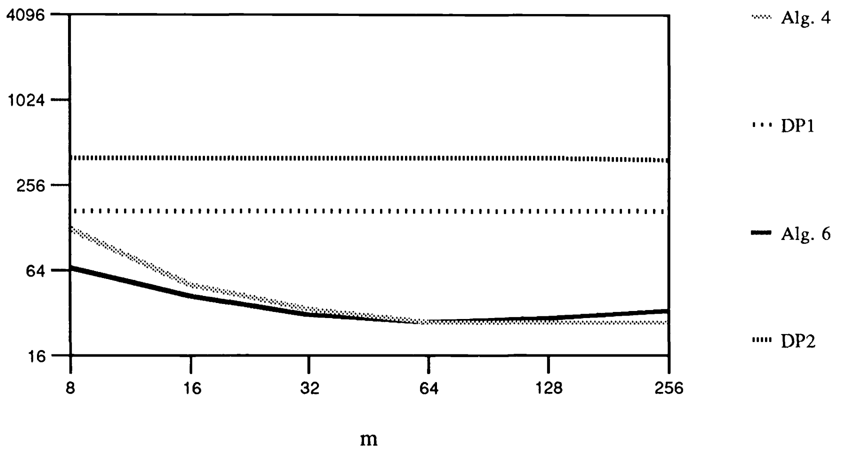


FIG. 4. *Total times for $k = 4$ and $c = 90$.*

patterns of varying lengths and random texts of length 100, 000 characters over alphabets of different sizes. The tests were run on a VAX 8800 under VMS. In order to decrease random variation, the figures of Table 1 are averages of ten runs. Still more repetitions should be necessary to eliminate variation, as can be seen in the duplicate entries of Table 1 corresponding to different test series with the same parameters.

Figures 3–6 have been drawn form the data of Table 1. Figures 3 and 4 show the total execution times when $k = 4$ and m varies for alphabet sizes $c = 2$ and 90. Figures 5 and 6 show the corresponding times when $m = 8$ and k varies for alphabet sizes $c = 4$ and 30.

Our algorithms, as all algorithms of Boyer–Moore type, work very well for large alphabets, and the execution time decreases when the length of the pattern grows. An increment of the error limit k slows down our algorithms more than the dynamic programming algorithms. Observe also that the Boyer–Moore approach is relatively better

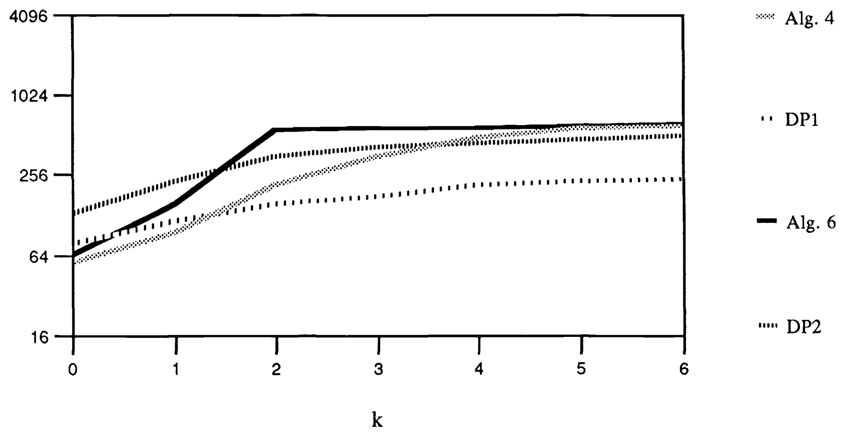


FIG. 5. *Total times for $m = 8$ and $c = 4$.*

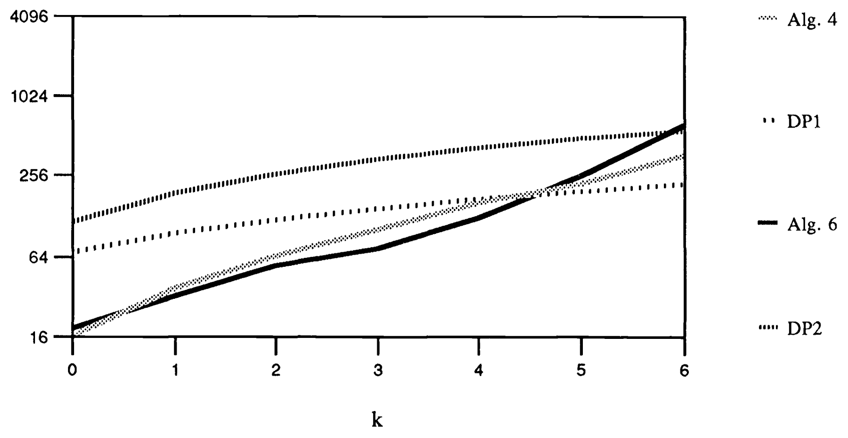


FIG. 6. *Total times for $m = 8$ and $c = 30$.*

in solving the k differences problem than in solving the k mismatches problem.

Our methods turned out to be faster than the previous methods; when the pattern is long enough ($m > 5$), the error limit k is relatively small and the alphabet is not very small ($c > 5$). Results of the practical experiments are consistent with our theoretical analysis. Devising a more accurate and complete theoretical analysis of the algorithms is left as a subject for further study.

TABLE 1

Execution times (in units of 10 *milliseconds) of the algorithms* ($n = 100,000$). *Prepr., Scan, and Check denote the preprocessing, scanning, and checking times, respectively.*

c	m	k	ALG. 4		DP1	ALG. 6				DP2
			Prepr.	Total		Prepr.	Scan	Check	Total	
2	8	4	0	574	227	0	129	406	535	403
2	16	4	0	681	403	0	240	705	945	700
2	32	4	0	681	371	0	451	759	1210	756
2	64	4	0	679	385	0	881	813	1694	817
2	128	4	0	688	349	0	1762	792	2554	786
2	256	4	0	691	361	0	3172	827	3999	824
4	8	4	0	451	213	0	129	469	598	465
4	16	4	0	453	224	0	235	557	792	553
4	32	4	0	447	222	0	427	731	1158	550
4	64	4	0	464	227	0	700	538	1238	563
4	128	4	0	459	226	0	849	216	1065	556
4	256	4	0	436	226	0	724	2	726	553
30	8	4	0	151	174	0	84	84	168	406
30	16	4	0	88	170	0	75	0	75	410
30	32	4	0	78	167	0	72	0	72	406
30	64	4	0	75	167	0	70	0	70	403
30	128	4	0	79	167	1	73	0	74	404
30	256	4	0	79	167	1	73	0	74	403
90	8	4	0	126	166	0	63	2	65	389
90	16	4	0	50	164	0	40	0	40	389
90	32	4	0	33	166	0	30	0	30	390
90	64	4	0	27	165	1	25	0	26	389
90	128	4	0	27	164	2	26	0	28	388
90	256	4	1	27	164	4	27	0	31	387
2	8	0	0	89	102	0	106	9	115	164
2	8	1	0	234	155	0	260	246	506	278
2	8	2	0	371	193	0	208	361	569	353
2	8	3	0	488	220	0	158	405	563	399
2	8	4	0	570	223	0	127	405	533	404
2	8	5	0	628	223	0	109	407	516	407
2	8	6	0	677	221	0	93	405	498	401
4	8	0	0	56	78	0	63	0	63	129
4	8	1	0	95	113	0	112	43	155	229
4	8	2	0	211	153	0	199	358	557	353
4	8	3	0	344	175	0	158	415	573	408
4	8	4	0	480	211	0	128	447	575	445
4	8	5	0	575	225	0	108	481	589	477
4	8	6	0	582	232	0	98	505	603	503
30	8	0	0	16	68	0	18	0	18	115
30	8	1	0	36	93	0	32	0	32	187
30	8	2	0	63	120	0	54	0	54	263
30	8	3	0	102	144	0	68	5	73	336
30	8	4	0	157	169	0	79	44	123	412
30	8	5	0	222	194	0	84	170	254	484
30	8	6	0	364	219	0	90	519	609	548
90	8	0	0	15	67	0	16	0	16	114
90	8	1	0	32	93	0	29	0	29	189
90	8	2	0	55	119	0	40	0	40	258
90	8	3	0	87	144	0	53	0	53	332
90	8	4	0	132	170	0	63	1	64	408
90	8	5	0	208	198	0	78	37	115	484
90	8	6	0	344	221	0	84	207	291	554

Acknowledgment. Petteri Jokinen performed the experiments, which is gratefully acknowledged.

REFERENCES

[Bae89a] R. BAEZA-YATES, *Efficient Text Searching*, Ph.D. thesis, Report CS-89-17, Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[Bae89b] ———, *String searching algorithms revisited*, in Proceedings of the Workshop on Algorithms and Data Structures, F. Dehne et al., ed., Lecture Notes in Computer Science 382, Springer-Verlag, Berlin, 1989, pp. 75–96.

[BoM77] R. BOYER AND S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762–772.

[ChL92] W. CHANG AND J. LAMPE, *Theoretical and empirical comparisons of approximate string matching algorithms*, in Proceedings of the Third Symposium on Combinatorial Pattern Matching, Tucson, AZ, 1992, pp. 172–181.

[ChL90] W. CHANG AND E. LAWLER, *Approximate string matching in sublinear expected time*, in Proceedings of the 31st IEEE Annual Symposium on Foundations of Computer Science, IEEE Press, New York, 1990, pp. 116–124.

[Fel65] W. FELLER, *An Introduction to Probability Theory and Its Applications*, Vol. I, John Wiley, New York, 1965.

[Fel66] ———, *An Introduction to Probability Theory and Its Applications*, Vol. II, John Wiley, New York, 1966.

[GaG86] Z. GALIL AND R. GIANCARLO, *Improved string matching with* k *mismatches*, SIGACT News, 17 (1986), pp. 52–54.

[GaG88] ———, *Data structures and algorithms for approximate string matching*, J. Complexity, 4 (1988), pp. 33–72.

[GaP89] Z. GALIL AND K. PARK, *An improved algorithm for approximate string matching*, in Proceedings of the 16th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 372, Springer-Verlag, Berlin, 1989, pp. 394–404.

[GrL89] R. GROSSI AND F. LUCCIO, *Simple and efficient string matching with* k *mismatches*, Inform. Process. Lett., 33 (1989), pp. 113–120.

[Hor80] N. HORSPOOL, *Practical fast searching in strings*, Software Practice & Experience, 10 (1980), pp. 501–506.

[JTU90] P. JOKINEN, J. TARHIO, AND E. UKKONEN, *A comparison of approximate string matching algorithms*, Report A-1991-7, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1991.

[Kos88] S. R. KOSARAJU, *Efficient string matching*, extended abstract, Department of Computer Science, Johns Hopkins University, Baltimore, MD, 1988.

[KMP77] D. KNUTH, J. MORRIS, AND V. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.

[LaV88] G. LANDAU AND U. VISHKIN, *Fast string matching with* k *differences*, J. Comput. System Sci., 37 (1988), pp. 63–78.

[LaV89] ———, *Fast parallel and serial approximate string matching*, J. Algorithms, 10 (1989), pp. 157–169.

[Sel80] P. SELLERS, *The theory and computation of evolutionary distances: Pattern recognition*, J. Algorithms, 1 (1980), pp. 359–372.

[Ukk85a] E. UKKONEN, *Algorithms for approximate string matching*, Inform. Control, 64 (1985), pp. 100–118.

[Ukk85b] ———, *Finding approximate patterns in strings*, J. Algorithms, 6 (1985), pp. 132–137.

[UkW90] E. UKKONEN AND D. WOOD, *Fast approximate string matching with suffix automata*, Report A-1990-4, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1990.

[WaF75] R. WAGNER AND M. FISCHER, *The string-to-string correction problem*, J. Assoc. Comput. Mach., 21 (1975), pp. 168–173.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 261-271, April 1993

TWO PROBABILISTIC RESULTS ON MERGING*

WENCESLAS FERNANDEZ DE LA VEGA[†], SAMPATH KANNAN[‡], AND MIKLOS SANTHA[†]

Abstract. This paper contains two probabilistic results about merging two sorted lists of sizes n and m with $m < n$. This paper designs a probabilistic algorithm, which in the worst case is significantly faster than any deterministic one in the range $1.618 < n/m \leq 3$. This paper extends it into a simple general algorithm that performs well for any ratio n/m. In particular, for $n/m > 1.618$ it is significantly faster than binary merge. This paper also proves an average case lower bound for a widely studied class of merging algorithms, when $1 < n/m < \sqrt{2}+1$.

Key words. merging, randomized algorithm, information theory, lower bound

AMS(MOS) subject classifications. 68Q20, 68Q25

1. Introduction. Merging is one of the basic problems in theoretical computer science. Given two sorted lists $A = \{a_1 < \ldots < a_m\}$ and $B = \{b_1 < \ldots < b_n\}$, the task consists of sorting the union of the two lists. We assume that the $m+n$ elements are distinct and $m \leq n$. The merging is performed by pairwise comparisons between items in A and items in B. The measure of complexity of a merging algorithm is the number of comparisons made by the algorithm, and the complexity of the merging problem is the complexity of the best merging algorithm. As usual, we can speak of worst case and average case complexity.

The worst case complexity of the merging problem is quite well studied. Let $C(m,n)$ denote this complexity with input lists of sizes m and n. When $m = 1$, merging degenerates into binary search whose complexity is $\lceil \log_2(n+1) \rceil$. The case $m = 2$ was completely solved by Graham [7] and Hwang and Lin [5]; their result is

$$C(2,n) = \lceil \log_2 7(n+1)/12 \rceil + \lceil \log_2 14(n+1)/17 \rceil.$$

The exact complexity is also known when the sizes of the two lists are not too far apart. For $m \leq n \leq \lfloor 3m/2 \rfloor$ we have

$$C(m,n) = m+n-1,$$

and the optimal algorithm is the well-known tape-merge (two-way merge). This result was obtained by Stockmeyer and Yao [10] and by Christen [2], after it was observed by several people for $m \leq n \leq m+4$. The best known algorithm—binary merge—which gives satisfactory results for all values of m and n is due to Hwang and Lin [6]. Let N denote the set of nonnegative integers. Set $n/m = 2^t x$, where $t \in N$ and $1 < x \leq 2$ are uniquely determined, and let $BM(m,n)$ denote the complexity of binary merge. Then

$$BM(m,n) = \lfloor (t+x+1)m \rfloor - 1,$$

and Hwang and Lin have also shown that

$$BM(m,n) \leq \lceil L(m,n) \rceil + m,$$

*Received by the editors June 4, 1990; accepted for publication (in revised form) December 15, 1991.

[†]Centre National de la Recherche Scientifique, Unité de Recherche Associée 410, Université Paris-Sud, 91405 Orsay, France.

[‡]Computer Science Department, University of Arizona, Tucson, Arizona 85721. The work of this author was supported by National Science Foundation grant CCR91-08969.

where $L(m, n) = \log_2 \binom{m+n}{m}$ is the lower bound from information theory. This means that if the ratio n/m goes to infinity, then the relative extra work done by binary merge is $o(C(m, n))$. As the relative extra work might be significant for ratios n/m of constant order, it is important to look for improvements in this case.

Several algorithms were proposed, which in some range for n/m perform better than binary merge. Let us say that merging algorithm A_1 with running time $T_1(m, n)$ is *significantly* faster for some fixed ratio n/m than merging algorithm A_2 with running time $T_2(m, n)$, if $T_2(m, n) - T_1(m, n) = \Omega(m)$. The merging algorithm of Hwang and Deutsch [4] is better than binary merge for small values of m, but not significantly faster. The first significant improvement over binary merge was proposed by Manacher [8]; he improved it for $n \geq 8m$ by $31m/336$ comparisons. It was further improved by Christen [1], who designed an ingenious but quite involved merging algorithm. It is better than binary merge if $n > 3m$, it uses at least $m/4$ fewer comparisons if $n \geq 4m$, and asymptotically it uses at least $m/3 - o(m)$ fewer comparisons when n/m goes to infinity. On the other hand, this algorithm is worse than binary merge when $n < 3m$.

In the first part of this paper we propose a simple probabilistic algorithm for merging. The algorithm will at some points flip a (biased) coin, and its next step will depend on the result of the coin toss. The algorithm is quite simple, and works well for all values of m and n. It is significantly faster than binary merge for $n/m > (\sqrt{5} + 1)/2 \approx 1.618$. To the best of our knowledge, binary merge is the best algorithm known for $n/m \leq 3$, thus our probabilistic algorithm is significantly faster than any deterministic one in this range.

In the second part of the paper we prove a nontrivial average case lower bound for a widely studied class of merging algorithms called *insertive* algorithms, defined as follows: During the run of a merging algorithm, we will say that an element a of a list is *active* if a has been compared to at least one element of the other list, but a's exact position in the other list has not been determined. We will call a merging algorithm *insertive* if there is no more than one active element in the smaller list at any time. Apart from this constraint the algorithm can be arbitrary. Many merging algorithms such as repeated binary search, tape merge, and binary merge are insertive. We can establish the lower bound for ratios n/m in the interval $(1, \sqrt{2}+1)$ which are bounded away by a constant from the endpoints of the interval. More precisely, we will prove that if $(1 + \eta) \leq n/m \leq (\sqrt{2} + 1 - \eta)$ for some constant $\eta > 0$, then every insertive merging algorithm makes on the average at least a constant factor more comparisons than the information theoretic lower bound. We were unable to extend the lower bound for constant ratios $n/m \geq \sqrt{2} + 1$. When $n/m \to \infty$, then the result of Hwang and Lin also implies that the information theoretic lower bound and the actual complexity are asymptotically equal.

The rest of the paper is organized as follows: In §2.1 we describe our algorithm. The analysis of its complexity will be done in §2.2 and we also prove there that the particular bias of the coin in the algorithm was optimally chosen. In §3 we prove the average case lower bound result for insertive algorithms. Finally, in §4 we mention some open problems. Throughout the paper the logarithm function in base 2 will be denoted by log, and in the natural base by ln.

2. The probabilistic algorithm.

2.1. Description of the algorithm. For this section let $s = (\sqrt{5} - 1)/2 \approx 0.618$, and $r = (\sqrt{2} - 1 + \sqrt{2}s)^2 \approx 1.659$. These numbers play a considerable role in the algorithm and in its analysis. The heart of the algorithm MERGE is the probabilistic procedure PROBMERGE, which merges two already sorted lists, where the longer list contains more than $(1+s)$-times, but at most $2r$-times as many elements as the shorter one. The intuition underlying PROBMERGE is described below. We know that if $n \le \lfloor 3m/2 \rfloor$, then the tape merge algorithm is best possible. Given two sorted lists A and B as before, the tape merge algorithm can be thought of as inserting the a's in order in list B. At each stage the next a to be inserted is compared with the first "eligible" element in B. If, however, B is a list of length $2m$, then it might be better to compare the next a to be inserted with the second eligible b (since, on average, there are two b's between every two consecutive a's). However, the deterministic algorithm that tries to do this turns out to be no better than tape merge. Its worst case complexity is also $3m-1$. A natural idea is to try to compare the next a to be inserted with either the first or the second eligible element in B, the decision being made probabilistically. In fact this is our algorithm.

We will use sentinel elements in the algorithm. They make the description simpler for the price of a little loss of efficiency.

Procedure PROBMERGE
Input: $A = \{a_1 < \ldots < a_m\}$ and $B = \{b_1 < \ldots < b_n\}$, where $1+s < n/m \le 2r$, and sentinel elements $b_0 = -\infty$, $b_{n+1} = b_{n+2} = \infty$.
Output: The merged list.

```
• $p := \begin{cases} s & \text{if } 1+s < n/m \le 2+s \\ \sqrt{n/m} - 1 & \text{if } 2+s < n/m \le 2r \end{cases}$   $\left\{ \begin{array}{l} \textit{the choice of the} \\ \textit{probability value} \end{array} \right\}$
• $i := 1; j := 1$ $\{i$ indexes $A$, $j$ indexes $B\}$
• **while** $i \le m$ **do**
    – with probability $1-p$ compare $a_i$ with $b_j$
        * **if** $a_i < b_j$ **then** $i := i+1$ $\{b_{j-1} < a_i < b_j\}$
        * **else** $j := j+1$
    – with probability $p$ compare $a_i$ with $b_{j+1}$
        * **if** $a_i < b_{j+1}$ **then**
            · compare $a_i$ with $b_j$
            · **if** $a_i < b_j$ **then** $i := i+1$ $\{b_{j-1} < a_i < b_j\}$
            · **else** $i := i+1; j := j+1$ $\{b_j < a_i < b_{j+1}\}$
        * **else** $j := j+2$
• **end**
```

The general algorithm MERGE uses PROBMERGE as a subroutine. Given lists A and B of size m and $n > (1+s)m$, respectively, the algorithm calls PROBMERGE directly if $n \le rm$. Otherwise it picks a uniformly spaced sublist C of B. The spacing between the elements of C is 2^t, where t is the unique integer such that cardinality of C is between rm and $2rm$. The algorithm first merges A with C probabilistically. This determines for each a in A a list of $2^t - 1$ b's into which a should be inserted. The insertion is done deterministically by binary search and takes t steps. Let us observe that MERGE actually coincides with the procedure PROBMERGE while $n \le 2rm$.

Algorithm MERGE
Input: $A = \{a_1 < \ldots < a_m\}$ and $B = \{b_1 < \ldots < b_n\}$, where $(1+s) < n/m$. For $r < n/m$ set $m = 2^t xm$, $t \in N$, $r < x \le 2r$.
Output: The merged list.

1. If $n/m \le r$ then PROBMERGE(A, B).
2. Let $C = \{c_1 < \ldots < c_{\lceil xm \rceil}\}$ be the sublist of B, where $c_k = b_{(k-1)2^t+1}$ for $k = 1, \ldots, \lceil xm \rceil$, and define sentinel elements $c_0 = -\infty$, $c_{\lceil xm \rceil + 1} = \infty$.
3. PROBMERGE(A, C).

4. For $i = 1, \ldots, m$ let $0 \leq j_i \leq \lceil xm \rceil$ be such that $c_{j_i} < a_i < c_{j_i+1}$.
5. For $i = 1, \ldots, m$ insert a_i by binary search into the list $\{b_{(j_i-1)2^t+2} < \cdots < b_{j_i 2^t}\}$.

2.2. Analysis of the algorithm. We would like to analyze the expected number of comparisons made by the algorithm PROBMERGE. Let $T(m, n)$ be this number when the input of the procedure is two lists of size m and n, respectively, $1 + s < n/m \leq 2r$, and let p be the probability value defined inside the procedure.

THEOREM 2.1. *We have*

$$T(m,n) \leq \begin{cases} sn + (1+s)m & \text{if } 1 + s < n/m \leq 2 + s, \\ 2\sqrt{nm} & \text{if } 2 + s < n/m \leq 2r. \end{cases}$$

Proof. The outcome of the procedure uniquely determines m nonnegative integers $k_1, \ldots, k_m$, where k_i is the number of b's between a_{i-1} and a_i. As the list B contains n elements, $\sum_{i=1}^m k_i \leq n$. The procedure puts the a's into B one by one. Let f_k be the expected number of comparisons to put an a into B, when the number of b's between a and its predecessor in A is k. f_k does not depend on the index of the element being inserted. Then

$$T(m,n) = \max_{\substack{k_1, \ldots, k_m \\ \sum_{i=1}^m k_i \leq n}} \sum_{i=1}^m f_{k_i}.$$

Thus we are interested in the values f_k. If $k \geq 2$, then after the first comparison the element a jumps over two b's with probability p, and jumps over one b with probability $1 - p$. This means that we get the following recurrence relation for f_k:

$$f_k = (1-p)f_{k-1} + pf_{k-2} + 1,$$

with the initial conditions

$$f_0 = 2p + 1(1-p) = p + 1,$$

$$f_1 = 3p(1-p) + 2((1-p)^2 + p) = -p^2 + p + 2.$$

By standard technique, the solution of this linear recurrence is

$$f_k = \frac{k}{p+1} + \frac{p^3 + p^2 - p}{(p+1)^2}(-p)^k + \frac{2p^2 + 4p + 1}{(p+1)^2}.$$

This tells us that

$$\sum_{i=1}^m f_{k_i} = \sum_{i=1}^m \frac{k_i}{p+1} + \frac{p^3 + p^2 - p}{(p+1)^2} \sum_{i=1}^m (-p)^{k_i} + \frac{2p^2 + 4p + 1}{(p+1)^2} m.$$

The term $\sum_{i=1}^m \frac{k_i}{p+1}$ is always bounded from above by $\frac{n}{p+1}$. If $p = s$, we have $p^3 + p^2 - p = 0$, and the result follows. If $p > s$ (when $2 + s < n/m$), we have $p^3 + p^2 - p > 0$, and $\sum_{i=1}^m (-p)^{k_i}$ is maximized by choosing $k_i = 0$. Simple arithmetic gives the result also in this case. □

The global algorithm MERGE calls PROBMERGE with two lists of size m and $\lceil mx \rceil$ and then makes n binary searches, each in a list of size $2^t - 1$. Applying Theorem 2.1, we immediately get the following result.

THEOREM 2.2. *Let $E(m,n)$ denote the expected number of comparisons made by* MERGE. *For $n/m \leq r$ set $n/m = x$, $t = 0$, and for $n/m > r$ set $n/m = 2^t x$, $t \in N$, $r < x \leq 2r$. Then*

$$E(m,n) \leq \begin{cases} (t + sx + 1 + s)m + 1 & \text{if } (1+s) < x \leq 2+s, \\ (t + 2\sqrt{x})m + 1 & \text{if } 2+s < x \leq 2r. \end{cases}$$

COROLLARY 2.1. *The algorithm* MERGE *is significantly faster than binary merge for any fixed ratio $n/m > 1 + s$.*

Proof. With the notation of Theorem 2.2 we have

$$BM(m,n) - E(m,n) \geq \begin{cases} (x - sx - s)m - 3 & \text{if } (1+s) < x \leq 2, \\ ((1-s) - x(s - 1/2))m - 3 & \text{if } 2 < x \leq 2+s, \\ (2 - \sqrt{x})^2 m/2 - 3 & \text{if } 2+s < x \leq 2r. \end{cases}$$

This difference is $\Omega(m)$. □

Let us point out an interesting special case of the above result. When $n = 2m$, PROBMERGE merges the two lists with less than $2.855m$ expected comparisons, whereas the best known deterministic algorithm [6] performs $3m - 2$ comparisons.

Some more calculation also yields a relation between the expected running time of MERGE and the information theoretic lower bound.

COROLLARY 2.2. $E(m,n) - L(m,n) \leq 0.471m$.

We also claim that the probability value p was optimally chosen in the procedure PROBMERGE. Let $T_p(m,n)$ be the expected number of comparisons made by PROBMERGE when the probability $0 \leq p \leq 1$ is taken as a variable. Let us observe that we have just analyzed $T_s(m,n)$ and $T_{\sqrt{n/m}-1}(m,n)$ in Theorem 2.1. For the purpose of this analysis let us permit any input lists such that the ratio n/m is at least 1. The following theorem gives optimal choices for the probability value for all ranges of n/m.

THEOREM 2.3. *For every p, for every large enough m and n, we have*

$$T_p(m,n) \geq \begin{cases} T_0(m,n) & \text{if } 1 \leq n/m \leq 1+s, \\ T_s(m,n) & \text{if } 1+s \leq n/m \leq 2+s, \\ T_{\sqrt{n/m}-1}(m,n) & \text{if } 2+s \leq n/m \leq 4, \\ T_1(m,n) & \text{if } 4 \leq n/m. \end{cases}$$

Proof. We will examine the function $T_p(m,n)$ according to two cases.

Case 1. $p \geq s$. For $k_1 = \ldots = k_{m-1} = 0$ and $k_m = n$, the expected number of comparisons made by the procedure PROBMERGE is

$$U_p(m,n) = \left(\frac{n/m}{p+1} + p + 1\right) m + \frac{p^3 + p^2 - p}{(p+1)^2}((-p)^n - 1).$$

By definition, for every p, m, n we have $U_p(m,n) \leq T_p(m,n)$. We claim that for every large enough m and n, for every $p \neq s$, we have

$$T_s(m,n) < U_p(m,n) \quad \text{if} \quad 1+s < n/m \leq 2+s,$$

and that for every large enough m and n, for every $p \neq \sqrt{n/m} - 1$, we have

$$T_{\sqrt{n/m}-1}(m,n) < U_p(m,n) \quad \text{if} \quad 2+s \leq n/m.$$

Let us define the functions $V(p) = (\frac{n/m}{p+1} + p + 1)m$ and $D(p) = \frac{p^3+p^2-p}{(p+1)^2}((-p)^n - 1)$. Then $U_p(m,n) = V(p) + D(p)$. We will first prove the claim for the function $V(p)$ rather than the function $U_p(m,n)$. Its derivative is $V'(p) = (\frac{-n/m}{(p+1)^2} + 1)m$, and it is easy to check that the function $V(p)$ takes its global minimum in the set of real numbers at the point $\sqrt{n/m} - 1$. Moreover, $V(p)$ is an increasing function for $p \geq \sqrt{n/m} - 1$. Thus the minimum of the function $V(p)$ in the interval $[s, 1]$ is at the point s, if $\sqrt{n/m} - 1 \leq s$, and is at the point $\sqrt{n/m} - 1$, if $\sqrt{n/m} - 1 \geq s$. If we observe that $\sqrt{n/m} - 1 \leq s$ if and only if $n/m \leq 2 + s$, then the claim follows for the function V.

If $\sqrt{n/m} - 1 \geq s$ then for every p, we have $V(p) - V(\sqrt{n/m} - 1) = \Omega(m)$, while $D(\sqrt{n/m} - 1) - D(p) \leq |D(\sqrt{n/m} - 1)| + |D(p)| = O(1)$. This implies the claim for $U_p(m,n)$ when $\sqrt{n/m} - 1 \geq s$, and a similar argument works when $\sqrt{n/m} - 1 \leq s$.

In conclusion, we have shown that for every p in the interval $[s, 1]$, for every large enough m and n,

$$T_p(m,n) \geq T_s(m,n) \quad \text{if} \quad n/m \leq 2+s,$$

and similarly, for every p, for every large enough m and n,

$$T_p(m,n) \geq T_{\sqrt{n/m}-1}(m,n) \quad \text{if} \quad n/m \geq 2+s.$$

Case 2. $p \leq s$. We will show that for every p in the interval $[0, s]$, for every large enough m and n,

$$T_p(m,n) \geq T_0(m,n) \quad \text{if} \quad n/m \leq 1+s,$$

and similarly, for every p, for every large enough m and n,

$$T_p(m,n) \geq T_s(m,n) \quad \text{if} \quad n/m \geq 1+s.$$

This clearly implies the theorem. The function $U_p(m,n)$ that we use is when $k_1 = \dots k_{m-1} = 1$ and $k_m = n - m + 1$. This is because, in this range for p, $p^3 + p^2 - p$ is negative. For these choices of k_i we get,

$$U_p(m,n) = \left(\frac{-p^3 + 3p + 1 + n/m}{p+1}\right)m + \left(\frac{p^3+p^2-p}{(p+1)^2}\right)((-p)^{n-m+1} + p).$$

Once again we break up $U_p(m,n)$ into its two terms which we call $V(p)$ and $D(p)$. Thus $V(p) = (\frac{-p^3+3p+1+n/m}{p+1})m$ and $D(p) = (\frac{p^3+p^2-p}{(p+1)^2})((-p)^{n-m+1} + p)$. As before, we concentrate on $V(p)$, $D(p)$ being $O(1)$. We compute the derivative $V'(p)$ of $V(p)$ with respect to p:

$$V'(p) = \left(\frac{-2p^3 - 3p^2 + 2 - n/m}{(p+1)^2}\right)m.$$

Note that $V'(p)$ is a decreasing function in the range $[0, \infty]$. This means that the minimum value for $V(p)$ for p in the range $[0, s]$ is attained either at 0 or at s. $V(0) =$

$m(n/m+1)$ and $V(s) = m((1+s)+sn/m)$. It is easy to see that $V(s) - V(0)$ decreases as n/m increases. At $n/m = (1+s)$, $V(s) = V(0)$. This proves both statements of the claim at the beginning of case 2. □

Thus while $n/m \leq (1+s)$, our probabilistic procedure gives deterministic tape-merge as a special case. In the range $1+s \leq n/m \leq 2+s$ the best choice is the constant $p = s$. For $n/m > 2+s$, the best choice grows with n/m until 1, when the procedure (obviously) degenerates into a deterministic one. There is no reason to use PROBMERGE for large values of n/m; MERGE is significantly faster when $n/m > 2r$. In fact that is how r is chosen.

3. The lower bound. In this section we will prove the following theorem.

THEOREM 3.1. *For every $\eta > 0$ there exists $\epsilon > 0$ such that for every large enough m and n with $(1+\eta)m \leq n \leq (\sqrt{2}+1-\eta)m$, every insertive merging algorithm makes at least $(1+\epsilon)L(m,n)$ comparisons on the average.*

Proof. The proof depends on the following Unbalance Lemma.

UNBALANCE LEMMA. *Let T denote a binary decision tree with k leaves, let the set of internal nodes be $\{D_1, \ldots, D_{k-1}\}$, and let d_j be the number of leaves of the subtree with root D_j. Let us suppose that for some $J \subseteq \{1, 2, \ldots, k-1\}$, the subset of nodes $\{D_j : j \in J\}$ satisfies the following conditions:*

(i) *There exists $\epsilon_1 > 0$ such that $\sum_{j \in J} d_j \geq \epsilon_1 k \log k$.*

(ii) *There exists $\epsilon_2 > 0$ such that for every $j \in J$, the answer probabilities (fraction of leaves in left and right subtrees) at the node D_j lie outside the interval $(1/2 - \epsilon_2, 1/2 + \epsilon_2)$. Then, the average path length of T is at least*

$$(1 + \epsilon_1 \epsilon_2^2) \log k.$$

Proof. The average path length of T is

$$L(T) = \frac{1}{k} \sum_{1 \leq j \leq k-1} d_j.$$

Let p_j denote the answer probability at D_j. The information in T is

$$I(T) = \sum_{1 \leq j \leq k-1} d_j H_j,$$

where $H_j = p_j \log 1/p_j + q_j \log 1/q_j$ is the entropy function. Elementary information theory [9] tells us that we have necessarily

$$I(T) = k \log k.$$

For every j, we obviously have $H_j \leq 1$, and Taylor's formula gives $H_j \leq 1 - \epsilon_2^2$ for $p_j \leq 1/2 - \epsilon_2$. Thus we obtain

$$\sum_{j \notin J} d_j + \sum_{j \in J} d_j (1 - {\epsilon_2}^2) \geq k \log k,$$

and finally

$$L(T) \geq (1 + \epsilon_1 {\epsilon_2}^2) \log k. \qquad \square$$

Let us fix some insertive algorithm, and let T be the binary decision tree associated with this algorithm. Let the internal nodes of T be $D_1, \ldots, D_{k-1}$, where $k = \binom{m+n}{m}$.

Since the average running time of the algorithm is the average path length of T, it will suffice to prove that the assumptions of the Unbalance Lemma are fulfilled for some subset of the nodes and for some proper ϵ_1 and ϵ_2. To begin, notice that there is a bijection between the outcomes of a merging algorithm and the set Z of words containing precisely m letters a and n letters b. For any word $w \in \{a, b\}^*$, let $|w|_a$ and $|w|_b$ denote, respectively, the number of occurrences of a and b in w.

It follows from the hypothesis that for some sufficiently small $\zeta > 0$ we have

$$1/2 + 3\zeta < n/(m+n) < 1/\sqrt{2} - 3\zeta.$$

Let $c_3 > 0$ be an appropriate constant to be specified later. For any $m \le i \le n$ and $w \in Z$, let *decomp*(w, i) denote the decomposition $uvxy = w$, where the factors u, v, x, y have the following lengths:

- $|u| = i - 3c_3$,
- $|v| = |x| = 3c_3$,
- $|y| = m + n - i - 3c_3$.

Let us denote by Z_i the set of words $w \in Z$ such that *decomp*(w, i) satisfies the following conditions:

1. $vx = (bba)^{c_3}(abb)^{c_3}$,
2. for every decomposition $y = y'y''$, where $|y'| = l$; and for every decomposition $u = u''u'$, where $|u'| = l$, we have

$$(1/2 + \zeta)l - c_2 \le |y'|_b, |u'|_b \le (1/\sqrt{2} - \zeta)l + c_2,$$

where $c_2 > 0$ is an appropriate constant also to be specified later. We will define a subset of nodes satisfying the assumptions of the Unbalance Lemma using the sets Z_i. First we prove two claims about these sets.

CLAIM 1. *There exists a constant $\epsilon_3 > 0$ such that for all $m \le i \le n$ we have*

$$|Z_i| \ge \epsilon_3 |Z| .$$

Proof. Let w be a random element of Z, and let $m \le i \le n$. The first requirement is obviously satisfied by w with constant probability. We will show that under this hypothesis, the conditional probability that $|y'|_b$ falls outside the interval $((1/2 + \zeta)l - c_2, (1/\sqrt{2} - \zeta)l + c_2)$ is less then 0.48 . As the second requirement is symmetrical in $|y'|_b$ and $|u'|_b$, this will imply the claim. We shall (rather crudely) bound this probability by the sum $\sum_{1 \le l \le |y|} s(l)$, where $s(l)$ denotes the probability that the left factor of y of length l violates the second requirement. Clearly $s(l) = q(l) + r(l)$, where $q(l)$ is the probability that the left factor of y of length l contains less than $(1/2 + \zeta)l - c_2$ occurrences of b's, and $r(l)$ is the probability that the same left factor contains more than $(1/\sqrt{2} - \zeta)l + c_2$ occurrences of b's. Let us recall Hoeffding's bound [3] about sampling without replacement.

LEMMA (Hoeffding's bound). [3] *Let us suppose that we have an urn that contains $M + N$ letters, N of which are b's and M of which are a's. Let $p = N/(M + N)$. We are sampling the urn without replacement. Let X_j be the $0 - 1$ valued random variable which is 1 if and only if at the jth trial we get a letter b. Let $S_{i,p} = \sum_{j=1}^{i} X_j$. Then for every $\delta > 0$ we have*

$$P[S_{i,p} \ge (1 + \delta)ip] \le \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{ip},$$

and similarly, for every $0 < \gamma \leq 1$ *we have*

$$P[S_{i,p} \leq (1-\gamma)ip] \leq \left(\frac{e^{\gamma}}{(1+\gamma)^{1+\gamma}}\right)^{ip}.$$

We will use Hoeffding's bound in the following way: set $N = n - 4c_3$ and $M = m - 2c_3$ (recall that $4c_3$ is the number of b's and $2c_3$ is the number of a's in vx). Let $p = N/(M+N)$. Then for large enough m, we have:

$$1/2 + 2\zeta < N/(M+N) < 1/\sqrt{2} - 2\zeta.$$

Let $q(l)$ (respectively, $r(l)$) be the probability that in a random w which satisfies the first condition, the left factor of y of length l contains more (fewer) occurrences of b than the second condition permits. Then

$$q(l) < P[S_{l,1/\sqrt{2}-2\zeta} \geq (1/\sqrt{2} - \zeta)l],$$

and

$$r(l) < P[S_{l,1/2+2\zeta} \leq (1/2+\zeta)l].$$

Hoeffding's bound implies that there exists a constant $0 < \xi < 1$ which depends only on ζ such that

$$q(l), r(l) < \xi^l.$$

Let $c_1 = c_1(\xi)$ be an integer such that

$$\sum_{l=c_1}^{\infty} \xi^l < 0.24.$$

Now, as a function of c_1 we choose c_2 large enough that $q(l) = r(l) = 0$, whenever $l \leq c_1$. Hence

$$\sum_{1 \leq l \leq |y|} s(l) < 2\sum_{l=c_1}^{\infty} \xi^l < 0.48\,. \qquad \square$$

We are now ready to choose the constant c_3 as a function of c_2 such that the following Claim becomes true. Let $c_3 = c_2/((3/\sqrt{2}) - 2 - 3\zeta)$.

CLAIM 2. *Let* $w \in Z_i$ *for some* $m \leq i \leq n$, *and let* $uvxy = decomp(w,i)$. *Let* z *be a nonempty prefix of* xy (*a suffix of* uv) *such that either the letter which follows* (*precedes*) z *in* w *is an* a, *or there is no more* a *in* w *after* (*before*) z. *Then we have*

$$(1/2+\zeta) \leq |z|_b/|z| \leq (1/\sqrt{2} - \zeta).$$

Proof. We show the upper bound. If z is a prefix of x, then $|z|_b/|z| = 2/3$. Otherwise this fraction is maximized when the corresponding prefix of y contains the most possible b's. If this prefix is of length l, we have

$$|z|_b/|z| \leq \frac{2c_3 + (1/\sqrt{2} - \zeta)l + c_2}{3c_3 + l} = 1/\sqrt{2} - \zeta. \qquad \square$$

We can now define a subset of the nodes of T which satisfies the conditions of the Unbalance Lemma. For every couple (w, i) such that $m \le i \le n$ and $w \in Z_i$, we first define an internal node $V(i, w)$ of T. Let $uvxy = decomp(w, i)$, and let $h = h(w, i)$ be such that a_h is the first instance of a in x (thus $h - 1$ is the number of instances of a in uv). If on the path of T corresponding to w the comparisons involving a_{h-1} precede those involving a_h, then let $V(w, i)$ be the node of the decision tree at which the first comparison involving a_h takes place. Otherwise let $V(w, i)$ be the node at which the first comparison involving a_{h-1} takes place. Finally let

$$J = \{j : D_j = V(w, i) \text{ for some } w \in Z_i\}.$$

We now show that the two conditions of the Unbalance Lemma are met by the nodes whose indices are in J. First we will show that

$$\sum_{j \in J} d_j \ge \sum_{m \le i \le n} |Z_i|.$$

On the right-hand side every word $w \in \bigcup_{m \le i \le n} Z_i$ is counted with multiplicity t_w, where

$$t_w = |\{i : w \in Z_i\}|.$$

If $i \ne i'$, then $V(w, i) \ne V(w, i')$, because $|h(w, i) - h(w, i')| > 1$. Thus the leaf corresponding to w is counted on the left-hand side at least t_w times. Therefore by using Claim 1 we get

$$\sum_{j \in J} d_j \ge \epsilon_3 m \binom{m+n}{m} = \epsilon_1 k \log k.$$

We now turn to the second condition of the Unbalance Lemma. Let $D_j = V(w, i)$ for some $w \in Z_i$, and let us suppose without loss of generality that the comparisons involving a_{h-1} precede those involving a_h. Let a_{h+r} denote the leftmost right neighbour of a_h which has already been treated when the comparisons involving a_h begin at D_j. (If there is no such element we set $r = m - h + 1$.) Let us further suppose that a_{h-1} and a_{h+r} are separated by s occurrences of b for some $s \ge 1$, which we denote by $b_e, \dots, b_{e+s-1}$. Let us observe that $s/(r+s) = |z|_b/|z|$ for some initial segment z of xy, where z satisfies the conditions of Claim 2. Therefore we have

$$1/2 + \zeta \le s/(r+s) \le 1/\sqrt{2} - \zeta.$$

The comparison at D_j is $a_h : b_{e+l-1}$ for some $1 \le l \le s$. Let $p(r, s, l) = Pr[a_h > b_{e+l-1}]$ denote the answer probability at D_j. Among the leaves of $V(w, i)$, the relative rankings of the sets $\{a_h, \dots, a_{h+r-1}\}$ and $\{b_e, \dots, b_{e+s-1}\}$ are all equally represented. For any word belonging to the leaves of D_j, the relative rank of a_h within the pooled set is at least l if and only if $a_h > b_{e+l-1}$. Thus we have

$$p(r, s, l) = \frac{s(s-1)\dots(s-l+1)}{(r+s)(r+s-1)\dots(r+s-l+1)}$$

We claim that for every l, the answer probability $p(r, s, l)$ falls outside the interval $(1/2 - \epsilon_2, 1/2 + \epsilon_2)$, for $\epsilon_2 = \min\{\zeta, 1/2 - (1/\sqrt{2} - \zeta)^2\}$. We prove this in two cases according to the value of l. If $l = 1$, then $p(r, s, 1) = s/(r+s) \ge 1/2 + \zeta$. If $l \ge 2$, then we have

$$p(r, s, l) \le \left(\frac{s}{r+s}\right)^2 \le (1/\sqrt{2} - \zeta)^2,$$

which means that the second condition of the Unbalance Lemma is also satisfied. We now set $\epsilon = \epsilon_1\epsilon_2^2$. The conclusion of the Unbalance Lemma gives exactly what we wanted to prove. □

4. Open problems. We have shown how to make significant improvements upon the tape merge algorithm when the ratio n/m is at least the golden ratio. An interesting open question is the value of the smallest ratio n/m where a significant improvement — probabilistic or deterministic — can be obtained. Although the lower bounds in [2], [10] hold only for deterministic algorithms, we conjecture that not even a probabilistic algorithm can achieve improvements over tape merge for ratios less than the golden ratio.

It would be interesting to design other, more complex probabilistic algorithms. Generalizing our average case lower bound for arbitrary merging algorithms also remains open.

Acknowledgments. Thanks are due to Charles Delorme for his most valuable assistance. We would also like to thank Russell Impagliazzo for his help in simplifying the analysis of the probabilistic algorithm.

REFERENCES

[1] C. Christen, *Improving the bound on optimal merging*, in Proceedings of the 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 259–266.
[2] ———, *On the optimality of the straight merging algorithm*, Tech. Report Publication 296, Dép. d'Info. et de Rech. Op., Université de Montréal, Montréal, Québec, Canada, 1978.
[3] V. Chvatal, *Probabilistic methods in graph theory*, Ann. Oper. Res., 1 (1984), pp. 171–182.
[4] F. K. Hwang and D. N. Deutsch, *A class of merging algorithms*, J. Assoc. Comput. Mach., 20 (1973), pp. 148–159.
[5] F. K. Hwang and S. Lin, *Optimal merging of 2 elements with n elements*, Acta Inform., 1 (1971), pp. 145–158.
[6] ———, *A simple algorithm for merging two disjoint linearly ordered lists*, SIAM J. Comput., 1 (1972), pp. 31–39.
[7] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[8] G. K. Manacher, *Significant improvements to the Hwang-Ling merging algorithm*, J. Assoc. Comput., Mach., 26 (1979), pp. 434–440.
[9] C. F. Picard, *Graphes et questionnaires*, Gauthiers-Villars, Paris, 1973.
[10] P. K. Stockmeyer and F. F. Yao, *On the optimality of linear merge*, SIAM J. Comput., 9 (1980), pp. 85–90.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 272–283, April 1993

OPTIMAL RANDOMIZED ALGORITHMS FOR LOCAL SORTING AND SET-MAXIMA*

WAYNE GODDARD[†], CLAIRE KENYON[‡], VALERIE KING[§], AND LEONARD J. SCHULMAN[¶]

Abstract. Randomized algorithms for two sorting problems are presented. In the local sorting problem, a graph is given in which each vertex is assigned an element of a total order, and the task is to determine the relative order of every pair of adjacent vertices. In the set-maxima problem, a collection of sets whose elements are drawn from a total order is given, and the task is to determine the maximum element in each set. Lower bounds for the problems in the comparison model are described and it is shown that the algorithms are optimal within a constant factor.

Key words. sorting, randomized algorithms, comparison model, partial order, graph algorithms

AMS(MOS) subject classifications. 68Q, 05C

1. Introduction. In this paper we study two sorting problems. The first is the *local sorting* problem: given a graph in which each vertex is assigned an element of a total order, one must determine the relative order of every pair of adjacent nodes. This problem restricts to standard sorting when the graph is complete, but in general its complexity depends on the graph selected. The second problem is *set-maxima*, where the task is to identify the maximum element in each of a collection of sets drawn from a total order. Set-maxima was investigated in [1], [3], and [5]. Local sorting appears to be new, although a restricted version was suggested in [6].

We present randomized algorithms for these problems and measure their complexity in the comparison (decision-tree) model. In that model, one pays only for comparisons between elements of the total order. The algorithms use random bits to help determine which comparisons will be made, but their outcomes must be correct. The complexity of the algorithm is thus the expected number of comparisons made on a worst-case input. We obtain information-theoretic lower bounds for both problems and show that our algorithms attain these bounds.

The output of a local-sorting algorithm is exactly an acyclic orientation of the graph G. Thus an information-theoretic lower bound on the complexity of local sorting, even for randomized algorithms, is given by the logarithm of the number, $\alpha(G)$, of acyclic orientations of the graph G. We present an algorithm that is optimal for all graphs—it makes an expected $\Theta(\log \alpha(G))$ comparisons.

We derive an estimate of $\alpha(G)$ in terms of the degrees, $\{d_v\}$, of the vertices $\{v\}$ of G: specifically $\log \alpha(G)$ is $\Theta(\sum_{v \in G} \log(d_v + 1))$. Thus the number of comparisons

*Received by the editors January 11, 1991; accepted for publication (in revised form) December 15, 1991. This paper includes the extended abstracts [2] and [4].

[†]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by National Science Foundation grant 8912586-CCR and Defense Advanced Research Project Agency contract N00014-89-J-1988.

[‡]LIENS, Ecole Normale Supérieure, 75230 Paris Cedex 05, France. The research of this author was supported in part by the Institut de Recherche d'Informatique et d'Automatique.

[§]NEC, Princeton, New Jersey 08540. The research of this author was supported in part by a grant from the Natural Sciences and Engineering Council of Canada and the ITRC at the University of Toronto, Toronto, Ontario, Canada.

[¶]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by National Science Foundation grant 8912586-CCR, Defense Advanced Research Project Agency contract N00014-89-J-1988 and an Office of Naval Research Graduate Fellowship.

our algorithm makes contrasts with the $\Theta(\sum_{v\in G} d_v)$ comparisons made by the naive algorithm, which examines all edges.

Closely related to local sorting is the *set-sort* problem: sort each of a family of possibly overlapping sets. Set-sort is equivalent to locally sorting the graph in which each set induces a clique. We also observe that local sorting may be thought of as a special case of set-maxima (where every set is a pair).

One application of local sorting is to the *element uniqueness problem on a graph*: given a graph G with each vertex assigned an element of a total order, are all pairs of adjacent values distinct? Manber and Tompa [7] introduced this question and showed that $\log \alpha(G)$ comparisons are necessary for this decision problem. Up to a constant factor, this is the bound achieved by our local sorting procedure (which can identify every equality).

The second problem we study, set-maxima, was introduced in [3], where an argument of Fredman was presented showing that, given m sets over an n-element universe, at most $\binom{m+n-1}{n-1}$ arrangements of maxima are possible. We in fact observe an information-theoretic lower bound of $\Omega(n\log(m/n)+n)$, matching this limitation.

We describe a randomized algorithm for set-maxima that performs an expected $O(n\log(m/n)+n)$ comparisons. By the above this is optimal in terms of n and m.

Local sorting provides a reasonable solution to the *partial order verification* problem suggested by A. Yao [10]: given a directed graph G with elements of a total order at each of the n vertices, verify that the orientation of every edge agrees with the order. But it is best to reduce the problem to set-maxima. In the reduction each set consists of the immediate predecessors of a vertex of the graph. Our set-maxima algorithm yields an optimal $O(n)$-comparison algorithm for this problem.

Deterministic algorithms are known for a few special cases of set-maxima. Komlós [5] made use of a reduction to set-maxima when he solved the *minimum spanning tree verification* problem. His deterministic algorithm finds the tree edge of largest weight in every simple cycle containing exactly one nontree edge. It solves this instance of set-maxima in $O(n\log(m/n))$ comparisons, where n and m are the number of tree and nontree edges. Then it verifies that the every nontree edge is greater than the appropriate maximum. We have been informed of an $O(m+n)$-time implementation of this algorithm by Tarjan [9].

Recently Bar-Noy, Motwani, and J. Naor [1] gave a deterministic algorithm that uses $O(n)$ comparisons when the n sets are the hyperplanes in a projective geometry. They also gave an $O(n)$-comparison algorithm for the case where n sets are chosen randomly, so that each element appears in each set with probability $p(n)$.

This paper has three main sections: the first deals with local sorting, the second with set-maxima, and some open problems are discussed in the third.

2. Local sorting. At the heart of our approach to local sorting is the *limited-degree* algorithm (§2.1) whose complexity, $O(n\log(\Delta+1))$, is a function only of the number n of vertices and the maximum degree Δ of a graph G (where Δ may be a function of n). However, the maximum degree of G is too crude a measure of its complexity for local sorting. A more accurate measure is the logarithm of $\mathcal{D}(G)$, defined by:

$$\mathcal{D}(G) = \prod_{v\in G}(d_v+1),$$

where d_v is the degree of v in G. In §2.3 we show that $\log \mathcal{D}(G)$ is $\Theta(\log \alpha(G))$, which is a lower bound on the complexity of local sorting. In §2.2 we describe a *reduction* procedure that alters the graph so that we can achieve the optimal $\Theta(\log \mathcal{D}(G))$ comparisons with several applications of the limited-degree algorithm.

We present our results under the assumption that the values at the vertices are distinct. Minor modifications suffice to cover the more general case where equalities are allowed, and "local sorting" includes reporting the equalities as such.

2.1. The limited-degree local sorting algorithm. We show here how to sort locally in a graph with vertex set X ($|X| = n$) and maximum degree Δ. The idea is to take a series of increasingly large random samples of the vertices and partially order each sample using the information given by the partial order on the previous sample.

We produce a series of samples $R_k, R_{k-1}, \ldots, R_0 = X$ (where k will be defined later). Starting with R_k, we then:

- Partially order each R_i so that it is *locally sorted to radius* 2^{2i}.

By this we mean that the relative order of vertices in R_i is known if, as measured in G, they are within distance 2^{2i} of each other. The end result is that $R_0 = X$ is locally sorted to radius 1, as called for.

Algorithm. We choose the samples as follows. We let $R_0 = X$. Then we randomly choose R_i from within R_{i-1} by taking elements independently with probability p_i/p_{i-1}. Thus the sample R_i has expected size np_i. (The descending sequence p_i will be specified later.)

We continue until k such that R_k has expected size $O(1)$. This we then sort using an expected $O(1)$ comparisons.

The main part of the algorithm is an iterative process in which the partial order on R_i is used to obtain the requisite partial order on R_{i-1}. The R_i are used as "signposts" for the two parts of this process. Let B_x^r denote the set of vertices of distance at most r from vertex x. Proceed as follows.

Step 1. *For each $x \in R_{i-1}$ do:*

- *Find the rank of x in $R_i \cap B_x^{2^{2i-1}}$ using a binary search.*

Any two elements of R_i in $B_x^{2^{2i-1}}$ are within a distance of 2^{2i} of each other—so, since R_i is locally sorted to distance 2^{2i}, their relative order is already known (i.e., $R_i \cap B_x^{2^{2i-1}}$ has been totally ordered). This allows us to find x's rank with a binary search.

Step 2. *For each $x \in R_{i-1}$ do:*

- *Determine C_x, the set of all $y \in R_{i-1} \cap B_x^{2^{2i-2}}$ that have the same rank as x with respect to $R_i \cap B_x^{2^{2i-2}}$.*

Note that for y in $B_x^{2^{2i-2}}$, $B_y^{2^{2i-1}}$ contains $B_x^{2^{2i-2}}$. (See Fig.2.1.) Thus C_x can be constructed using no further comparisons.

Step 3. *For each pair $x, y \in R_{i-1}$ do:*

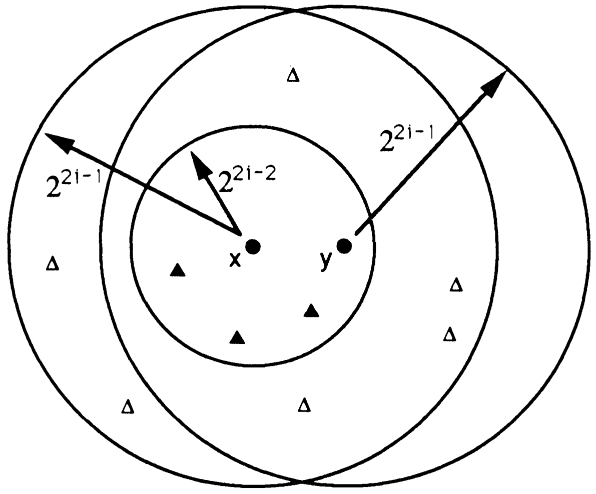
- *If $x \in C_y$ and $y \in C_x$ then compare x and y.*

If either $x \notin C_y$ or $y \notin C_x$ while x and y are at most distance 2^{2i-2} apart, then there is at least one intervening signpost (element of $R_i \cap B_x^{2^{2i-2}}$ or $R_i \cap B_y^{2^{2i-2}}$ whose value is between x and y), and therefore their relative order is known. Thus Step 3 is sufficient to determine for each $x \in R_{i-1}$ the relation between it and every element of $R_{i-1} \cap B_x^{2^{2i-2}}$. In other words Step 3 locally sorts R_{i-1} to radius 2^{2i-2}.

Analysis. How many comparisons do we use?

Step-by-step analysis. For each $x \in R_{i-1}$, the number of comparisons in Step 1 is at most $\log |B_x^{2^{2i-1}}|$ which, by our assumption on the degrees, is at most $2^{2i-1} \log(\Delta + 1)$. Thus the expected number of comparisons for the whole of Step 1 is at most

$$np_{i-1} 2^{2i-1} \log(\Delta + 1). \tag{2.1}$$



(▲, △ : signposts for x ; ▲ : signposts determining C_x)

FIG. 2.1. *y is a candidate for C_x.*

Step 2 involves no comparisons.

We bound the number of comparisons in Step 3 by $\frac{1}{2}\sum_{x\in R_{i-1}} |C_x|$, since each $x \in R_{i-1}$ is involved in at most $|C_x - \{x\}|$ comparisons. The following lemma provides a bound on the conditional expectation of $|C_x|$ given R_{i-1}.

LEMMA 2.1. *For any R_{i-1} and any $x \in R_{i-1}$, the expected size of C_x is at most $2p_{i-1}/p_i$.*

Proof. Consider first the set B^+ consisting of those elements of $B_x^{2^{2i-2}} \cap R_{i-1}$ greater than x. Then let z^+ be the smallest element of $B^+ \cap R_i$, and note that $B^+ \cap C_x$ consists of those elements of B^+ less than z^+. Each element of B^+ is in R_i with probability $p = p_i/p_{i-1}$. Hence (as this is a Bernoulli process), if B^+ were infinite, $|B^+ \cap C_x| + 1$ would have a geometric distribution with parameter p and thus expected value $1/p$. The finiteness of B^+ only reduces this value.

Now define B^- and z^- analogously (so that $E(|B^- \cap C_x|) < (1-p)/p$). As $C_x = (B^+ \cap C_x) \cup (B^- \cap C_x) \cup \{x\}$, it follows that $E(|C_x|) < 1 + 2(1-p)/p$. □

Thus the expected number of comparisons in Step 3, given *any* particular R_{i-1}, is bounded by $|R_{i-1}|p_{i-1}/p_i$. Therefore the expected number of comparisons for Step 3 is at most the expectation (ranging over R_{i-1}) of this bound, which is

$$p_{i-1}/p_i E(|R_{i-1}|) = np_{i-1}^2/p_i. \tag{2.2}$$

Definition of the parameters. A good choice of the $\{p_i\}$ is obtained by balancing the costs (2.1) and (2.2). This requires $2^{2i-1}\log(\Delta+1) = p_{i-1}/p_i$. Solving for p_i in terms of p_{i-1} and noting that $p_0 = 1$, this leads us to choose

$$p_i = \frac{1}{(\log(\Delta+1))^i 2^{i^2}}.$$

The value k is chosen such that $E(|R_k|) = O(1)$. For this it is sufficient that $k = \left\lceil \sqrt{\log n} \right\rceil$.

Total number of comparisons. Substituting this choice of p_i into bounds (2.1) and (2.2) and summing, it follows that the expected number of comparisons in going from R_i to R_{i-1} is at most

$$\frac{n}{(\log(\Delta+1))^{i-2}2^{i^2-4i+1}}.$$

Therefore the expected number of comparisons in going from R_k to R_1 is at most

$$\sum_{i=2}^{k} \frac{n}{(\log(\Delta+1))^{i-2}2^{i^2-4i+1}} \leq n \sum_{i \geq 2} 2^{-i^2+4i-1} < 13n.$$

(This bound is independent of Δ: this is possible because Δ determined the size of R_1.) Thus the overall number of comparisons is dominated by the transition from R_1 to R_0, and we have the following theorem.

THEOREM 2.2. *The limited-degree local sorting algorithm requires*

$$E(\textit{comparisons}) \leq 4n \log(\Delta+1) + 13n$$

on a graph with n vertices and maximum degree Δ.

As an example, if the degree of every vertex is polylogarithmic in n, then the procedure runs in $O(n \log\log n)$ comparisons.

2.2. A reduction for arbitrary graphs. In this subsection we show how to sort locally an arbitrary graph in $O(\log \mathcal{D}(G))$ comparisons. This is achieved through a reduction to the limited-degree case. In the reduction we transform G into a new graph H in which every edge of G is represented exactly once and each vertex of G is represented several times. H is constructed such that locally sorting each component of H with the limited-degree algorithm represents an efficient way of locally sorting G. Our reduction is deterministic and requires no comparisons.

Given a graph G, first discard all isolated vertices. Then let U_0 be the set of vertices of degree 1, and $G_0 = G - U_0$. We define U_i and G_i inductively for $i \geq 0$ by: U_i is the set of vertices of degree at most $a_i = 2^{2^i} - 1$ in G_{i-1}, and $G_i = G_{i-1} - U_i$. Halt when G_i is the null graph.

We now consider the graph whose vertex set is $U_i \cup N_{i-1}(U_i)$ (where $N_{i-1}(U_i)$ denotes the set of neighbors of U_i in G_{i-1}), and whose edges are the edges that are interior to U_i or that connect U_i to $N_{i-1}(U_i)$. In that graph, some vertices of $N_{i-1}(U_i)$, the "red" vertices, have degree at most a_i. Others have degree higher than a_i: we split each of these into several "green" vertices each retaining between $a_i/2$ and a_i of its originator's edges. (The coloring will be used in the analysis.) Let H_i be this new graph. An example is shown in Fig. 2.2.

We now apply the limited-degree local sorting algorithm (§2.1) to each H_i. This is equivalent to locally sorting G.

THEOREM 2.3. *Given any graph G, the local sorting algorithm uses an expected $O(\log \mathcal{D}(G))$ comparisons.*

Proof. Let $h_i = |H_i|$. The maximum degree of H_i is at most a_i. Thus we require an expected $O(\sum_i h_i \log(a_i+1))$ comparisons to sort H locally. We separate the contributions to $\eta = \sum_i h_i \log(a_i+1)$ into the contribution of the vertices in $\bigcup U_i$, the contribution of the red vertices, and that of the green vertices. (Note that $\log(a_i+1) = 2^i$.)

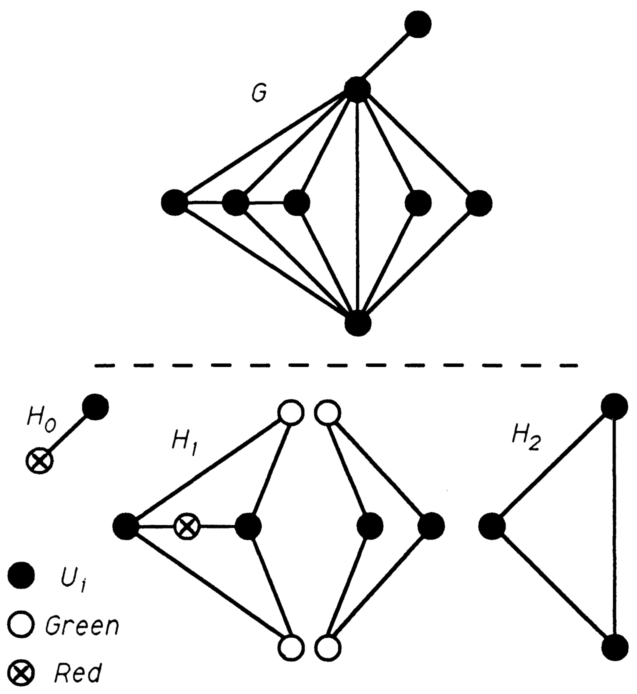


FIG. 2.2. *Example of the graph reduction.*

Every nonisolated vertex v of G shows up, perhaps lacking some of its original edges, in just one U_i. For each $j < i$, $N_{j-1}(U_j)$ may contain vertices corresponding to v: either several green vertices, or just one red vertex. Note that $i \leq \lceil \log\log(d_v + 1)\rceil$.

Consider a vertex v' in U_i, derived from v in G. As v was *not* chosen for U_{i-1}, its degree d_v must exceed a_{i-1}. Therefore $\log(d_v + 1) \geq \log(a_{i-1} + 1) = \frac{1}{2}\log(a_i + 1)$. Thus the contribution of $\bigcup U_i$ to η is at most $2\log\mathcal{D}(G)$.

A vertex v of G has at most one red vertex derived from it in each $N_{i-1}(U_i)$ for $j \leq \lceil \log\log(d_v + 1)\rceil - 1$. Hence its total red contribution is at most

$$\sum_{j=1}^{\lceil \log\log(d_v+1)\rceil - 1} 2^j \leq 2\log(d_v + 1).$$

Therefore the contribution of the red vertices is at most $2\log\mathcal{D}(G)$.

The green vertices of H_i are adjacent only to vertices of U_i. Further, green vertices of H_i are of degree at least $a_i/2$, whereas those of U_i are of degree at most a_i. Thus there are at most $2|U_i|$ green vertices, and their contribution is at most twice that from $\bigcup U_i$.

Combining all the above, it follows that $\eta \leq 8\log\mathcal{D}(G)$. □

Set-sort. Using the concavity of the log function, we can conclude from the above that if G has m edges, then our local sorting algorithm uses $O(n\log((2m + n)/n))$ comparisons. Applying this to the set-sort problem, we obtain the following corollary.

COROLLARY 2.4. *Let $S_1, S_2, \ldots, S_m$ be sets from a totally ordered universe of size n. Then these sets can be sorted using an expected $O(n\log(\sum_j |S_j|^2/n))$ comparisons.*

2.3. The acyclic orientations of a graph. In this subsection we provide an estimate of the number $\alpha(G)$ of acyclic orientations of a graph G, based solely on the degrees of the vertices of G. The upper bound is due to Manber and Tompa [7]. (The parameter α has also been studied, for instance, in [8].)

Define

$$\mathcal{F}(G) = \prod_{v \in G} f(d_v + 1),$$

where d_v is the degree of v in G, and $f(x) = (x!)^{1/x}$.

THEOREM 2.5. *For any graph G,*

$$\sqrt{\mathcal{D}(G)} \le \mathcal{F}(G) \le \alpha(G) \le \mathcal{D}(G).$$

In particular, $\log \mathcal{D}(G) = \Theta(\log \alpha(G))$.

Proof. The proof that $\alpha(G) \le \mathcal{D}(G)$ is in [7]. For positive integral x, $\sqrt{x+1} \le f(x+1)$, hence $\sqrt{\mathcal{D}(G)} \le \mathcal{F}(G)$. Now we show that $\mathcal{F}(G) \le \alpha(G)$.

The proof is by induction on the number of vertices of G. The case of a single vertex is trivial.

Let v be a vertex of minimum degree δ. If $\delta = 0$, then $\alpha(G) = \alpha(G - v)$ and $\mathcal{F}(G) = \mathcal{F}(G - v)$; so hereafter we assume that $\delta \ge 1$. We prove first that

$$\alpha(G) \ge (\delta + 1)\alpha(G - v). \tag{2.3}$$

Take any acyclic orientation $\mathcal{A}$ of $G - v$ and look at v's neighbors N in G. If N is totally ordered by $\mathcal{A}$, then there are $\delta + 1$ ways of extending $\mathcal{A}$ to an acyclic orientation of G (by choosing v's rank with respect to N); otherwise there are even more ways to extend $\mathcal{A}$.

The next thing to notice is that

$$\mathcal{F}(G) = \mathcal{F}(G - v) f(\delta + 1) \left(\prod_{v \in N} \frac{f(d_v + 1)}{f(d_v)} \right),$$

where $\{d_v\}$ indicate degrees in G (each greater than or equal to δ). It can be shown that $f(x + 1)/f(x)$ is monotone decreasing for positive integral x. Therefore

$$\mathcal{F}(G) \le \mathcal{F}(G - v) f(\delta + 1) \left(\frac{f(\delta + 1)}{f(\delta)} \right)^{\delta}.$$

Thus from inequality (2.3) above and the induction hypothesis, we find that

$$\alpha(G) \ge (\delta + 1)\mathcal{F}(G - v) \ge \mathcal{F}(G) \frac{(\delta + 1)(f(\delta))^{\delta}}{(f(\delta + 1))^{\delta + 1}} = \mathcal{F}(G). \qquad \square$$

The bounds on $\alpha(G)$ are the best possible of the form $\prod_v g(d_v)$. The lower bound is attained by any disjoint union of cliques. For the upper bound consider $G = K(a, b)$ with $b \gg a$: here $\alpha(G) \sim a!(a + 1)^b$. (We would like to thank Mark Haiman for discussions on this point.) The latter graphs also disprove the upper bound $\prod_v \max\{2, d_v\}$ claimed in [6].

Conclusion. The results of this section establish the following theorem.

THEOREM 2.6. *For any graph G, the local sorting algorithm uses an expected*

$$\Theta(\log \alpha(G)) = \Theta \left(\sum_{v \in G} \log(d_v + 1) \right)$$

comparisons and is optimal up to a constant factor.

3. Set-maxima. One can use the set-sort algorithm to solve set-maxima; this is fine when the sets are very small, but it is inefficient for large sets. The key idea in handling large sets is a *reduction* process: take a random sample, partially sort it, and then use this to reduce the sizes of the sets. Another useful idea is a generalization of set-maxima (called *t-maxima* in this paper), in which one must find and sort the t largest elements in each set.

In §3.1 we discuss our approach to set-maxima, and in §3.2 we provide the algorithm for t-maxima. In §3.3 we analyze the algorithm and show that for $t = O(1)$ it uses an expected $O(n \log(m/n) + n)$ comparisons, where m is the number of sets and n the size of the universe. This we show is optimal as a function of m and n.

3.1. The approach to set-maxima. In this subsection we outline the set-maxima algorithm. Assume we have sets $S_1, S_2, \dots, S_m$ in a universe X of size n. Our general strategy for set-maxima can be summarized in three stages.

1. Choose a random sample R, and determine for each set S_i a "representative" r_i which is the maximum sample point in that set.
2. Determine for each set S_i the reduced set S_i' formed by discarding those elements less than the representative.
3. Solve set-maxima on the reduced system.

Observe that the maximum element in the reduced set is the same as that in the original set.

Implementation of Stage 2 if *R* is sorted. We look here at an efficient implementation of Stage 2 when the sample R has been sorted. Define for each x the "dual" set

$$T_x = \{ r_j : x \in S_j \}.$$

To determine the reduced sets we need to determine the relationship between x and r_j for all j and $x \in S_j$. This is equivalent to finding for each x, the set $\{ r_j : r_j \in T_x \,\&\, r_j < x \}$.

We perform a "doubling" search, rather than a binary search, to find the interval of T_x in which x lies: One compares x with the elements of T_x of ranks $1, 2, 4, \dots, 2^l$ from the bottom, until an element greater than x (if it exists) is found, and then performs a binary search in the interval $[2^{l-1}, 2^l)$ to find where x lies.

A doubling search is more efficient than an ordinary binary search because the average x is likely to be smaller than most of the representatives, and thus near the bottom of T_x.

Example. Let m be $\Theta(n \log^c n)$. We solve set-maxima as follows. First we take a random sample R where each element of X is chosen independently with probability $1/\log n$. Then we sort R, achieving Stage 1. It can then be shown that the expected size of each reduced set is $O(\log n)$, and thus that the doubling searches of Stage 2 take a total of $O(n \log\log n)$ comparisons. (Proofs omitted.) We then use an expected $O(n \log\log n)$ comparisons to apply the set-sort algorithm to the reduced sets, determining their maxima. Thus the entire procedure requires an expected $O(n \log\log n) = O(n \log(m/n))$ comparisons.

Generalization. It appears that the $O(n \log\log n)$ comparisons of the example is the best one can do with a completely sorted sample. However, if m is small this quantity is $\omega(n \log(m/n))$, and so, in general, we cannot afford to sort the sample completely. We therefore must handle Stage 2 differently. We note that for most $x \in X$, the search of T_x ends after x has been compared with only the smallest elements of T_x. This motivates an extension of set-maxima.

DEFINITION. In the *t-maxima* problem, one is given a set system and a parameter t, and one must find and sort the t largest elements of each set. The *t-minima* problem is defined analogously.

It is this level of generality that proves robust in our recursion.

We implement Stage 2 as follows. We first invoke u-minima on the $\{T_x\}$ to find and sort the u smallest elements in each T_x. (The parameter u will be specified later.) We then attempt for each x the doubling search of T_x as before. But we are able to complete the doubling search only when x is smaller than the uth smallest element of T_x. We set aside the few "bad" x's for which the doubling searches fail, and handle them separately.

3.2. The set-maxima algorithm. In this subsection we present the full set-maxima algorithm. Given an element x and a set A containing x, we say that the rank (bottom-rank) of x in A is the number of elements not less than (not greater than) x.

We are given a universe $\mathcal{X}$ of n elements from some total order; and $S_1, S_2, \ldots, S_m$, subsets of $\mathcal{X}$. We will assume that $m \geq 2n$.

The algorithm **Large** is defined below, recursively for X, $S_1, S_2, \ldots, S_m$ subsets of the universe, and a parameter $t \geq 16$. **Large** solves t-maxima on the collection $\{X \cap S_j\}_{j \leq m}$. **Small** is defined analogously for t-minima.

We call the procedure **Large** $(16, \mathcal{X}, \{S_j\}_{j \leq m})$ to solve the set-maxima problem on $\mathcal{X}$ and $S_1, S_2, \ldots, S_m$.

```
Large(t, X, {S_j}_{j≤m}):
    If |X| ≤ 16 then sort X.
    Else
Step 1. Let R be the sample generated by choosing each x ∈ X independently with
    probability 1/t.
    Do Large(t, R, {S_j}). For 1 ≤ j ≤ m, let r_j be the element of rank t in R ∩ S_j, or
    −∞ if there is no such element.
```

The call to **Large**$(t, R, \{S_j\})$ finds the t largest sample elements in each set S_j. Then, rather than taking the representative to be the maximum sample element in each set, we choose the tth largest value. This is so that each reduced set contains at least t elements, if the original set started with that many.

The implementation of Stage 2 has three steps:

Step 2A. *For each* $x \in X$, *let* $T_x = \{r_j : S_j \ni x\}$. *Do* **Small**$(mt^3/n, R, \{T_x\}_{x \in X})$. This finds and sorts the mt^3/n smallest elements in each T_x.

Step 2B. *For each* $x \in X$, *compare* x *with the element* τ_x *of* T_x *of rank* mt^3/n *from the bottom. If* $x > \tau_x$, *then put* x *in* Y.

If $x \leq \tau_x$, *then do a doubling search on the sorted portion of* T_x : *compare* x *to the elements of* T_x *of bottom-ranks* $2^0, 2^1, \ldots$ *until you find* x *is less than some element of bottom-rank* $2^k \leq mt^3/n$. *Then perform a binary search in the interval between* 2^{k-1} *and* 2^k *until* x *is placed among the elements of* T_x.

If x is high up in T_x, specifically, greater than the element of bottom-rank mt^3/n, it is placed aside in Y. Otherwise a doubling search is performed. After this we know for each S_j the subset S_j^1 consisting of the $x \in S_j \cap (X - Y)$ which are at least as big as the representative r_j.

Step 2C. *Do* **Large**$(t, Y, \{S_j\})$.

This yields for each S_j the subset S_j^2 consisting of the t largest elements in $S_j \cap Y$ (if there are that many). By combining S_j^1 and S_j^2, we obtain for each S_j the reduced set S'_j, which contains its t largest elements (if S_j had that many originally).

Step 3. *Let* $S'_j = \{x \in S_j \cap (X - Y) : x \geq r_j\} \cup \{$*largest t elements of* $S_j \cap Y\}$.
Apply the set-sort algorithm (*cf. Corollary* 2.4) *to the sets* $\{S'_j\}_{j \leq m}$.

Step 3 sorts each reduced set, and yields the t largest elements in each $S_j \cap X$, as required.

3.3. Analysis. We analyze the expected number of comparisons performed during the procedure **Large**$(t, X, \{S_j\}_{j \leq m})$. All comparisons occur in Steps 2B and 3, and in the recursive calls.

Comparisons in Steps 2B and 3. For $1 \leq j \leq m$ let $\rho(r_j)$ denote the rank of r_j in $S_j \cap X$.

The number of comparisons needed for each element $x \in X$ in Step 2B is the cost of searching for the interval in which x lies and, if one is found, the cost of locating x within the interval. Thus, Step 2B requires at most

$$\sum_{x \in X} 2 \log (|\{ j : r_j \leq x,\ x \in S_j \cap X \}|)$$

comparisons. But $\sum_{x \in X} |\{ j : r_j \leq x,\ x \in S_j \cap X \}|$ is equal to $\sum_j |\{ x \in S_j \cap X : x \geq r_j \}|$, the sum of the ranks of the representatives. Hence, by the concavity of the log function, Step 2B requires at most

$$2n \log \left(\sum_j \rho(r_j)/n \right) \tag{3.1}$$

comparisons.

In Step 3, the m sets S'_j have size at most $t + \rho(r_j)$. Therefore the set-sort algorithm (cf. Corollary 2.4) takes at most

$$c_0 n \log \left(\sum_j (t + \rho(r_j))^2 / n \right) \tag{3.2}$$

comparisons, for c_0 a constant.

To compute the expectations of expressions (3.1) and (3.2), we determine the following upper bounds.

LEMMA 3.1. *$E(\rho(r_j)) \leq t^2$ and $E((t + \rho(r_j))^2) \leq 2t^4$.*

Proof. If $S_j \cap X$ were infinite, then $\rho(r_j)$ would be the rank of the tth sample element in that set. Elements are in the sample independently with probability $1/t$; so we would have a Bernoulli process, and $\rho(r_j)$ would be given by the sum of t independent geometric random variables with parameter $1/t$. For a geometric random variable Z with parameter p, $E(Z) = 1/p$ so that $E(\rho(r_j)) \leq t^2$. Further, $E(Z^2) = (2-p)/p^2$ and therefore a simple calculation shows that $E((t + \rho(r_j))^2) \leq 2t^4$. □

Because of the concavity of logarithms, we thus have that the expected sum of expressions (3.1) and (3.2) is at most

$$2n \log((mt^2)/n) + c_0 n \log((2mt^4)/n) \leq cn \log(mt/n),$$

with $c = 4 + 4c_0$, since $m \geq 2n$.

Full analysis. Let $f(t, n, m)$ denote the average cost of **Large**$(t, X, \{S_j\})$, including recursive calls, when X and $\{S_j\}$ are the worst possible inputs such that $|X| = n$ and the number of subsets is m. We will show by induction on n that for $m \geq 2n$ and $t \geq 16$

$$f(t, n, m) \leq 2cn \log(mt/n). \tag{3.3}$$

This is true for $n \leq 16$: then $t \geq n$, while the sort of X takes at most $dn \log(n)$ for d a small constant.

So assume that $n > 16$. Then, examining all stages of the algorithm, we see that $f(t, n, m)$ satisfies the following equation:

$$f(t,n,m) \leq cn \log(mt/n) + E(f(t,|R|,m)) + E(f(mt^3/n,|R|,n)) + E(f(t,|Y|,m)),$$

where the expectations are taken over the distribution of the sample R.

By induction and because $n \log 1/n$ is concave, we may write:

$$\begin{aligned} f(t,n,m) \leq cn \log(mt/n) &+ f(t, E(|R|), m) \\ &+ f(mt^3/n, E(|R|), n) + f(t, E(|Y|), m). \end{aligned}$$

LEMMA 3.2. *$E(|R|) = n/t$ and $E(|Y|) \leq n/t$.*

Proof. The first part of the lemma is obvious. To prove the second part, we note that $x \in Y$ if and only if x is greater than at least mt^3/n values r_j for which $S_j \ni x$. Thus $(mt^3/n)|Y| \leq |\{(x,j) : x \geq r_j, x \in S_j\}| = \sum_j \rho(r_j)$. Hence $E(|Y|) \leq mt^2/(mt^3/n) = n/t$. □

The equation bounding f can now be formulated as

$$f(t,n,m) \leq cn \log(mt/n) + 2f(t, n/t, m) + f(mt^3/n, n/t, n). \tag{3.4}$$

Observe that if $m \geq 2n$ and $f(t', n', m')$ is one of the terms on the right-hand side of (3.4), then $t' \geq t$ and $m' \geq 2n'$.

Thus, by our inductive hypothesis (3.3),

$$\begin{aligned} f(t,n,m) &\leq cn \log(mt/n) + 2(2c(n/t)\log(mt^2/n)) + 2c(n/t)\log(mt^4/n) \\ &\leq cn \log(mt/n)(1 + 8/t + 8/t) \\ &\leq 2cn \log(mt/n), \end{aligned}$$

since $t \geq 16$ and $m \geq 2n$.

THEOREM 3.3. *Let m sets be given from a totally ordered n-element universe, with $m \geq 2n$, and a parameter $t \geq 16$. Then the expected number of comparisons required by the t-maxima algorithm is $O(n \log(mt/n))$. In particular, for all m we solve set-maxima in*

$$\Theta(n \log(m/n) + n)$$

expected comparisons, which is optimal as a function of n and m.

We established the upper bound above. Here is the lower bound. (Note that for $m \geq n$, $\Theta(\log(mt/n)) = \Theta(\log(mt^2/n))$.)

THEOREM 3.4. *If mt^2 is $O(n^2)$, then there exist collections of m sets such that*

$$\Omega(n \log(mt^2/n) + n)$$

comparisons are required for t-maxima.

Proof. If one set contains all elements, then $n-1$ comparisons are necessary. On the other hand, one may partition the universe into blocks of size B and arrange $\Theta(nB/t^2)$ sets of size t such that t-maxima forces each block to be sorted. This requires $\Omega(n \log B)$ comparisons. □

4. Open questions. We list below some questions we have encountered.

(1) What about deterministic algorithms? No nontrivial deterministic algorithm for local sorting is known other than on dense graphs, where one might as well totally sort the elements. The same is true of set-maxima with the exception of the restricted set systems discussed in [5] and [1].

(2) Can the time complexity of the set-maxima and local sorting algorithms (currently polynomial) be brought into line with their comparison model complexity?

(3) We have shown that there exist collections of m sets for which the information-theoretic lower bound for set-maxima matches the complexity of our algorithm. However, some collections of m sets have fewer arrangements of maxima. Is there a set-maxima algorithm that does better on such collections? We do not know of a simple characterization of the number of possible arrangements of maxima for an arbitrary collection of sets.

(4) A fourth question is raised by our local sorting algorithm, which makes comparisons between distant elements of the graph in order to finally determine the local order relationships. Is this necessary? Specifically: Is there a local sorting algorithm that uses $\Theta(\log \alpha(G))$ comparisons but never compares elements over a distance greater than some constant *comparison horizon* h? Our algorithm, modified so that $E(|R_k|) = n/\log n$, needs $h(n) = \sqrt{\log\log n}$. The question is particularly intriguing because our algorithm actually makes rather few long-distance comparisons (e.g., only linearly many beyond distance 4).

We know only that $h = 1$ is not sufficient. This answers a question of Linial [6]. Consider a bipartite graph where all values in one color class are greater than those in the other: all edges must be checked in order to certify this.

(5) A combinatorial question is that of obtaining better upper and lower bounds on $\alpha(G)$.

Acknowledgments. The authors would like to thank Michael Sipser for many discussions and considerable advice. Further, they would like to thank Michelangelo Grigni, Mark Haiman, Russell Impagliazzo, and Mauricio Karchmer for helpful discussions.

REFERENCES

[1] A. Bar-Noy, R. Motwani, and J. Naor, *A linear-time approach to the set maxima problem*, SIAM J. Discrete Math., 5 (1992), pp. 1–9.

[2] W. Goddard, V. King, and L. J. Schulman, *Optimal randomized algorithms for local sorting and set-maxima*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 45–53.

[3] R. Graham, F. Yao, and A. Yao, *Information bounds are weak in the shortest distance problem*, J. Assoc. Comput. Mach., 27 (1980), pp. 428–444.

[4] C. Kenyon and V. King, *Verifying partial orders*, in Proceedings of the 21st ACM Symposium on Theory of Computing, 1989, pp. 367–374.

[5] J. Komlós, *Linear verification for spanning trees*, Combinatorica, 5 (1985), pp. 57–65.

[6] N. Linial, *Legal coloring of graphs*, Combinatorica, 6 (1986), pp. 49–54.

[7] U. Manber and M. Tompa, *The effect of number of Hamiltonian paths on the complexity of a vertex-coloring problem*, SIAM J. Comput., 13 (1984), pp. 109–115.

[8] R. Stanley, *Acyclic orientations of graphs*, Discrete Math., 5 (1973), pp. 171–178.

[9] R. Tarjan, private communication.

[10] A. Yao, private communication.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 284–293, April 1993

A MONTE-CARLO ALGORITHM FOR ESTIMATING THE PERMANENT*

N. KARMARKAR[†], R. KARP[‡], R. LIPTON[§], L. LOVÁSZ[¶], AND M. LUBY[1]

Abstract. Let A be an $n \times n$ matrix with 0-1 valued entries, and let $\text{per}(A)$ be the permanent of A. This paper describes a Monte-Carlo algorithm that produces a "good in the relative sense" estimate of $\text{per}(A)$ and has running time $\text{poly}(n)2^{n/2}$, where $\text{poly}(n)$ denotes a function that grows polynomially with n.

Key words. permanent, matching, Monte-Carlo algorithm, algorithm, bipartite graph, determinant

AMS(MOS) subject classifications. 05C50, 05C70, 68Q25, 68R05, 68R10

1. Introduction. Let A be an $n \times n$ matrix with 0-1 valued entries, $\det(A)$ denote the determinant of A, and $\text{per}(A)$ denote the permanent of A. The marked contrast between the computational complexity of computing $\det(A)$ versus that of computing $\text{per}(A)$, despite the deceiving similarity between the two tasks, has baffled researchers for years. One of the reasons for interest in computing $\text{per}(A)$ is that A can be viewed as the adjacency matrix of a bipartite graph, $H = (X, Y, E)$, where X corresponds to the rows in A, Y to the columns in A, and $A_{ij} = 1$ if there is an edge between X_i and Y_j. The quantity $\text{per}(A)$ is exactly the number of perfect matchings in H.

It is well known that $\det(A)$ can be computed in $\text{poly}(n)$ time. On the other hand, the fastest algorithm known for computing $\text{per}(A)$ runs in $n2^n$ time [20]. Solid grounds for arguing that computing $\text{per}(A)$ is an inherently difficult problem were first provided in [21], which shows that the problem is $\#P$-complete. One implication of this result is that if $P \neq NP$, then there is no $\text{poly}(n)$ time algorithm for computing $\text{per}(A)$.

Because of the apparent nonexistence of a $\text{poly}(n)$ time algorithm for computing $\text{per}(A)$ exactly, we focus our attention on finding an algorithm that produces a good estimate of $\text{per}(A)$ and has a small running time. An (ϵ, δ) *approximation algorithm* for $\text{per}(A)$ is a Monte-Carlo algorithm that accepts as input A and two positive parameters ϵ and δ. The output of the algorithm is an estimate Y of $\text{per}(A)$, which satisfies

$$\Pr[(1-\epsilon)\text{per}(A) \leq Y \leq (1+\epsilon)\text{per}(A)] \geq 1 - \delta.$$

The papers [9], [15], [16] discuss (ϵ, δ) approximation algorithms for counting problems in greater detail. We develop an (ϵ, δ) approximation algorithm for $\text{per}(A)$, which runs in $2^{n/2}\frac{1}{\epsilon^2}\log(\frac{1}{\delta})\text{poly}(n)$ time. For fixed ϵ and δ, the running time of the approximation algorithm is essentially the square root of the running time for the fastest known algorithm that computes $\text{per}(A)$ exactly.

In [3], [10], [11] (ϵ, δ) approximation algorithms for $\text{per}(A)$ are given which run in $\text{poly}(n)$ time in the special case when each row and column in A contains at least $n/2$ $1's$. The report [13], which appeared during the final revision of this paper, gives an

*Received by the editors November 12, 1990; accepted for publication (in revised form) December 19, 1991.

[†]AT&T Bell Laboratories, Incorporated, Murray Hill, New Jersey 07974-2010.

[‡]Computer Science Division, University of California, Berkeley, California 94720, and International Computer Science Institute, Berkeley, California 94704-1105. The research of this author was supported by National Science Foundation grant CCR-9005448.

[§]Computer Science Department, Princeton University, Princeton, New Jersey 08544-0001.

[¶]Hungarian Academy of Sciences, 1051 Budapest, Roosevelt-Ter9, Hungary and Computer Science Department, Princeton University, Princeton, New Jersey 08544-0001.

[1]International Computer Science Institute, Berkeley, California 94720. The research of this author was partially supported by National Science Foundation operating grant CCR-9016468 and by grant number 89-00312 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

(ϵ, δ) approximation algorithm for per(A), which runs in time $O(c^{\sqrt{n}\log^2(n)}\epsilon^{-2}\log(\frac{1}{\delta}))$. Whether or not there is an (ϵ, δ) approximation algorithm for per(A), which runs in poly(n) time for general A, is still an open problem.

2. Some general considerations regarding (ϵ, δ) approximation algorithms. Suppose we would like to estimate some quantity Q and have available a stochastic experiment whose output is a random variable X such that $\mathrm{E}[X] = Q$ and $\mathrm{E}[X^2]$ is finite. Suppose further that we can repeat this experiment as many times as we wish, and that the outcomes of the successive trials will be independent and identically distributed, with the same distribution as X. Let X_i be the outcome of the ith trial. A straightforward application of Chebyshev's inequality shows that, if we conduct N trials, where $N = (\mathrm{E}[X^2]/\mathrm{E}[X]^2)(1/\epsilon^2\delta)$ then $(\sum_{i=1}^{N} X_i/N)$ gives an (ϵ, δ) approximation to Q.

We can improve the dependence of the number of trials on δ using a well-known trick. Setting $\delta = \frac{1}{4}$, we find that, if $N = (4\mathrm{E}[X^2]/\mathrm{E}[X]^2)(1/\epsilon^2)$, the probability is at least $\frac{3}{4}$ that the average of the N trials will lie within ϵ of Q. To obtain an (ϵ, δ)-approximation algorithm, we repeat such an N-sample experiment K times, where K is an odd integer greater than a suitable constant c times $\log(1/\delta)$, and take as our estimator of Q the median of the estimators produced by the K experiments of N samples each. Let us say that the outcome of a N-sample experiment is *good* if it lies within ϵ of Q. Then the median of the K outcomes will be good whenever the majority of the K outcomes are good. Using the fact that the outcomes are independent and identically distributed, and that each outcome has probability at least $\frac{3}{4}$ of being good, standard bounds on the tail of the binomial distribution [5] reveal that the median is good with probability at least $1 - \delta$. Thus, the number of trials required for an (ϵ, δ) approximation to Q is

$$O\left(\frac{\mathrm{E}[X^2]}{\mathrm{E}[X]^2}\frac{1}{\epsilon^2}\log\left(\frac{1}{\delta}\right)\right).$$

3. The Godsil/Gutman estimator of per(A). The discussion of the last section shows how an (ϵ, δ)-algorithm for approximating a quantity Q can be constructed from any computable stochastic experiment whose outcome is a random variable Y such that $\mathrm{E}[Y] = Q$ and $\mathrm{E}[Y^2]$ is finite. The efficiency of the algorithm will be based on the computational difficulty of performing the stochastic experiment, and on the ratio $\mathrm{E}[Y^2]/\mathrm{E}[Y]^2$. The rest of the paper is devoted to studying two particular stochastic experiments for estimating per(A). The first of these, which we call the Godsil/Gutman estimator, was suggested in [8], but no analysis of the number of trials needed for an (ϵ, δ) approximation algorithm was provided; the second one is a variant of the Godsil/Gutman estimator, which has a smaller second moment and thus leads to a more efficient algorithm.

The Godsil/Gutman estimator is defined as follows:

```
(1) An $n \times n$ matrix $B$ is formed from $A$ as follows:
        For all $i, j$, $1 \le i, j \le n$,
        If $A_{ij} = 0$ then $B_{ij} \leftarrow 0$
        Elseif $A_{ij} = 1$ then randomly and independently choose $B_{i,j} \in \{-1, 1\}$,
            each choice with probability $\frac{1}{2}$.
(2) $Y \leftarrow (\det(B))^2$.
```

This stochastic experiment can be executed in poly(n) time.

In §4 we introduce some technical language that is appropriate for all of the following analysis. Section 5 concerns the Godsil/Gutman estimator; we show that $\mathrm{E}[Y] =$ per(A) and derive an upper bound on $\mathrm{E}[Y^2]$. Then, in §6, we present a refinement of the Godsil/Gutman estimator, show that it is unbiased, and derive an upper bound on its second moment.

4. Terminology. Let P be the set of all $n!$ permutations of $1, \ldots, n$. For all $\sigma \in P$, let $\operatorname{sgn}(\sigma) = -1^t$, where t is the number of transpositions to form σ.

Let $P(A) \subseteq P$ be the set of all permutations σ such that for $i = 1, \ldots, n$, $A_{i,\sigma(i)} = 1$. Then, $\operatorname{per}(A) = \sum_{\sigma \in P(A)} 1 = |P(A)|$. For each $\sigma \in P(A)$, for $i = 1, \ldots, n$, we label $\langle i, \sigma(i) \rangle$ with the symbol σ. Let $P^2(A) = P(A) \times P(A)$. For each $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle \in P^2(A)$, let $G(\dot{\sigma})$ be the unlabelled graph where there is an unlabelled node for each $\langle i, j \rangle$ which is labelled with either or both of σ_1, σ_2, and where there is an edge between two distinct nodes $\langle i, j \rangle$ and $\langle i', j' \rangle$ if and only if $i = i'$ or $j = j'$. Each connected component of $G(\dot{\sigma})$ is an isolated node or an even length cycle. For each cycle in $G(\dot{\sigma})$, designate one of the nodes in the cycle as the root of the cycle. Let $D = \{G(\dot{\sigma}) : \dot{\sigma} \in P^2(A)\}$. For each graph $G \in D$, let $c(G)$ be the number of cycles in G. For each $G \in D$, let

$$\dot{\mathrm{eq}}(G) = \{\dot{\sigma} \in P^2(A) : G(\dot{\sigma}) = G\}.$$

PROPOSITION 4.1. $|\dot{\mathrm{eq}}(G)| = 2^{c(G)}$.

Proof. Let $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle \in \dot{\mathrm{eq}}(G)$. Each isolated node in G is labelled with both σ_1 and σ_2. Let c be a cycle in G. If the label of the root in c is σ_1, then every node at an even distance from the root in c must be labelled σ_1 and every node at an odd distance from the root in c must be labelled σ_2. The case when the root is labelled σ_2 is symmetric, interchanging the roles of σ_1 and σ_2. Thus, for each cycle there are two possible labellings and the total number of labellings of all cycles is then $2^{c(G)}$. □

Let $D' = \{G \in D : c(G) = 0\}$.

PROPOSITION 4.2.

(1) $G(\langle \sigma_1, \sigma_2 \rangle) \in D' \Leftrightarrow \sigma_1 = \sigma_2$.

(2) $G(\langle \sigma_1, \sigma_1 \rangle) = G(\langle \sigma_2, \sigma_2 \rangle) \Leftrightarrow \sigma_1 = \sigma_2$.

(3) $|D'| = \operatorname{per}(A)$.

(4) *For all* $G \in D'$, $|\dot{\mathrm{eq}}(G)| = 1$.

5. Analysis of the Godsil/Gutman estimator.

THEOREM 5.1. $\mathrm{E}[Y] = \operatorname{per}(A)$.

Proof. For each $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle \in P^2(A)$, let

$$x(\dot{\sigma}) = \prod_{k=1}^{2} \left(\operatorname{sgn}(\sigma_k) \prod_{i=1}^{n} B_{i,\sigma_k(i)} \right).$$

Then, since $\det(B) = \sum_{\sigma \in P(A)} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} B_{i,\sigma(i)}$,

$$Y = (\det(B))^2 = \sum_{\dot{\sigma} \in P^2(A)} x(\dot{\sigma}) = \sum_{G \in D} \sum_{\dot{\sigma} \in \dot{\mathrm{eq}}(G)} x(\dot{\sigma})$$

$$= \sum_{\dot{\sigma} = \langle \sigma_1, \sigma_1 \rangle \in P^2(A)} x(\dot{\sigma}) \qquad (1)$$

$$+ \sum_{G \in D - D'} \sum_{\dot{\sigma} \in \dot{\mathrm{eq}}(G)} x(\dot{\sigma}). \qquad (2)$$

For each $\dot{\sigma} = \langle \sigma_1, \sigma_1 \rangle \in P^2(A)$, $x(\dot{\sigma}) = 1$ independent of the values chosen for B. Thus, part (1) is equal to $\operatorname{per}(A)$ because the number of terms in the sum is $|P(A)| = \operatorname{per}(A)$. We show that the expected value of part (2) is equal to zero as follows. Fix $G \in D - D'$ and $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle \in \dot{\mathrm{eq}}(G)$. We show that $\mathrm{E}[x(\dot{\sigma})] = 0$, thus showing that the expected value of every term in part (2) is zero. Because $G \in D - D'$, G contains at least one cycle. Let $\langle i,j\rangle$ be some node in some cycle of G. Because either $\langle i,j\rangle$ is labelled with σ_1 and not with σ_2 or vice versa, $x(\dot\sigma)$ can be written as $y(\dot\sigma)B_{i,j}$, where $y(\dot\sigma)$ does not contain $B_{i,j}$. Because $B_{i,j}$ is independent of $y(\dot\sigma)$,

$$\mathrm{E}[x(\dot\sigma)] = \mathrm{E}[y(\dot\sigma)]\mathrm{E}[B_{i,j}].$$

Because $\mathrm{E}[B_{i,j}] = 0$, $\mathrm{E}[x(\dot\sigma)] = 0$. □

THEOREM 5.2.

$$\frac{\mathrm{E}[Y^2]}{\mathrm{E}[Y]^2} = \frac{\sum\limits_{G\in D} 6^{c(G)}}{\sum\limits_{G\in D} 2^{c(G)}}.$$

Proof.

$$\mathrm{E}[Y]^2 = (\mathrm{per}(A))^2 = \sum_{\langle\sigma_1,\sigma_2\rangle\in P^2(A)} 1 = \sum_{G\in D}\ \sum_{\langle\sigma_1,\sigma_2\rangle\in \dot{\mathrm{eq}}(G)} 1 = \sum_{G\in D} 2^{c(G)},$$

where the last equality is from Proposition 4.1. Let $P^4(A) = P(A) \times P(A) \times P(A) \times P(A)$. For each $\ddot\sigma = \langle\sigma_1,\sigma_2,\sigma_3,\sigma_4\rangle \in P^4(A)$, let

$$x(\ddot\sigma) = \prod_{k=1}^{4}\left(\mathrm{sgn}(\sigma_k)\prod_{i=1}^{n} B_{i,\sigma_k(i)}\right).$$

Then,

$$Y^2 = (\det(B))^4 = \sum_{\ddot\sigma\in P^4(A)} x(\ddot\sigma)$$

and

$$\mathrm{E}[Y^2] = \sum_{\ddot\sigma\in P^4(A)} \mathrm{E}[x(\ddot\sigma)].$$

Let ODD $= \{\ddot\sigma \in P^4(A)$: there is some $\langle i,j\rangle$ which is labelled with an odd number of labels from $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}\}$, and let EVEN $= P^4(A) -$ ODD. For each $\ddot\sigma \in$ ODD, there is some $\langle i,j\rangle$ such that $x(\ddot\sigma)$ can be written as $y(\ddot\sigma)B_{i,j}$, where $y(\ddot\sigma)$ does not contain $B_{i,j}$. Thus, $\mathrm{E}[x(\ddot\sigma)] = \mathrm{E}[y(\ddot\sigma)]\mathrm{E}[B_{i,j}]$. Because $\mathrm{E}[B_{i,j}] = 0$, $\mathrm{E}[x(\ddot\sigma)] = 0$. For each $\ddot\sigma \in$ EVEN, for each row $i = 1,\dots,n$, either there is a j such that $\langle i,j\rangle$ is labelled with all four of $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$ or there is a j and a $j' \neq j$ such that $\langle i,j\rangle$ is labelled with exactly two of $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$, and $\langle i,j'\rangle$ is labelled with the other two. Let $G(\ddot\sigma)$ be the graph where there is a node for each $\langle i,j\rangle$ such that $\langle i,j\rangle$ is labelled with at least one of $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$. There is an edge between two distinct nodes $\langle i,j\rangle$ and $\langle i',j'\rangle$ if and only if $i = i'$ or $j = j'$. Then, $\{G(\ddot\sigma) : \ddot\sigma \in \text{EVEN}\} = D$, where D is as previously defined. For each $G \in D$, define

$$\ddot{\mathrm{eq}}(G) = \{\ddot\sigma \in \text{EVEN} : G(\ddot\sigma) = G\}.$$

We claim that $|\ddot{\mathrm{eq}}(G)| = 6^{c(G)}$. The reasoning is similar to that used for the proof of Proposition 4.1. Let $\ddot\sigma = \langle\sigma_1,\sigma_2,\sigma_3,\sigma_4\rangle \in \ddot{\mathrm{eq}}(G)$. Each isolated node in G is labelled with all four of $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$. For each cycle c in G, the root of c and every node at an even distance from the root must be labelled with the same two elements of $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ and every node at an odd distance from the root must be labelled with the remaining two. Thus, there are $\binom{4}{2} = 6$ possible labellings of each cycle, and consequently a total of $6^{c(G)}$ labellings.

It is not hard to see that for each $\ddot{\sigma} \in$ EVEN, $x(\ddot{\sigma}) = 1$ independent of the values chosen for the entries in B. This fact rests on the following two observations:

• For any two permutations σ and τ, $\mathrm{sgn}(\sigma\tau) = \mathrm{sgn}(\sigma)\cdot \mathrm{sgn}(\tau)$ and $\mathrm{sgn}(\sigma^{-1}) = \mathrm{sgn}(\sigma)$. Hence $\prod_{k=1}^{4} \mathrm{sgn}(\sigma_k) = \mathrm{sgn}(\prod_{k=1}^{4} \sigma_k) = \prod_{k=2}^{4} \mathrm{sgn}(\sigma_1^{-1}\sigma_k)$. It follows from the definition of EVEN that, if each of the three permutations $\sigma_1^{-1}\sigma_k, k = 2, 3, 4$ is written in the usual cycle notation as a product of disjoint cycles, then the cycles occurring will correspond to the cycles of even length in $G(\ddot{\sigma})$, and each cycle will occur in exactly two of the three permutations $\sigma_1^{-1}\sigma_k$. Thus, if each of these cycles is expressed in a standard way as a product of transpositions, then each transposition will occur an even number of times. It follows that $\prod_{k=1}^{3} \mathrm{sgn}(\sigma_1^{-1}\sigma_k) = 1$, and hence $\prod_{k=1}^{4} \mathrm{sgn}(\sigma_k) = 1$;

• Each factor in the product$\prod_{k=1}^{4} \prod_{i=1}^{n} B_{i,\sigma_k(i)}$ occurs an even number of times, and thus the product is equal to 1.

Thus, $\mathrm{E}[Y^2] = \sum_{\ddot{\sigma}\in \mathrm{EVEN}} 1 = \sum_{G\in D} 6^{c(G)}$. Since $\mathrm{E}[Y]^2 = \sum_{G\in D} 2^{c(G)}$, the proof is complete. □

COROLLARY 5.3. *The Godsil/Gutman estimator yields an* (ϵ,δ)*- approximation algorithm for estimating* $\mathrm{per}(A)$ *which runs in time* $\mathrm{poly}(n)3^{n/2}\frac{1}{\epsilon^2}\log(\frac{1}{\delta})$.

Proof. Each evaluation of the estimator can be performed in time $\mathrm{poly}(n)$. Also,

$$\frac{\mathrm{E}[Y^2]}{\mathrm{E}[Y]^2} = \frac{\sum\limits_{G\in D} 6^{c(G)}}{\sum\limits_{G\in D} 2^{c(G)}} \leq \max_{G\in D} 3^{c(G)} \leq 3^{n/2},$$

where the previous inequality follows because there are at most $n/2$ cycles in any $G \in D$. □

6. A better estimator and its analysis.

We now present a variant of the Godsil/Gutman estimator which yields a more efficient (ϵ, δ)-approximation algorithm for $\mathrm{per}(A)$. Let

$$w_0 = 1, \quad w_1 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i, \quad w_2 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i$$

be the three cube roots of unity. If $y = a + bi$ is a complex number, then $\overline{y} = a - bi$ is the complex conjugate of y.

The estimator is computed as follows.

```
(1) An $n \times n$ matrix $B$ is formed from $A$ as follows:
    For all $i, j,\ 1 \leq i, j \leq n$,
    If $A_{i,j} = 0$ then $B_{i,j} \leftarrow 0$
    Elseif $A_{i,j} = 1$ then randomly and independently choose
        $B_{i,j} \in \{w_0, w_1, w_2\}$, each choice with probability $\frac{1}{3}$.
(2) $Z \leftarrow \det(B)\overline{\det(B)}$.
```

This estimator can be evaluated in $\mathrm{poly}(n)$ time.

THEOREM 6.1.

$$\mathrm{E}[Z] = \mathrm{per}(A).$$

Proof. The proof is similar to the proof of Theorem 5.1. For each $\dot\sigma = \langle \sigma_1, \sigma_2 \rangle \in P^2(A)$, let

$$x(\dot\sigma) = \mathrm{sgn}(\sigma_1)\mathrm{sgn}(\sigma_2) \prod_{i=1}^{n} B_{i,\sigma_1(i)} \overline{B}_{i,\sigma_2(i)}.$$

Then,

$$\begin{aligned} Z &= \det(B)\overline{\det(B)} = \sum_{\dot\sigma \in P^2(A)} x(\dot\sigma) \\ &= \sum_{\dot\sigma = \langle \sigma_1, \sigma_1 \rangle \in P^2(A)} x(\dot\sigma) && (3) \\ &\quad + \sum_{G \in D - D'} \sum_{\dot\sigma \in \dot{\mathrm{eq}}(G)} x(\dot\sigma). && (4) \end{aligned}$$

For each $\dot\sigma = \langle \sigma_1, \sigma_1 \rangle \in P^2(A)$, $x(\dot\sigma) = 1$ independent of the values chosen for B. Thus, part (3) is equal to per(A). Showing that the expected value of part (4) is equal to zero is very similar to the corresponding portion of the proof in Theorem 5.1. The observation needed is again that for any $\langle i, j \rangle$, $\mathrm{E}[B_{i,j}] = 0$. □

THEOREM 6.2.

$$\frac{\mathrm{E}[Z^2]}{\mathrm{E}[Z]^2} = \frac{\sum_{G \in D} 4^{c(G)}}{\sum_{G \in D} 2^{c(G)}}.$$

Proof. The proof follows exactly the outline of the proof of Theorem 5.2. We only note the differences here.

For each $\ddot\sigma = \langle \sigma_1, \sigma_2, \sigma_3, \sigma_4 \rangle \in P^4(A)$, let

$$x(\ddot\sigma) = \left(\prod_{k=1}^{2} \mathrm{sgn}(\sigma_k) \prod_{i=1}^{n} B_{i,\sigma_k(i)} \right) \left(\prod_{k=3}^{4} \mathrm{sgn}(\sigma_k) \prod_{i=1}^{n} \overline{B}_{i,\sigma_k(i)} \right).$$

Then, $\mathrm{E}[Z^2] = \sum_{\ddot\sigma \in P^4(A)} \mathrm{E}[x(\ddot\sigma)]$. From the definitions in the proof of Theorem 5.2,

$$P^4(A) = \mathrm{ODD} \cup \mathrm{EVEN} = \mathrm{ODD} \cup \bigcup_{G \in D} \ddot{eq}(G).$$

For $\ddot\sigma \in$ ODD there is some $\langle i, j \rangle$ such that $x(\ddot\sigma)$ can be written as $y(\ddot\sigma)B_{i,j}$ or as $y(\ddot\sigma)\overline{B}_{i,j}$, where in either case $y(\dot\sigma)$ contains neither $B_{i,j}$ nor $\overline{B}_{i,j}$. Because $\mathrm{E}[B_{i,j}] = \mathrm{E}[\overline{B}_{i,j}] = 0$, $\mathrm{E}[x(\ddot\sigma)] = 0$. For each $G \in D$ we further partition $\ddot{\mathrm{eq}}(G)$ as follows:

$$\begin{aligned} \ddot{\mathrm{eq}}'(G) = \ & \{\ddot\sigma \in \ddot{\mathrm{eq}}(G) : \text{in each cycle } c \text{ in } G \text{ the root of } c \text{ is} \\ & \text{labelled with exactly one of } \{\sigma_1, \sigma_2\} \text{ and with exactly} \\ & \text{one of } \{\sigma_3, \sigma_4\}\}. \end{aligned}$$

Because there are four possible labellings for each cycle in G, $|\ddot{\text{eq}}'(G)| = 4^{c(G)}$. For each $\ddot{\sigma} \in \ddot{\text{eq}}'(G)$, for each node $\langle i,j\rangle$ in G there are an equal number of occurrences of $B_{i,j}$ and $\overline{B}_{i,j}$ in $x(\ddot{\sigma})$. Thus, $x(\ddot{\sigma}) = 1$ independent of the values chosen for B, and $\mathrm{E}[x(\ddot{\sigma})] = 1$. For each $\ddot{\sigma} \in \ddot{\text{eq}}(G) - \ddot{\text{eq}}'(G)$, there is some node $\langle i,j\rangle$ in G such that $B_{i,j}$ occurs twice in $x(\ddot{\sigma})$ and $\overline{B}_{i,j}$ does not occur at all. Thus, $x(\ddot{\sigma})$ can be written as $y(\ddot{\sigma})B_{i,j}^2$, where $y(\ddot{\sigma})$ contains no occurrences of $B_{i,j}$ or $\overline{B}_{i,j}$. Then, $\mathrm{E}[x(\ddot{\sigma})] = \mathrm{E}[y(\ddot{\sigma})]\mathrm{E}[B_{i,j}^2]$. Because $\mathrm{E}[B_{i,j}^2] = 0$, $\mathrm{E}[x(\ddot{\sigma})] = 0$.

Putting this together, $\mathrm{E}[Z^2] = \sum_{G\in D} 4^{c(G)}$, and thus

$$\frac{\mathrm{E}[Z^2]}{\mathrm{E}[Z]^2} = \frac{\sum_{G\in D} 4^{c(G)}}{\sum_{G\in D} 2^{c(G)}}. \qquad \square$$

COROLLARY 6.3. *The estimator Z yields an (ϵ,δ)-approximation algorithm for estimating* $\mathrm{per}(A)$ *which runs in time* $\mathrm{poly}(n)2^{n/2}\frac{1}{\epsilon^2}\log(\frac{1}{\delta})$.

Proof. Each evaluation of the estimator can be performed in time $\mathrm{poly}(n)$. Also,

$$\frac{\mathrm{E}[Z^2]}{\mathrm{E}[Z]^2} = \frac{\sum_{G\in D} 4^{c(G)}}{\sum_{G\in D} 2^{c(G)}} \le \max_{G\in D} 2^{c(G)} \le 2^{n/2},$$

where the previous inequality follows because there are at most $n/2$ cycles in any $G \in D$. $\square$

It is natural to consider a generalization of the above two algorithms, in which $B_{i,j}$ is set equal to zero when $A_{i,j} = 0$, and to a random kth root of unity whenever $A_{i,j} = 1$. The Godsil/Gutman estimator corresponds to the case $k = 2$, and the algorithm of the present section, to the case $k = 3$. Theorem 3 holds for all integers $k \ge 2$, and Theorem 4 holds for all integers $k \ge 3$. However, choosing k greater than 3 appears to give no reduction in the variance of the estimator.

7. Some special cases.

We have introduced two unbiased estimators of $\mathrm{per}(A)$: the Godsil/Gutman estimator Y and a second estimator Z which refines the Godsil/Gutman technique by using cube roots of unity. We showed that

$$\frac{\mathrm{E}[Y^2]}{\mathrm{E}[Y]^2} = \frac{\sum_{G\in D} 6^{c(G)}}{\sum_{G\in D} 2^{c(G)}}$$

and

$$\frac{\mathrm{E}[Z^2]}{\mathrm{E}[Z]^2} = \frac{\sum_{G\in D} 4^{c(G)}}{\sum_{G\in D} 2^{c(G)}}.$$

We then obtained an upper bound of $3^{n/2}$ on $\mathrm{E}[Y^2]/\mathrm{E}[Y]^2$ and an upper bound of $2^{n/2}$ on $\mathrm{E}[Z^2]/\mathrm{E}[Z]^2$ using the trivial observation that, for all G, $c(G) \le \frac{n}{2}$. Although this bound seems terribly pessimistic, the following example shows that there are cases in which it is close to the truth. Let n be even and let A be the $n \times n$ matrix such that for $i = 1, \dots, \frac{n}{2}$, $A_{2i-1,2i-1} = A_{2i-1,2i} = A_{2i,2i-1} = A_{2i,2i} = 1$ and all other entries in A are zero. For the first algorithm the expected number of trials before there is even one trial where $Y \neq 0$ is $\Omega(2^{n/2})$, and for the second algorithm it is $\Omega((\frac{3}{2})^{n/2})$. There is a lot of room between $3^{n/2}$ and $2^{n/2}$ and between $(\frac{3}{2})^{n/2}$ and $2^{n/2}$; we are not sure which bound is closer to the worst case behavior for the respective algorithms.

Despite this bad example, we suspect that in most situations, these bounds are far too pessimistic, since $c(G)$ will "typically" be much smaller than $n/2$. As one heuristic indication of this phenomenon we note that, if $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle$, where σ_1 and σ_2 are independent random permutations, then $c(G(\dot{\sigma}))$ will be close to $\ln(n)$ with very high probability.

In this section we analyze a concrete example in which the bounds of $3^{n/2}$ for the Godsil/Gutman algorithm and $2^{n/2}$ for its refinement are provably too pessimistic: the $n \times n$ matrix C in which every element is equal to 1. The permanent of this matrix is, of course, $n!$.

Let the random variable Y be the estimator produced by the Godsil/Gutman algorithm applied to the matrix C, and let Z be the estimator produced by the refinement given in §6 applied to the matrix C. Our main result is as follows.

THEOREM 7.1. *In the particular case of the $n \times n$ matrix C,*

$$\frac{\mathrm{E}[Y^2]}{\mathrm{E}[Y]^2} \leq \frac{(n+1)(n+2)}{2} \quad \textit{and} \quad \frac{\mathrm{E}[Z^2]}{\mathrm{E}[Z]^2} \leq n+1.$$

Proof. Since the two estimators are unbiased, $\mathrm{E}[Y] = \mathrm{E}[Z] = n!$. In order to discuss the second moments of Y and Z we require a definition: for any permutation σ of $\{1, 2, \dots, n\}$, let $d(\sigma)$ be the number of cycles of length greater than one in the permutation σ.

We know from Theorems 5.2 and 6.2 that $\mathrm{E}[Y^2] = \sum_G 6^{c(G)}$ and $\mathrm{E}[Z^2] = \sum_G 4^{c(G)}$. We note that, for any $\dot{\sigma} = \langle \sigma_1, \sigma_2 \rangle$, $c(G(\dot{\sigma})) = d(\sigma_1^{-1}\sigma_2)$. Combining this with the fact that $|\dot{\mathrm{eq}}(G)| = 2^{c(G)}$ we find that $\sum_{G \in D} 6^{c(G)} = \sum_{\langle \sigma_1, \sigma_2 \rangle \in P^2(A)} 3^{d(\sigma_1^{-1}\sigma_2)}$. In the case of the complete graph, $P(A)$ is equal to P, the set of all permutations of $\{1, 2, \dots, n\}$, and thus $\mathrm{E}[Y^2] = \sum_{\langle \sigma_1, \sigma_2 \rangle \in P^2} 3^{d(\sigma_1^{-1}\sigma_2)}$. Since each permutation σ can be written as $\sigma_1^{-1}\sigma_2$ in exactly $n!$ ways, the right-hand side reduces to $n! \sum_{\sigma \in P} 3^{d(\sigma)}$. Similarly, we obtain $\mathrm{E}[Z^2] = n! \sum_{\sigma \in P} 2^{d(\sigma)}$. But it follows from [17] (solution to Exercise 3.12, p. 203) that $\sum_{\sigma \in P} 3^{d(\sigma)} \leq \frac{(n+2)!}{2}$ and $\sum_{\sigma \in P} 2^{d(\sigma)} \leq (n+1)!$. Hence, $\mathrm{E}[Y^2]/\mathrm{E}[Y]^2 \leq (n+2)(n+1)/2$ and $\mathrm{E}[Z^2]/\mathrm{E}[Z]^2 \leq n+1$. $\square$

It follows that, in the special case of the matrix C, a quadratic number of trials of the Godsil/Gutman Monte-Carlo algorithm, or a linear number of trials of our refinement of the Godsil/Gutman algorithm, are sufficient for an (ϵ, δ)-approximation.

In a preliminary version of this paper we conjectured that similar claims hold true for almost all $n \times n$ zero-one matrices. This was established in [9], where it was shown that, for almost every 0, 1 matrix, $O(n\omega(n)\epsilon^{-2})$ independent trials, using the estimator of §6, suffice to obtain an $(\epsilon, \frac{1}{4})$ approximation to the permanent. Here $\omega(n)$ is any function tending to infinity as $n \to \infty$. It was previously known that [12], combined with [7] or [19], yields a polynomial-time (ϵ, δ) approximation algorithm for the permanent of a random zero-one matrix of arbitrary density. However, Jerrum's result, based on the algorithm of §6, gives a better running time than these methods.

8. Comments, generalizations, refinements. For both algorithms presented here, once the values of B have been chosen, the value of the estimator can be computed exactly in $\mathrm{poly}(n)$ time. In the second algorithm, this requires that $\sqrt{3}$ be represented symbolically. The symbol $\sqrt{3}$ will not appear in the final answer, which is an integer.

Both Monte-Carlo algorithms can easily be modified to estimate $\mathrm{per}(A)$ when the entries in A are allowed to be arbitrary positive numbers. However, in this case there is an issue with precision; the estimators cannot be computed exactly in $\mathrm{poly}(n)$ time, although they can be closely approximated. The modification for the Godsil/Gutman estimator is to randomly choose

$$B_{i,j} \in \left\{\sqrt{A_{i,j}}, -\sqrt{A_{i,j}}\right\},$$

each choice with probability $\frac{1}{2}$. The modification for the second estimator is to randomly choose

$$B_{i,j} \in \left\{\sqrt{A_{i,j}}w_0, \sqrt{A_{i,j}}w_1, \sqrt{A_{i,j}}w_2\right\},$$

each choice with probability $\frac{1}{3}$. In both cases, the expected value of the estimator is $\mathrm{per}(A)$ and the upper bounds on the number of trials to guarantee an (ϵ, δ) approximation algorithm stated in Corollaries 5.3 and 6.3 still apply, assuming that at each trial the estimator is computed exactly. The analysis would have to be modified to allow for truncation error.

Each of our Monte-Carlo algorithms consists of $O(\log(\frac{1}{\delta}))$ phases, with each phase consisting of some number N of independent trials. We required that each phase be an $(\epsilon, \frac{1}{4})$ approximation algorithm. Our analysis of the upper bound on the number of trials to guarantee this property was based on Chebyshev's inequality, and thus the analysis still holds if trials are pairwise independent. Consider running the first Monte-Carlo algorithm with a fixed ϵ and δ. As it is written it requires $\Theta(3^{\frac{n}{2}}n^2)$ random bits per phase in total, i.e., n^2 random bits per trial to randomly choose the values for B. This can be reduced to $O(n^3)$ random bits per phase using standard methods of generating pairwise independent unbiased random bits [1], [6], [18].

9. Open questions.

(1) Is there an (ϵ, δ) approximation algorithm for $\mathrm{per}(A)$ which runs in $\mathrm{poly}(n)$ time? One possible approach to solving this problem is based on the observation that the trials within a phase need only be pairwise independent, rather than completely independent. The result of each phase is $\sum_{i=1}^{\ell} Y_i/\ell$ where ℓ, the number of trials, is exponential in n. Pairwise independence permits the ℓ samples to be generated using only $O(n^3)$ random bits. Perhaps the rule for generating the samples from the random bits can be designed so that the quantity $\sum_{i=1}^{\ell} Y_i$ can be computed directly and efficiently from the random bits, without the need to calculate the result of each trial explicitly. Similar ideas have been used successfully in other contexts [1], [2], [6], [4], [14], [18].

(2) Is there a deterministic algorithm with running time $o(2^n)$, which accepts as input A and ϵ and which outputs Y such that

$$(1-\epsilon)\,\mathrm{per}(A) \leq Y \leq (1+\epsilon)\,\mathrm{per}(A)?$$

REFERENCES

[1] W. ALEXI, B. CHOR, O. GOLDREICH, AND C. P. SCHNORR, RSA *Rabin functions: certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[2] E. BACH, *Realistic analysis of some randomized algorithms*, in 19th Proceedings of the ACM Symposium on the Theory of Computing, 1987, pp. 453–461.

[3] A. BRODER, *How hard is it to marry at random (on the approximation of the permanent)*, in 19th Proceedings of the ACM Symposium on the Theory of Computing, 1986, pp. 50–58.

[4] J. CARTER AND M. WEGMAN, *Universal class of hash functions*, J. Comput. Systems Sci., 18 (1979), pp. 143–154.

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

[6] B. CHOR AND O. GOLDREICH, *On the power of two-points based sampling*, J. Complexity, to appear.

[7] A. M. FRIEZE, *A note on computing random permanents*, manuscript, 1989.

[8] C. D. GODSIL AND I. GUTMAN, *On the matching polynomial of a graph*, Algebraic Methods in Graph Theory, I, L. Lovász and V. T. Sós, eds., Math. Soc. János Bolyai, North-Holland, Amsterdam, 1981, pp. 241–249.

[9] M. JERRUM, *An analysis of a Monte-Carlo algorithm for estimating the permanent*, Tech. Report ECS-LFCS-91-164, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, Edinburgh, Scotland, June, 1991.

[10] M. JERRUM, L. L. VALIANT, AND U. VAZIRANI, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[11] M. JERRUM AND A. SINCLAIR, *Conductance and the rapid mixing property for Markov chains: the approximation of the permanent resolved*, Proceedings of the ACM Symposium on the Theory of Computing, 1988, pp. 235–243.

[12] ———, *Approximating the permanent*, Internal Report CSR-275-88, Department of Computer Science, University of Edinburgh, Edinburgh, Scotland, 1991; SIAM J. Comput., 18 (1989), pp. 1149–1178.

[13] M. JERRUM AND U. VAZIRANI, *A mildly exponential approximation algorithm for the permanent*, Internal Report ECS-LFCS-91-179, Department of Computer Science, University of Edinburgh, Scotland, 1991.

[14] H. KARLOFF AND P. RAGHAVAN, *Randomized algorithms and pseudorandom numbers*, in Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 310–321.

[15] R. KARP AND M. LUBY, *Monte-Carlo algorithms for enumeration and reliability problems*, in 24th Proceedings of the IEEE Foundation of Computer Science, 1983, pp. 56–64.

[16] R. KARP, M. LUBY, AND N. MADRAS, *Monte-Carlo approximation algorithms for enumeration problems*, J. Algorithms, 10 (1989), pp. 429–448.

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

[18] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[19] R. MOTWANI, *Expanding graphs and the average-case analysis of algorithms for matchings and related problems*, in Proceedings of the ACM Symposium on the Theory of Computing, 1989, pp. 550–561.

[20] H. RYSER, *Combinatorial Mathematics*, The Carus Mathematical Monographs, 14, the Mathematical Association of America, Washington, DC, 1963.

[21] L. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 294–302, April 1993

LIMITING NEGATIONS IN CONSTANT DEPTH CIRCUITS*

MIKLOS SANTHA† AND CHRISTOPHER WILSON‡

Abstract. It follows from a theorem of Markov that the minimum number of negation gates in a circuit sufficient to compute any Boolean function on n variables is $l = \lfloor \log n \rfloor + 1$. It can be shown that, for functions computed by families of polynomial size, $O(\log n)$ depth and bounded fan-in circuits (NC^1), the same result holds: on such circuits l negations are necessary and sufficient. In this paper it is proven that this situation changes when polynomial size circuit families of constant depth are considered: l negations are no longer sufficient. For threshold circuits it is proven that there are Boolean functions computable in constant depth (TC^0) such that no such threshold circuit containing $o(n^\epsilon)$, for all $\epsilon > 0$, negations can compute them. There is a matching upper bound: for any $\epsilon > 0$, everything computable by constant depth threshold circuits can be computed by constant depth threshold circuits using n^ϵ negations asymptotically. There are also tight bounds for constant depth, unbounded fan-in circuits (AC^0): $n/\log^r n$, for any r, negations are sufficient, and $\Omega(n/\log^r n)$, for some r, are necessary.

Key words. circuit complexity, monotone circuits, negation gates

AMS(MOS) subject classifications. 68Q15, 68Q10, 94C10

1. Introduction. Although extensively studied, not very much is known about the circuit complexity of Boolean functions. The results are especially few concerning lower bounds. While it is conjectured that NP-complete problems cannot be computed with circuits of less than exponential size, the best-known lower bounds are linear with small constants. In striking opposition to this situation, important progress has been made recently on monotone circuits. In his famous result Razborov [15] has proved a super-polynomial lower bound on the monotone circuit complexity of an appropriate clique function. Later this lower bound was strengthened to exponential size by Alon and Boppana [5]. In another development, Tardos [16] pointed out that there are even problems in P whose monotone circuit complexity is exponential, thus proving that negation may be exponentially powerful.

Of course, we would like to extend Razborov's lower bound result to the general model. As this seems to be at the moment quite elusive, a natural intermediate step is the study of circuits with a limited number of negations. If negations are also permitted in the circuit, then we should not restrict the study just to monotone functions. But if we consider also nonmonotone functions in our investigations, then before the study of lower bounds there is an even more basic question: can a given function be computed at all with a limited number of negations?

This question was answered by Markov [12] without any complexity theoretical considerations. He defined for any Boolean function $f : \{0,1\}^n \to \{0,1\}^m$, the *inversion* complexity $inv(f)$ of f as the minimum number of negation gates contained in a circuit which computes f. Let $f = (f_1, \dots, f_m)$, and let $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ be two Boolean vectors in $\{0,1\}^n$. By definition let $x \le y$ if $x_i \le y_i$ for $i = 1, \dots, n$. The ordered pair (x, y) is a *gap* for f if $x < y$ and for some j, $1 \le j \le m$, $f_j(x) > f_j(y)$. Let $x_1 < \cdots < x_r$ be an increasing sequences of Boolean vectors in $\{0,1\}^n$. The *decrease* of f on the sequence $x_1, \dots, x_r$ is the number of indices i such that (x_i, x_{i+1}) is a gap for f. Finally the *decrease* $dec(f)$ of f is the maximum decrease over all increasing sequence

*Received by the editors April 19, 1991; accepted for publication (in revised form) December 16, 1991.

†Centre National de Recherche Scientifique, Unité de Recherche Associée 410, Université Paris-Sud, Laboratoire de Recherche en Informatique, 91405 Orsay, France.

‡Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403. The research of this author was supported by National Science Foundation grant CCR-8810051.

of Boolean vectors. The result of Markov establishes a precise relationship between the inversion complexity of f and its decrease.

THEOREM 1.1 (MARKOV'S THEOREM) [12]. *For every Boolean function f we have*

$$inv(f) = \lfloor \log(dec(f)) \rfloor + 1.$$

Because the length of any increasing sequence of n-dimensional Boolean vectors is at most $n + 1$, by Markov's theorem $\lfloor \log n \rfloor + 1$ negations are sufficient to compute any Boolean function on n variables. On the other hand, it is easy to find a very simple function f for which there exists an increasing sequence of vectors with n gaps. Thus, for some functions $\lfloor \log n \rfloor + 1$ negations are also necessary.

In this paper we will study what remains true of this necessary and sufficient condition when restrictions are imposed on the size and depth of the circuits computing f. The restriction we will impose on the size of the circuits is polynomial size. Thus, the question we would like to answer is the following: Let $f : \{0,1\}^* \rightarrow \{0,1\}^*$ be a Boolean function that can be computed by a family of polynomial size circuits. Is it true that f can also be computed with a family of polynomial size circuits that contain at most $\lfloor \log n \rfloor + 1$ negations?

It turns out that the answer strongly depends on whether any restriction is put on the depth of the circuits. If f can be computed in polynomial size and depth $d(n)$, where $d(n) = \Omega(\log n)$, then the answer is *yes* (Theorem 2.3): f can be computed in the same order of depth with $\lfloor \log n \rfloor + 1$ negations, even if the underlying model has bounded fan-in. This result is implicitly contained in an early survey paper of Fischer [7], where he also considers circuits with limited negation.

Our results on the other hand show that the answer is *no* for constant depth circuits. In the case of threshold circuits we show that there exists a function computable in constant depth which cannot be computed in constant depth on threshold circuits using $o(n^\epsilon)$, for all $\epsilon > 0$, negations (Corollary 3.3). We also establish a matching upper bound on the number of negations sufficient for constant depth (Theorem 3.7): For any $\epsilon > 0$, every function which can be computed in constant depth on a family of threshold circuits can be computed in constant depth by threshold circuits with n^ϵ negations asymptotically. The method will give us a sublinear bound on the number of negations needed for AC^0 circuits (Theorem 4.1): For any r, every Boolean function computable in constant depth can also be computed in constant depth with at most $n/\log^r n$ negations. This is the best bound we can obtain (Corollary 4.4): There is a (multi-output) function computable in constant depth that cannot be computed in constant depth with $o(n/\log^r n)$, for all $r > 0$, negotiations.

The tight lower bounds of Corollary 3.3 and Corollary 4.4 are obtained from trade-off results between depth and number of negations in constant depth. Theorem 3.2 says that depth d threshold circuits for NEG (see the definition in §2) require $d(n+1)^{1/d} - d$ negations, and Theorem 4.3 claims that any circuit family computing NEG in depth d has $\Omega(n/\log^{d+3} n)$ negations. We can also prove that depth d threshold circuits for *PARITY* have $d(\lceil n/2 \rceil)^{1/d} - d$ negations. However, we are not able to obtain a tight lower bound on the number of negotiations required by a constant depth *AND/OR* circuit for a *single valued* function (*NEG* has n outputs). See §5 for more comments on this problem.

Let us mention at this point a result of Okolnishnikova [13] and Ajtai and Gurevich [2] related to our Theorem 3.2: There exists a monotone function that can be computed with polynomial size, constant depth circuits, but can not be computed with monotone, polynomial size, constant depth circuits.

A trade-off result between depth and randomness was obtained for the st-connectivity problem by Raz and Wigderson [14]. They have shown that in every polynomial size,

$O(\log n)$ depth circuit family, in which all the negations are placed on the input variables, at least a constant fraction of them should be negated.

The paper is organized as follows. After some preliminaries, §2 contains a short outline of the proof of the already known result about circuits with $\Omega(\log n)$ depth. Section 3 deals with upper and lower bounds for constant depth threshold circuits. Section 4 derives upper and lower bounds for unbounded fan-in *AND/OR* circuits. Finally, in §5 we conclude and mention some open problems.

2. Preliminaries. We will use standard notions from circuit complexity theory, for which the reader is referred, e.g., to Wegener's book [18]. We will also use some conventions throughout the paper. When it is not otherwise specified, we will deal with circuits on n variables. Let x denote the Boolean vector $(x_1, \dots, x_n)$, and $x - x_i$ the vector $(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. We often identify the vector x with the word $x_1 \cdots x_n$. For $w \in \{0,1\}^*$, the *weight* of w is the number of ones in w, denoted by $|w|$. If f is a one-output Boolean function, then $\bar{f}$ is the negation of f.

Throughout the paper there may be nonintegral indices which should, in fact, be rounded to an appropriate integer. This has not been done for the sake of legibility and is in any case straightforward.

Two circuits are *equivalent* if they compute the same function. A circuit is *monotone* if it does not contain any negation gates. Ignoring uniformity considerations, for $i \geq 0$, the classes NC^i and AC^i are defined to be the set of functions computable by polynomial size, $O(\log^i n)$ depth circuit families with bounded and unbounded fan-in, respectively.

Another important class of circuits we examine is that of threshold circuits. By definition, for $k = 0, \dots, n$, the kth *threshold* function $T_k(x) = 1$ if and only if $|x| \geq k$. The class TC^i is defined to be the class of functions computed by a family of polynomial size, $O(\log^i n)$ depth circuits consisting of negations and gates that compute threshold functions. It is known that $NC^i \subseteq AC^i \subseteq TC^i \subseteq NC^{i+1}$. An especially interesting problem recently has been that of separating these classes when $i = 0$. It is known that $AC^0 \subset TC^0$ [9], but whether the containment $TC^0 \subseteq NC^1$ is proper is still an open problem. In [8] it is shown that depth 2 threshold circuits are weaker than depth 3 threshold circuits. In [20] it is shown that depth k monotone threshold circuits are weaker than depth $k+1$ monotone threshold circuits for any k.

The *sorting* function $S(x)$ and the *exact* function $E(x)$ are closely related to threshold functions. By definition $S(x) = (T_1(x), \dots, T_n(x))$, and $E(x) = (E_0(x), \dots, E_n(x))$, where $E_k(x) = 1$ if and only if $|x| = k$. Indeed, $S(x)$ is the simultaneous computation of all the nontrivial threshold functions, $E_n(x) = T_n(x)$, and $E_k(x) = T_k(x) \wedge \bar{T}_{k+1}(x)$ for $0 \leq k \leq n-1$. These functions will be extensively used as well as the function NEG, defined by $NEG(x_1, \dots, x_n) = (\bar{x}_1, \dots, \bar{x}_n)$.

Let $\mathcal{C}$ be the class of functions computed by a class of families of polynomial size circuits, and let $g(n)$ be a function from the natural numbers to the natural numbers. Then $\mathcal{C}_{g(n)}$ is the set of functions that can be computed by a circuit family in the class that contains at most $g(n)$ negation gates. The class $\mathcal{C}_0$ will be denoted *mon-$\mathcal{C}$*; this is the set of functions computable by a monotone circuit family in the class. By the *type* of a circuit we mean bounded fan-in, unbounded fan-in, or threshold. The following facts are well known.

FACT 2.1 (Valiant [17], Ajtai, Komlós, and Szemerédi [3]). *The sorting function $S(x)$ is in mon-NC^1.*

FACT 2.2 (Ajtai and Ben-Or [1], Wegener, Wurm, and Yi [19]). *For every $t \geq 0$, the threshold function $T_{\log^t n}(x)$ is in mon-AC^0.*

The importance of the function NEG lies in the fact that it incorporates all the "non-monotone" information we need to compute any function by a circuit. This is expressed in the following completeness lemma.

LEMMA 2.1 (Completeness Lemma). *Let C be a class of functions computed by families of polynomial size circuits of the types described above, such that the allowable depth of the circuit families is closed under multiplication by a constant. Let $g(n)$ be a function on the natural numbers. Then we have*

$$C_{g(n)} = C \quad \textit{if and only if } NEG \in C_{g(n)}.$$

Proof. The implication is straightforward from left to right. The other direction is implied by the following well-known result (see, e.g., Wegener [18]): For every circuit C of size s and depth d, there exists a monotone circuit C' of the same type, of size at most $2s$ and depth d which is equivalent to C, when the output of $NEG(x)$ is also given as input to C'. □

The function NEG can easily be computed by a monotone AC^0 circuit from the outputs of the sorting function and the exact functions. This is stated in the following lemma, which shares some ideas with the theory of slice functions (again, see [18]).

LEMMA 2.2. *For $1 \leq i \leq n$, we have*

$$\bar{x}_i = \bigvee_{k=0}^{n} (T_k(x - x_i) \ \wedge \ E_k(x)).$$

Proof. First we claim that for any $1 \leq i \leq n$, $\bar{x}_i = T_{|x|}(x - x_i)$. If $\bar{x}_i = 1$, then the number of ones in $x - x_i$ is $|x|$ and $T_{|x|}(x - x_i) = 1$. On the other hand, if $\bar{x}_i = 0$, then the number of ones in $x - x_i$ is $|x| - 1$ and $T_{|x|}(x - x_i) = 0$. The result follows since $E_k(x) = 1$ if and only if $k = |x|$. □

Fischer [7] constructed a circuit that contains only $\lfloor \log n \rfloor + 1$ negations, and computes the exact function, when the inputs are already sorted. The size of the circuit is polynomial, and its depth is $O(\log n)$. This enabled him to compute the exact functions by a polynomial size circuit family containing $\lfloor \log n \rfloor + 1$ negations. The depth of his circuit family depended on the circuit depth of the sorting function, which was at that time still an open problem. Today, it is known (Fact 2.1) that threshold functions can in fact be computed in *mon-*NC^1. This means that Fischer's result implicitly implies the following theorem.

THEOREM 2.3. *For every circuit family of polynomial size and depth $d(n)$, there exists an equivalent circuit family of the same type, also of polynomial size and depth $d(n) + O(\log n)$, which contains only $\lfloor \log n \rfloor + 1$ negations.*

COROLLARY 2.4. *For all $k \geq 1$, we have*

(i) $NC^k_{\lfloor \log n \rfloor + 1} = NC^k$,

(ii) $AC^k_{\lfloor \log n \rfloor + 1} = AC^k$,

(iii) $TC^k_{\lfloor \log n \rfloor + 1} = TC^k$.

This method of computing NEG in $NC^1_{\lfloor \log n \rfloor + 1}$ can be viewed as a constructive (and efficient) implementation of Markov's result. This, to some extent, was foreshadowed by Akers [4]. An analysis of his method reveals that NEG can be computed using few negotiations in $O(\log n)$ depth using threshold circuits; that is, $TC^1_{\lfloor \log n \rfloor + 1}$.

3. Bounds for threshold circuits.

3.1. Lower bounds. Here we shall prove that it is impossible to compute *NEG* on a depth d threshold circuit that uses fewer than $d(n+1)^{1/d} - d$ negotiations. In the proof of this lower bound we will concentrate on inputs that are integers in unary notation. These inputs are sequences of n bits with the ones preceding the zeros. When we say that j is the input value, we mean that $1^j 0^{n-j}$ is the input string.

With each gate g in a circuit we associate a satisfying set $I_g \subseteq \{0, \dots, n\}$ such that gate g outputs 1 if and only if input $j \in I_g$. For example, the satisfying set of x_i is $[i, n]$, that of $\bar{x}_5 \vee x_{10}$ is $[0, 5) \cup [10, n]$, and that of $x_5 \wedge \bar{x}_{10}$ is $[5, 10)$.

Let $I \subseteq \{0, \dots, n\}$. We define j as a *right boundary* of I if $j \in I$ and $j + 1 \notin I$. The value j is a right boundary of a gate of a circuit if it is a right boundary of the satisfying set of the gate, and j is a right boundary of a circuit if it is the right boundary of some of its gates. For example, the unique right boundary of $\bar{x}_i$ is $i - 1$. In what follows, it is important to count the number of right boundaries of a circuit since these are the inputs where it behaves nonmonotonically. First we will note that the only way to create new right boundaries is by the use of negotiations.

LEMMA 3.1. *Let C be a circuit of any type whose gates are either negations or monotone functions. Suppose that gate g is a monotone function of its input gates. Then any right boundary of g is a right boundary of at least one of its input gates.*

Proof. Let us suppose that $g = f(g_1, \dots, g_r)$, where f is a monotone function and $g_1, \dots, g_r$ are the input gates to g. We claim that if j is not a right boundary of g_i, for $1 \le i \le r$, then j is not a right boundary for g either. This is true since $g_i(j) \le g_i(j+1)$, for $1 \le i \le r$, implies $g(j) \le g(j+1)$ by the monotonicity of f. □

THEOREM 3.2. *Let C be a circuit computing NEG on inputs of size n. Suppose that C has depth d, uses ν negations, and has gates that are either negations or arbitrary monotone functions. Then $\nu \ge d(n+1)^{1/d} - d$.*

Proof. For $0 \le i \le d$, let level i of the circuit consist of all gates whose longest path to an input is of length i. Level 0 consists of inputs and constants, and thus only presents the single right boundary n. The circuit must eventually create n other right boundaries. Our bound will follow by showing that added depth can create only few right boundaries if insufficient negotiations are available.

Observe what happens when a node is negated. If gate g has k right boundaries and satisfying set $[i_1, i_2) \cup [i_3, i_4) \cup \cdots \cup [i_{2k-1}, i_{2k})$, then $\bar{g}$ has satisfying set $[0, i_1) \cup [i_2, i_3) \cup \cdots \cup [i_{2k}, n]$. This creates up to k new right boundaries. By the previous lemma, no other type of gate can create new right boundaries. Thus, if up to some level the gates present altogether t right boundaries and at the next level μ gates are negated, this next level creates at most $t\mu$ new right boundaries. This gives a total of $t(1+\mu)$ possible right boundaries up to the next level.

For $i = 1, \dots d$ let ν_i be the number of negations at level i, where $\sum_{i=1}^{d} \nu_i = \nu$. By the above, the circuit can create at most $\prod_{i=1}^{d}(1 + \nu_i)$ right boundaries. This product is maximized when $\nu_i = \nu/d$. Since we must have $(1 + \nu/d)^d \ge n + 1$, it follows that $\nu \ge d(n+1)^{1/d} - d$. □

Similarly we can prove that if C is a circuit computing *PARITY* that satisfies the conditions of Theorem 3.2, then $\nu \ge d(\lceil n/2 \rceil)^{1/d} - d$. As threshold gates are monotone, the following corollary is immediate.

COROLLARY 3.3. *Let $g(n) = o(n^\epsilon)$ for all $\epsilon > 0$. Then*

1. $NEG,\ PARITY \notin TC^0_{g(n)}$;
2. $TC^0 \neq TC^0_{g(n)}$.

Notice that these results did not put any restrictions on the size of the circuit. Even exponentially many monotone gates are of little use without enough negations. We can state something strong about depth as well. For example, no family of threshold circuits of depth $(\log n)^{1-\epsilon}$, $\epsilon > 0$, with $2^{\log^\epsilon n}$ negations can compute *NEG* or *PARITY*. Furthermore, Corollary 3.3 immediately implies that $NC^1_{\text{poly-log}}$ properly contains $TC^0_{\text{poly-log}}$.

COROLLARY 3.4. *Let $g(n) \geq \log n$ and for all $\epsilon > 0, g(n) = o(n^\epsilon)$. Then $TC^0_{g(n)} \neq NC^1_{g(n)}$.*

It is interesting to compare Corollary 3.4 to the result in [20] showing that *mon-TC^0* is properly contained in *mon-NC^1*. Corollary 3.4 can also be generalized to show a separation of NC^1 and TC^0 restricted to, say, $\log\log n$ negations. This generalization does not subsume the result in [20].

THEOREM 3.5. *Let $0 < f(n) \leq \lfloor \log n \rfloor + 1$. If for all $\epsilon > 0$, $g(n) = o(2^{\epsilon f(n)})$, then*

$$NC^1_{f(n)} - TC^0_{g(n)} \neq \emptyset.$$

Proof. Let $k = n/2^{f(n)-1}$ and consider the function

$$e(x) = (x_0 \wedge \bar{x}_k) \vee (x_{2k} \wedge \bar{x}_{3k}) \vee \cdots \vee (x_{n-k} \wedge \bar{x}_n)$$

(x_0 by default is 1). This can be viewed as a function on $\frac{n}{k} = 2^{f(n)-1}$ inputs, so by Theorem 2.3 is in $NC^1_{f(n)}$. Following the proof of Theorem 3.2, $e(x)$ has $\frac{n}{2k}$ right boundaries. Thus, any depth d threshold circuit with ν negations computing $e(x)$ must have $(1 + \nu/d)^d \geq \frac{n}{2k} = 2^{f(n)-2}$ or $\nu \geq d2^{(f(n)-2)/d} - d$. This is not possible under the assumption on $\nu = g(n)$ in the statement of the theorem. □

3.2. Upper bounds. We have seen that not everything computable in TC^0 can be computed using $o(n^\epsilon)$, for all $\epsilon > 0$, negations. The question arises naturally: how many negations are sufficient to give full power to TC^0? We can show that the lower bounds derived above for threshold circuits are essentially optimal.

Our main tool will be the computation of the exact function in constant depth on a threshold circuit. Since this is just two levels away from *NEG*, it is evident that we cannot compute it in depth $d-2$ using less than $d(n+1)^{1/d} - d$ negations. We will show how to compute it in depth $3d + O(1)$ using no more $dn^{1/d} - d + 1$ negations. Hence, the upper and lower bounds are nearly tight.

LEMMA 3.6. *Let $d \geq 1$ be an integer. There exists a depth $3d+O(1)$ and size $O(n^{1+(1/d)})$ family of threshold circuits with $dn^{1/d} - d + 1$ negations computing the exact function E.*

Proof. Set $x_0 = 1$ for the sake of convenience. We will assume that the input has been sorted as $x_1 \geq x_2 \geq \cdots \geq x_n$. This can be done in depth 1 on a threshold circuit, and this is the only place that we need threshold gates. The rest of the circuit will consist of negotiations and unbounded fan-in *AND*/*OR* gates.

The circuit we describe will have d layers, each layer will consist of several levels. Let us define the functions F_i^k for $0 \leq k \leq d$ and $0 \leq i < n^{k/d}$:

$$F_i^k = 1 \Leftrightarrow in^{(d-k)/d} \leq |x| < (i+1)n^{(d-k)/d}.$$

The construction we describe is iterative. One layer of the circuit will use $n^{1/d} - 1$ negations in transforming the $\{F_i^k\}_{i=0}^{n^{k/d}-1}$ into the $\{F_i^{k+1}\}_{i=0}^{n^{(k+1)/d}-1}$. Clearly, the F_i^d are the desired outputs E_i, so long as we add $E_n = x_n$. The base case is simple as there is only one possible i: $i = 0$. We let $F_0^0 = \bar{x}_n$.

Given the F_i^k for $0 \leq i < n^{k/d}$, the following describes how to construct the F_i^{k+1} for $0 \leq i < n^{(k+1)/d}$ in constant depth.

(k,1) For $0 \leq i < n^{k/d}$ and $0 \leq j < n^{1/d}$ compute

$$G^k_{i,j} = F^k_i \wedge x_{in^{(d-k)/d}+jn^{(d-k-1)/d}}.$$

(k,2) For $0 \leq j < n^{1/d}$ compute $H^k_j = \bigvee_{i=0}^{n^{k/d}-1} G^k_{i,j}$.
(k,3) For $1 \leq j < n^{1/d}$ compute $\bar{H}^k_j$.
(k,4) For $0 \leq j \leq n^{1/d} - 2$ compute $I^k_j = H^k_j \wedge \bar{H}^k_{j+1}$. Let $I^k_{n^{1/d}-1} = H^k_{n^{1/d}-1}$.
(k,5) For $0 \leq i < n^{k/d}$ and $0 \leq j < n^{1/d}$ compute $F^{k+1}_{in^{1/d}+j} = I^k_j \wedge G^k_{i,0}$.
We can describe when the functions above are satisfied.
(1) $G^k_{i,j} = 1$ if and only if $in^{(d-k)/d} + jn^{(d-k-1)/d} \leq |x| < (i+1)n^{(d-k)/d}$.
(2) $H^k_j = 1$ if and only if there exists i $(1 \leq i < n^{k/d})$ $in^{(d-k)/d} + jn^{(d-k-1)/d} \leq |x| < (i+1)n^{(d-k)/d}$.
(3) $\bar{H}^k_j = 1$ if and only if there exists i $(1 \leq i < n^{k/d})$ $in^{(d-k)/d} \leq |x| < in^{(d-k)/d} + jn^{(d-k-1)/d}$.
(4) $I^k_j = 1$ if and only if there exists i $(1 \leq i < n^{k/d})$ $in^{(d-k)/d} + jn^{(d-k-1)/d} \leq |x| < in^{(d-k)/d} + (j+1)n^{(d-k-1)/d}$.
(5) $F^{k+1}_{in^{1/d}+j} = 1$ if and only if $(in^{1/d} + j)n^{(d-k-1)/d} \leq |x| < (in^{1/d} + j + 1)n^{(d-k-1)/d}$, as desired.

The steps (k,4), (k,5), and ($k+1$,1) are all computed by $\wedge$ gates, so they can be combined into one level. This yields a circuit for the $E_i = F^d_i$ of depth $3d + O(1)$. The number of negations needed is $1 + d(n^{1/d} - 1)$. The size bound follows from observing that steps (k,1) and (k,5) use $n^{k/d}n^{1/d} = n^{(k+1)/d}$ gates. The sum $\sum_{k=1}^d n^{(k+1)/d}$ is clearly $O(n^{(d+1)/d})$. □

THEOREM 3.7. *For every $\epsilon > 0$, we have asymptotically*

$$TC^0_{n^\epsilon} = TC^0.$$

Proof. Choose d so that $1/d < \epsilon$. For this d we have that $dn^{1/d} \leq n^\epsilon$ asymptotically. From Lemma 3.6 we can compute E in constant depth using asymptotically less than n^ϵ negotiations. From Lemma 2.2, we see that $\bar{x}_i = \bigvee_{k=0}^n (T_k(x - x_i) \wedge E_k(x))$. The Completeness Lemma then implies the result. □

4. Constant depth AND/OR circuits. In this section we show that AC^0 remains invariant under a restriction to some sublinear number of negations: $n/\log^r n$ negations, for any r, are sufficient. Furthermore, this bound is tight.

The upper bound follows from the construction used in the previous section.

THEOREM 4.1. *Let $r \geq 0$ and $g(n) = n/\log^r n$. Then we have*

$$AC^0_{g(n)} = AC^0.$$

Proof. We will break the input up into n/N groups, each of size $N = 4\log^{2r} n$. According to Fact 2.2, T_N is in *mon-AC^0*. Thus we can sort any group $y_1, \ldots, y_N$ in monotone constant depth and polynomial (in n) size.

As we have seen in the proof of Lemma 3.6, we can compute E from $y_1 \geq \cdots \geq y_N$ in constant depth using *AND/OR* gates and $2\sqrt{N}$ negations. The thresholds can be applied again to find $\bar{y}_i = \bigvee_{k=0}^N (T_k(y - y_i) \wedge E_k(y))$, as per Lemma 2.2. The total number of negations used will be $2\sqrt{N}(n/N) = n/\log^r n$. The result then follows from the Completeness Lemma. □

DEFINITION. The *sensitivity on* a string w of the single valued function f, $s(f, w)$, is the number of neighbors w' of w differing in exactly one bit such that $f(w) \neq f(w')$. The *sensitivity* of f, $s(f)$, is the average over $w \in \{0,1\}^n$ of $s(f, w)$.

There have been several papers in recently relating the sensitivity of a Boolean function to its Fourier transform [10], [11]. To show a matching lower bound to Theorem 4.1, we make use of the following application of these results to AC^0 functions.

LEMMA 4.2 [11]. *If f is computed by a circuit family of depth d, then* $s(f) = 0(\log^{d+3} n)$.

THEOREM 4.3. *Let $\{C_n\}$ be a depth d circuit family which computes $NEG(x)$ with $\nu(n)$ negotiations. Then $\nu(n) = \Omega(n/\log^{d+3} n)$.*

Proof. Let us suppose on the contrary that there is depth d circuit family $\{C_n\}$ which computes $NEG(x)$ with $\nu(n) \neq \Omega(n/\log^{d+3} n)$ negotiations. By Lemma 4.2 there exists a constant $c > 0$ such that for every n, for every function f which is computed at some gate of C_n,

$$s(f) < c \log^{d+3} n.$$

Our hypothesis implies that there exists n such that

$$\nu(n) < n/(c \log^{d+3} n).$$

Let n be such an integer, and let $f_1, \dots, f_k$, $k \leq \nu(n)$ be the outputs of the negation gates of C_n.

Given a string w, we say a bit of w is sensitive to f if changing that bit changes the output of f on w. Otherwise, that bit is insensitive to f. It follows that there is a string w, which has a bit insensitive to all $f_1, \dots, f_k$. This is because

$$\begin{aligned} E[s(f_1, w) + \cdots + s(f_k, w)] &= E[s(f_1, w)] + \cdots + E[s(f_k, w)] \\ &\leq k c \log^{d+3} n < n, \end{aligned}$$

where E indicates the expectation of an event over all strings w of length n uniformly distributed.

Let w be a string whose jth bit is insensitive to all $f_1, \dots, f_k$. Obtain w' by changing this bit. We can suppose that $w_j = 0$ and $w_j{}' = 1$, which implies $w < w'$. Since the bit is insensitive, we have for all i, $f_i(w) = f_i(w')$. Thus, between w and w' the outputs of all negation gates of C_n are constant. This implies that no gate of C_n can take a greater value on w than on w'. However, C_n computes $\bar{x}_j$, so on input w it outputs $\bar{w}_j = 1$ and on w' it outputs $\bar{w}'_j = 0$. This is impossible if no negation gate changes. □

COROLLARY 4.4. *If $g(n) = o(n/\log^r n)$ for all r, then $AC^0 \neq AC^0_{g(n)}$.*

5. Conclusions and open problems. Let us say a few words about the uniformity of our circuits. In fact all of them are uniform. Indeed, while the earlier circuits of Valiant [17] and Ajtai and Ben-Or [1] of, respectively, Fact 2.1 and Fact 2.2 are probabilistic, the later constructions of Ajtai, Komlós and Szemerédi [3] and Wegener, Wurm, and Yi [19] are already uniform. The circuits of Fact 2.2 can also be constructed uniformly using Theorem 4.1 of [6].

Although the upper bound of Theorem 4.1 and the lower bound of Theorem 4.3 were matching, the lower bound held only for a multivalued function. An intriguing problem is to find out the exact number of negations necessary and sufficient to compute every single-valued function in AC^0. We can show a weaker lower bound for some single valued function. Let the problem *EXISTODD*, defined on $x_1, \dots, x_n$, be

$$(x_1 \wedge \bar{x}_2) \vee (x_3 \wedge \bar{x}_4) \vee \cdots \vee (x_{n-1} \wedge \bar{x}_n)$$

(if n is odd, then the formula will be $\cdots \vee (x_{n-2} \wedge \bar{x}_{n-1}) \vee x_n$). Since *EXISTODD* on inputs of the form $1^j 0^{n-j}$ is equivalent to *PARITY*, the proof technique of Theorem 3.2 works for this function. Clearly, *EXISTODD* is in AC^0 but it is not in $AC^0_{g(n)}$ if, for all $\epsilon > 0$, $g(n) = o(^\epsilon)$.

Another direction is where we started our reasoning: can we find lower bounds with limited negations, without any restriction on the depth? For example, can we prove Razborov type results for circuits containing a few (say, a constant number of) negations?

Acknowledgments. The authors would like to thank Noam Nisan for pointing out the use of Lemma 4.2 in deriving the lower bound for AC^0.

REFERENCES

[1] M. Ajtai and M. Ben-Or, *A theorem on probabilistic constant depth circuits*, in Proceedings of the 16th ACM Association for Computing Machinery Symposium on the Theory of Computing, ACM Press, 1984, pp. 471–474.

[2] M. Ajtai and Y. Gurevich, *Monotone versus positive*, J. Assoc. Comput. Mach., 34 (1987), pp. 1004–1015.

[3] M. Ajtai, J. Komlós, and E. Szemerédi, *An $O(n \log n)$ sorting network*, Combinatorica, 3 (1983), pp. 1–19.

[4] S. Akers, *On maximum inversion with minimum inverters*, IEEE Trans. Comput., 17 (1968), pp. 134–135.

[5] N. Alon and R. Boppana, *The monotone circuit complexity of Boolean functions*, Combinatorica, 7 (1987), pp. 1–22.

[6] R. Boppana, *Threshold functions and bounded depth monotone circuits*, J. Comput. System Sci., 32 (1986), pp. 222–229.

[7] M. Fischer, *The complexity of negation-limited networks — a brief survey*, in Automata Theory and Formal Languages, 2nd GI Conference, LNCS, Vol. 33, H. Brakhage, ed., Springer-Verlag, New York, 1975, pp. 71–82.

[8] A. Hajnal, P. P. W. Maass, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, in Proceedings of the 28th IEEE Foundations of Computer Science, 1987, pp. 99–110.

[9] J. Hastad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th Association for Computing Machinery Symposium on the Theory of Computing , 1986, pp. 6–20.

[10] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proceedings of the 29th IEEE Foundations of Computer Science, 1988, pp. 68–80.

[11] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform and learnability*, in Proceedings of the 30th IEEE Foundations of Computer Science, pp. 574–579.

[12] A. Markov, *On the inversion complexity of systems of Boolean functions*, Soviet Math. Dokl., 4 (1963), pp. 694–696.

[13] E. A. Okolnishnikova, *On the influence of negation on the complexity of a realization of monotone Boolean functions by formulas of bounded depth*, Metody Diskret. Analiz., 38 (1982), pp. 74–80.

[14] R. Raz and A. Wigderson, *Probabilistic communication complexity of Boolean relations*, in Proceedings of the 30th IEEE Foundations of Computer Science, 1989, pp. 562–567.

[15] A. A. Razborov, *Lower bounds on the monotone complexity of some Boolean functions*, Soviet Math. Dokl., 281 (1985), pp. 798–801.

[16] E. Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica, 7 (1987), pp. 393–394.

[17] L. Valiant, *Short monotone formulae for the majority function*, J. Algorithms, 5 (1984), pp. 363–366.

[18] I. Wegener, *The complexity of Boolean functions*, Wiley-Teubner, Stuttgart, 1987.

[19] I. Wegener, N. Wurm, and S. Z. Yi, *Symmetric functions in AC^0 can be computed in constant depth with a very small size*, in Boolean Function Complexity, Selected papers from the LMS Durham Symposium, M. Paterson, ed., Cambridge Univ. Press, London, 1992.

[20] A. Yao, *Circuits and local computation*, in Proceedings of 20th Association for Computing Machinery Symposium on the Theory of Computing, 1989, pp. 186–196.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 303–317, April 1993

AVERAGE TIME ANALYSIS OF CLAUSE ORDER BACKTRACKING*

KHALED M. BUGRARA† AND PAUL WALTON PURDOM, JR.‡

Abstract. Backtracking algorithms solve problems by selecting a variable and assigning each possible value to the variable. The resulting subproblems are simplified and solved recursively. Simple backtracking selects variables in a fixed order. Clause order backtracking selects variables from the first nontrivial clause that has not yet been satisfied. Formulas are given for the average time used by clause order backtracking when solving random CNF satisfiability problems, where the problem sets have v variables, t clauses, and a probability p of a literal being in a clause. The average time for clause order backtracking is always less than that for simple backtracking. It leads to polynomial time under many conditions where simple backtracking uses exponential average time. Cases where clause order backtracking uses average time less than v^n (in the limit of v going to infinity) include $p < 1/(2v)e^{[(n-1)\ln v - \ln t]/t}$ and $p > \sqrt{[\ln t + (\ln v)/2]/v}$. (The second result needs a slight increase in the coefficient of $\ln t$ when t increases faster than $v^{\ln v}$.)

Key words. average time, backtracking, combinatorial search, Davis–Putnam, pure literal rule, NP-complete, satisfiability, searching

AMS(MOS) subject classifications. 68P10, 68Q20, 68Q25, 68T15

1. Constraint satisfaction and backtracking.

Many interesting problems require determining whether a set of constraints on variables with discrete values can be satisfied. Let $R_1(x_1, \ldots, x_v), \ldots, R_t(x_1, \ldots, x_v)$ be a set of relations and $x_1, \ldots, x_v$ be a set of variables, where each variable has a finite set of possible values. A constraint satisfaction problem consists of determining whether the variables can be set in a way that makes all of the relations *true*. Such problems can be quite difficult even when each relation is simple. For example, if each relation is a clause, then the constraint satisfaction problem becomes the classical Conjunctive Normal Form (CNF) satisfiability problem. Many special forms of the constraint satisfaction problem are NP-complete [3], [6], [12].

Searching is one common way to solve constraint satisfaction problems. The basic idea of searching is to choose a variable and generate subproblems by assigning each possible value to the variable. In each subproblem the relations are simplified by plugging in the value of the selected variable. If any subproblem has a solution, then the original problem has a solution. Otherwise, the original problem has no solution. The subproblems are solved by applying the technique recursively. *Simple search algorithms* stop the recursion when all variables have values. They use exponential time when used to find all solutions.

If any relation of a constraint satisfaction problem is always *false*, then the problem has no solution. *Backtracking* improves over plain search by immediately reporting no solution for problems with a false relation. Often this short cut saves a huge amount of time. Backtracking can take either exponential or polynomial average time, depending on the set of problems being solved [1], [11].

Simple backtracking has a fixed order for selecting variables. Sometimes it wastes time by assigning values to variables that do not appear in the problem. Even when the original problem uses all the variables, some of the simplified subproblems may use

*Received by the editors July 16, 1990; accepted for publication December 21, 1991.

†College of Computer Science, Northeastern University, 360 Huntington Avenue, Boston, Massachusetts 02115.

‡Computer Science Department, Indiana University, Lindley Hall, Bloomington, Indiana 47405–4101. The work of this author was supported by Indiana University, and FAW (Research Institute for Applied Knowledge Processing at the University of Ulm).

only a few of them. For problems with a few short clauses simple backtracking *frequently* assigns values to absent variables.

Clause order backtracking reports no solution if the problem has a relation that is always *false*. Otherwise, it selects the first relation that is not always *true*. For the first variable that affects the value of this relation, each possible value is plugged in, the predicate simplified, and the resulting subproblem is solved by recursive application of the algorithm. Each solution of the subproblem (along with the the partial assignments of values that lead to the subproblem) gives a solution to the original problem. If the original problem has no nontrivial relations, then every assignment of values to the remaining variables results in a solution.

The following is a precise statement of the version of the algorithm that we analyze. This version is tailored for CNF satisfiability problems. The algorithm finds every solution to the given CNF problem, but it reports the solutions in a compressed form. A clause is always *false* if and only if it is empty (contains no literals). A clause is a *tautology* if and only if it contains a variable and its negation. A tautological clause always evaluates to *true*.

Clause Order Backtracking Algorithm for CNF problems.

1. If the CNF problem has an empty clause, return with an empty set of solutions, and charge one time unit.
2. If the first remaining clause of the CNF problem is a tautology, then remove it from the problem. Repeat this part of the step as long as it applies. If there are no clauses, then return with the current assignment of values to the variables as a solution (each assignment of values for the remaining unset variables results in a solution) and charge one time unit.
3. Let k be the number of unset variables in the first remaining clause. (Step 1 ensures that $k \geq 1$, and Step 2 ensures that each variable occurs in at most one literal of the clause.) For j starting at 1 and increasing to at most k, generate the jth subproblem by setting the first $j-1$ variables of the clause so that their literals are *false* and setting the jth variable so that its literal is *true*. Use the assignment of values to simplify the CNF problem (remove each false literal from its clause and remove from the problem each clause with a true literal). Apply the algorithm recursively to solve the simplified problem. If setting the first $j-1$ literals of the first remaining clause to *false* results in some clause being empty, then stop generating subproblems. The set of solutions for the original problem is the union of the set of solutions for the subproblems. If the loop stops with $j = h$, then charge $h + 1$ time units.

The cost in time units has been defined to be the same as the number of nodes in the backtrack tree generated by the algorithm. The actual running time of the algorithm depends on how cleverly it is implemented, but a good implementation will result in a time that is proportional to the number of nodes multiplied by a factor that is between 1 and tv, where v is the number of variables and t is the number of clauses.

The backtrack tree includes nodes for determining that the first remaining clause is empty. The computation associated with these nodes can be done quickly. In the analysis we briefly consider the effect of not charging for these nodes (giving an upper limit of k time units for Step 3). A cost of up to $k + 1$ units at Step 3 is natural when comparing the algorithm with simple backtracking, but a limit of k units is more natural when comparing the algorithm with unit clause backtracking. (Unit clause backtracking selects variables from clauses of length one when possible.)

Figure 1 illustrates the counting of nodes. Each internal node is labelled with the variable that is set at the corresponding point in the computation, the left branch corresponds to setting the variable to *false*, and the right branch corresponds to setting it to *true*. The first clause, $c_1 = x \vee y$, contributes nodes 1, 2, and 3. The second clause, $c_2 = x \vee z$, contributes nodes 4 and 5. Node 6 represents the solution $\{x = \textit{false}, y = \textit{true}, z = \textit{true}\}$. When x is *true*, both the first and second clauses are satisfied, but the third clause, $c_3 = \bar{x} \vee \bar{z}$ contributes nodes 7 and 9. Node 8 represents the set of solutions with $\{x = \textit{true}, z = \textit{false}\}$. All values of y are permitted at node 8. If the predicate had a fourth clause, $c_4 = x \vee \bar{y}$, then the predicate would be *false* at node 4 and clause c_2 would contribute only node 4. Nodes 5 and 6 would not be in the tree. With the three clause problem of Fig. 1, nodes 3, 5, and 9 would not be charged for using the alternate charging method of the previous paragraph.

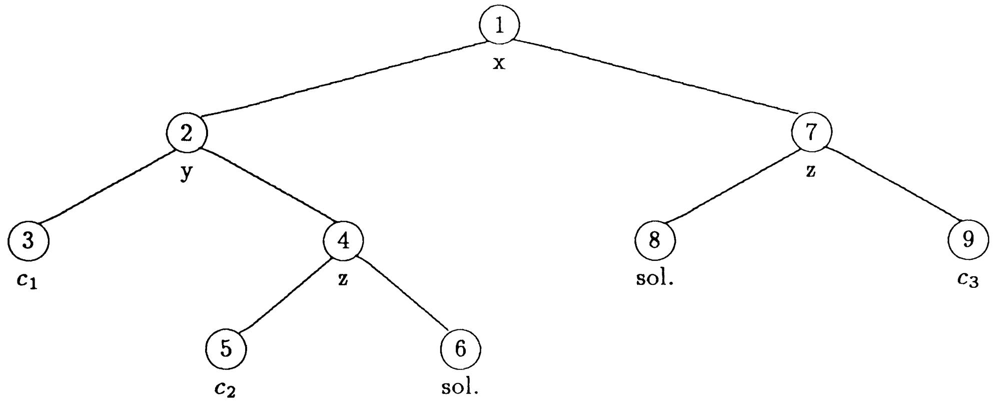


FIG. 1. *The backtrack tree for the predicate consisting of the three clauses* $c_1 = x \vee y$, $c_2 = x \vee z$, $c_3 = \bar{x} \vee \bar{z}$.

2. Probability model. The *random clause model* generates random CNF satisfiability problems using a set of v variables. A random clause is formed by independently selecting each of the $2v$ literals with probability p. A random predicate is formed by independently forming t random clauses.

In this model some variables may not occur in a particular problem. Some clauses may be empty. Some clauses may be tautologies. Some clauses may be duplicates of others. Each of these features may be considered to be a defect in the model, since problems initially given to a satisfiability algorithm are not likely to have them. However, these defects are not as important as it may at first appear because most of the features do occur in the subproblems that are produced by searching algorithms. The model has the advantage that the distribution of clauses does not change much when setting a variable. This leads to a relatively easy analysis. As a result more satisfiability algorithms have been analyzed with this model than with any other. The problems generated by this model can be either easy or hard (for the algorithms that have been analyzed so far) depending on how the parameters are set [2], [4], [5], [7], [8], [9], [10], [11]. The model is useful for comparing strengths and weaknesses of various satisfiability algorithms.

3. Discussion of results. This section discusses the the average time performance of clause order backtracking and compares it with that of other satisfiability algorithms. The following sections contain the detailed derivation of the results.

A contour plot of the performance for problems with one hundred variables is given in Fig. 2. A similar plot for fifty variables is given in [9]. The line that runs from the left side to the right side shows for each t, the value of p that leads to the largest average number of nodes. The remaining contours denote constant running time and are as follow. The outer contour in Fig. 2 shows the conditions where the average number of nodes per problem is 100. Each contour inward shows the conditions where the average number of nodes per problem is a factor of 100 larger. Thus, the contours show the conditions where the average number of nodes per problem is v, v^2, v^3, and v^4. The actual average running time is bounded by a low degree polynomial times the average number of nodes, where the details of the polynomial depend on just how cleverly one programs the algorithm.

The average number of nodes for clause order backtracking is never more than the average number of nodes for the version of simple backtracking that reports every solution. Under many conditions, clause order backtracking is much faster, but when pv is small and t/v is large the improvement is not important. (This is the lower right region of Fig. 2.) The comparison with simple backtracking [11] implies that the average number of nodes for clause order backtracking is bounded by v^n for large v when

$$(1) \qquad \frac{t}{v} > \frac{\ln 2 - n(\ln v)/v}{-\ln(1-e^{-pv})} \quad \text{and} \quad pv < \ln 2,$$

and also when

$$(2) \qquad \frac{t}{v} > \frac{pt[2\ln 2 - n(\ln v)/v]}{(\ln 2)\ln(1+pt/\ln 2) + pt\ln[1+(\ln 2)/pt]} \quad \text{and} \quad pv > \ln 2,$$

provided $pv/\ln v$ goes to zero.

There are additional cases where clause order backtracking is fast. Under most of these conditions the average number of solutions per problem is exponential. The reason that clause order backtracking is able to run in polynomial time and report every solution is that solutions are reported in compressed form. When the algorithm reports a solution, it often does not assign values to all the variables. Any truth assignment to the remaining variables is a solution.

For small p we show that the average number of nodes is no more than v^n when

$$(3) \qquad p \le \frac{1}{2v} e^{[(n-1)\ln v - \ln t - O(1)]/t}.$$

This limit approaches $1/(2v)$ when t grows faster than a constant times $\ln v$. This result corresponds to the fact that in Fig. 2 the contours all remain above $p = 1/(2v)$. The result in (3) is better than the one in (1) when

$$(4) \qquad \frac{t}{v} \le 0.743 < \frac{\ln 2}{-\ln(1-e^{-1/2})}.$$

For large p we show that the average number of nodes is no more than v^n when

$$(5) \qquad p \ge \sqrt{\frac{(1+\delta)\ln t + (\ln v)/2}{v}},$$

for any $\delta > 0$ and large v. When $\ln[(\ln t)/(\ln v)] < \ln v$ (i.e., when $t < v^{\ln v}$), δ can be replaced with zero. This result corresponds to the upper branch of the contours in

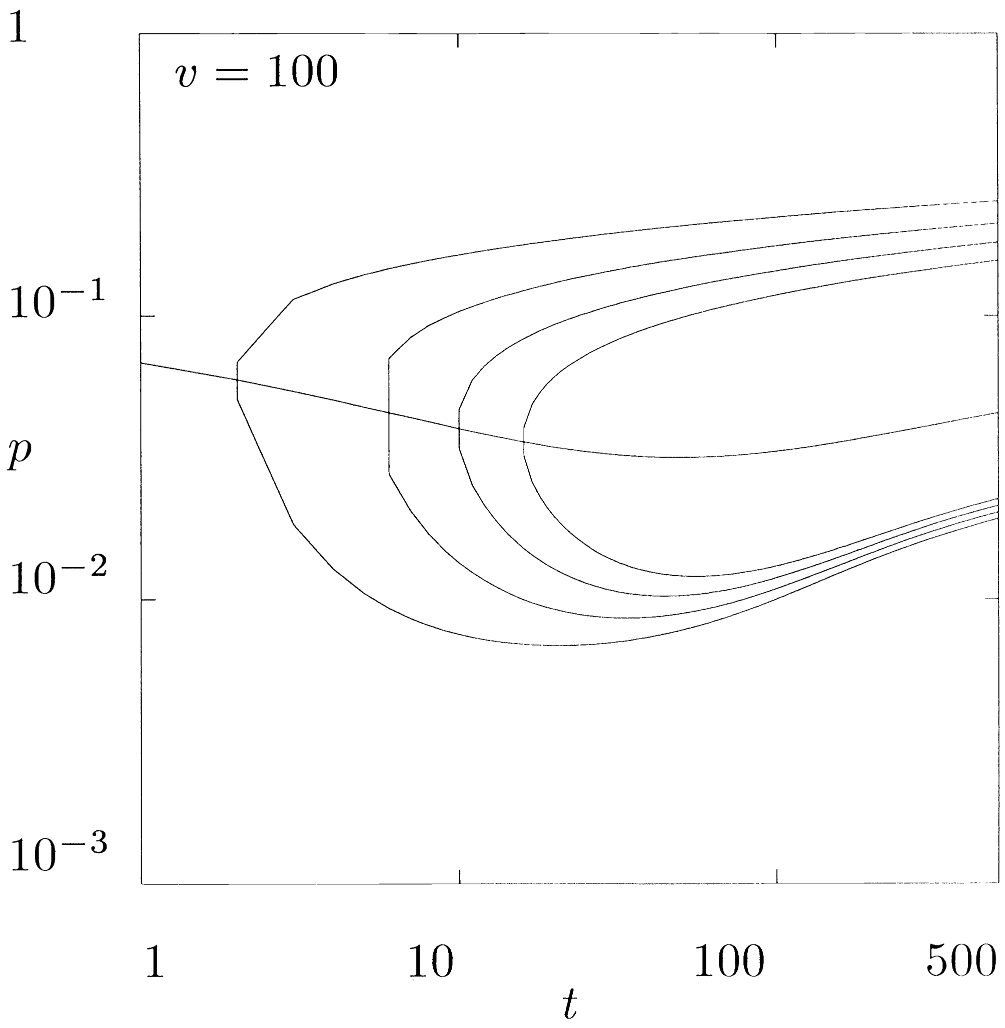


FIG. 2. *The outer contour shows for each t the value of p that results in an average of* 100 *nodes. Proceeding inward, contours for an average of* 100^2, 100^3, *and* 100^4 *are also shown. The line from the left side to the right side shows for each t which value of p results in the largest average number of nodes. Consecutive points on a contour are connected by a straight line.*

Fig. 2. Iwama's counting algorithm [7] for satisfiability has an average number of nodes bounded by v^n when

$$p > \sqrt{\frac{\ln t - \ln n}{v}}. \tag{6}$$

This is quite similar to (5), but somewhat better. Exact calculations for particular cases [9] confirm the suggestion of these upper bound calculations, i.e., Iwama's algorithm is faster for large p. (It is slow for small p.) Iwama's algorithm provides a count of the number of solutions, but does not give the solutions. When p is above the bound in (5) clause order backtracking quickly provides a complete list of solutions (in compressed form).

Equations (3) and (5) show that for all p we have polynomial average time when

$$t < 2(n-1) - O([\ln \ln v]/\ln v). \tag{7}$$

Other algorithms are known to be faster for small t, but none of them report all solutions. For example, the upper bound for the pure literal rule algorithm [10] is better than the upper bound for clause order backtracking for small t. (Comparing upper bounds does not always show which algorithm is best, but the measurements in [9] suggest that the upper bounds are close to the true values.) Comparing (3) with (28) of [10] shows that for small p, the pure literal rule algorithm has a better upper bound for

$$t < 2(nv \ln v)^{1/2}. \tag{8}$$

Equation (18) of [10] shows that the pure literal rule has a better upper bound for large p when

$$t < \frac{n-2}{\ln 2} \ln v \left[1 + O\left(\frac{1}{v}\right)\right]. \tag{9}$$

The fastest analyzed algorithm [5] for small t combines the unit clause rule, the pure literal rule, and resolution to eliminate variables that occur no more than two times. The use of resolution results in polynomial average time when t is below $v^{2/3}$ so long as p is not too large (when p is too large for this result, Iwama's algorithm is fast).

Among the analyzed satisfiability algorithms that report all solutions, clause order backtracking is one of the fastest. Selecting short clauses before long clauses would clearly improve the speed but complicate the analysis. A version of clause order backtracking where unit clauses are selected before longer clauses has been partly analyzed [9], but it is significantly faster than simple clause order backtracking only when pv is small. Resolution-based algorithms are much faster than clause order backtracking for small t and some values of p. For some applications the fact that they do not report all solutions is a disadvantage. Clause order backtracking runs in low degree polynomial average time for both large and small p. Only intermediate values of p lead to problems that are difficult for the algorithm.

4. Analysis. In the analysis, we associate clauses with nodes in the backtrack tree in a way that is equivalent to that described in §1 but different in detail. The backtrack tree always has a root. Each variable that is set when processing a clause results in two nodes, one for setting it to *true* and one for setting it to *false*. Thus, in Fig. 1, this way of counting associates node 1 with the root; nodes 2, 7, 3, and 4 with clause c_1; nodes 5 and 6 with clause c_2; and nodes 8 and 9 with clause c_3.

We now derive a sum that gives the average number of nodes generated by clause order backtracking. First, we consider the probability that the first clause attempts to contribute j nodes to the all false branch of the search tree (the branch where all variables are set to *false*). Then, we consider the probability that the first clause contributes zero nodes to the branch. (This happens when the first clause is a tautology.) Next, we consider the probability that the clauses following the first clause frustrate its attempt to contribute j nodes (by becoming *false* before j variables have been set). Finally, we use the fact that the probabilities are the same on all branches to obtain the sum for the average number of nodes.

The probability that a clause contains the first variable appearing positively but not negatively is $p(1-p)$. The probability that the first variable appears as either positive or negative, but not both is $2p(1-p)$. The probability that a clause contains k literals and the clause is not a tautology is

$$\binom{v}{k}2^k(p-p^2)^k(1-p)^{2v-2k} = \binom{v}{k}2^k p^k(1-p)^{2v-k}. \tag{10}$$

Suppose i variables are set to *false*. The probability that a random clause contains no true literals, contains k unset literals, and is not a tautology is

$$\binom{v-i}{k}2^k p^k(1-p)^{2v-i-k}. \tag{11}$$

The probability that such a clause contains its first true literal when j of its variables have been set to *false* is 2^{-j} (provided $j \le k$). Let $a_j(i)$ be the probability that a random clause is not a tautology, that it contains no true literals initially, and that it first contains a true literal after setting the first j of its previously unset variables. Summing (11) times 2^{-j} gives

$$a_j(i) = \sum_{k \ge j}\binom{v-i}{k}2^{k-j}p^k(1-p)^{2v-i-k} \quad \text{for } j \ge 1. \tag{12}$$

This is the probability that the first clause attempts to contribute j nodes to the all false branch of the backtrack tree.

The probability that a clause contains no true literals and is not a tautology is

$$\sum_k \binom{v-i}{k}2^k p^k(1-p)^{2v-i-k} = (1-p)^v(1+p)^{v-i}. \tag{13}$$

Let $a_0(i)$ be the probability that a random clause evaluates to *true* after i of its variables have been set to *false*. Any clause meets these conditions except those considered in (13). Thus,

$$a_0(i) = 1-(1-p)^v(1+p)^{v-i}. \tag{14}$$

This is the probability that a clause is skipped in Step 2 of the algorithm, and thus contributes zero nodes to the all false branch of the backtrack tree.

A clause evaluates to *false* if it contains only false literals. It cannot contain true or unset literals. The probability that a clause contains no such literals in $(1-p)^{2v-i}$. Thus, the probability that a clause evaluates to *false* is

$$1-(1-p)^{2v-i}. \tag{15}$$

Consider a particular predicate and a particular node in the associated backtrack tree. Let n be the number of clauses that contribute nodes to the path from the root to the selected node. (These are the clauses that are skipped in Step 2 and those that are used in Step 3 of the algorithm.) Let j_m be the number of nodes that the mth clause contributes to the path. Then $j_m \ge 0$ for $1 \le m < n$, and $j_n \ge 1$. Let s_m be the number of nodes contributed before those of the mth clause. That is,

$$s_m = \sum_{1 \le k < m} j_k. \tag{16}$$

Consider a particular set of j_m's and the set of all predicates that generate backtrack trees with that set of j_m's. Assume that each set variable is assigned the value *false*. The mth clause must contribute j_m nodes to the all false branch, with s_m variables having already been set. The remaining $t - n$ clauses (those that are not associated with any m) must not be *false* after $s_{n+1} - 1$ variables have been set. If all these conditions are met, then the backtrack tree has a node for both values of the last literal: one value results in the current clause containing a true literal, while the other value results in each literal of the current clause being either *false* or *unset*. Let $B(n, j_1, \ldots, j_n)$ be the probability that the backtrack tree for a random predicate contains a node where the mth clause contributes j_m (for $1 \le m \le n$) nodes to the path and where the last set variable makes the nth clause *true*. Since each condition pertains to one clause, and each clause is independently selected,

$$B(n, j_1, \ldots, j_n) = [1 - (1-p)^{2v - s_{n+1} + 1}]^{t-n} \prod_{1 \le m \le n} a_{j_m}(s_m). \tag{17}$$

The a_{j_1} factor in this formula gives the probability that the first clause of a random predicate contributes j_1 nodes, the a_{j_2} factor gives the probability that the second clause contributes j_2 nodes, etc. The $[1-(1-p)^{2v-s_{n+1}+1}]^{t-n}$ factor gives the probability that the clauses that have not contributed nodes have not previously evaluated to *false*. If you consider node 5 in Fig. 1, it was reached by the first clause contributing two nodes (nodes 2 and 4), the second clause contributing one node (node 5). To reach node 5 it was also necessary for the remaining clauses (clause c_3 in this case) not to evaluate to *false* at node 4. If the noncontributing clauses were not *false* at node 4, then they could not have been *false* at any of the earlier nodes on the path. The factor $a_2(0)$ gives the probability that the first random clause will contribute two nodes, and the $a_1(2)$ factor gives the probability that the second clause will contribute one node (even though two variables have already been set). The factor $[1-(1-p)^4]^1$ gives the probability that the one remaining clause does not evaluate to *false* even though two variables have been set at the parent of node 5.

Equation (17) gives the probability that the tree has a node with the specified characteristics. In particular, the last variable has been set so that its literal in the selected clause evaluates to *true*. With the same probability the tree has a node where that variable is set so that the literal evaluates to *false*. Thus, the average number of nodes corresponding to a particular $j_1, \ldots, j_n$, with each variable being assigned any preselected value (such as *false*), is $2B(n, j_1, \ldots, j_n)$.

In the random clause model each branch of the backtrack tree has the same probability. There are 2^i branches that have i variables set. Also each backtrack tree has a root node. Thus, the expected number of nodes in the backtrack tree is

$$N(t) = 1 + \sum_{1 \le n \le t} \sum_{j_1 \ge 0} \cdots \sum_{j_{n-1} \ge 0} \sum_{j_n \ge 1} 2^{s_{n+1}+1} B(n, j_1, \ldots, j_n). \tag{18}$$

The initial 1 counts the root of the backtrack tree, and the remaining terms count the nodes on level s_n for $1 \le s_n \le v$. Using $b_j(i) = 2^j a_j(i)$ and (17), equation (18) can be written as

$$N(t) = 1 + 2 \sum_{1 \le n \le t} \sum_{j_1 \ge 0} \cdots \sum_{j_{n-1} \ge 0} \sum_{j_n \ge 1} [1-(1-p)^{2v-s_{n+1}+1}]^{t-n} \prod_{1 \le m \le n} b_{j_m}(s_m). \tag{19}$$

5. Recurrence equation. The number of terms needed to evaluate $N(t)$ with (19) increases rapidly with t and v. We now derive a recurrence equation that is quicker to evaluate. Define

(20)
$$N(t,i) = 1+2\sum_{1\le n\le t}\sum_{j_1\ge 0}\cdots\sum_{j_{n-1}\ge 0}\sum_{j_n\ge 1}[1-(1-p)^{2v-i-s_{n+1}+1}]^{t-n}\prod_{1\le m\le n} b_{j_m}(i+s_m).$$

Note that $N(t) = N(t,0)$.

Giving separate consideration to the $n = 1$ case in (20) results in

$$N(t,i) = 1+2\sum_{j\ge 1}[1-(1-p)^{2v-i-j+1}]^{t-1}b_j(i)$$
$$+2\sum_{2\le n\le t}\sum_{j_1\ge 0}\cdots\sum_{j_{n-1}\ge 0}\sum_{j_n\ge 1}[1-(1-p)^{2v-i-s_{n+1}+1}]^{t-n}\prod_{1\le m\le n} b_{j_m}(i+s_m).$$

Giving separate consideration to the j_1 sum results in

$$N(t,i) = 1+2\sum_{j\ge 1}[1-(1-p)^{2v-i-j+1}]^{t-1}b_j(i)$$
$$+2\sum_{j} b_j(i)\sum_{2\le n\le t}\sum_{j_2\ge 0}\cdots\sum_{j_{n-1}\ge 0}\sum_{j_n\ge 1}$$
$$[1-(1-p)^{2v-i-j-s'_{n+1}+1}]^{t-n}\prod_{2\le m\le n} b_{j_m}(i+j+s'_m),$$

where $s'_k = s_k - j$. Now replace n by $n+1$, m by $m+1$, j_i by j_{i-1}, and s'_{i+1} by s_i. As a result, the new s_i is the sum of the new j_1 through j_n, and

(21)
$$N(t,i) = 1+2\sum_{j\ge 1}[1-(1-p)^{2v-i-j+1}]^{t-1}b_j(i)$$
$$+\sum_{j} b_j(i)\sum_{1\le n\le t-1}\sum_{j_1\ge 0}\cdots\sum_{j_{n-1}\ge 0}\sum_{j_n\ge 1}$$
$$2[1-(1-p)^{2v-i-j-s_{n+1}+1}]^{t-n-1}\prod_{1\le m\le n} b_{j_m}(i+j+s_m).$$

The part of (21) to the right of the second $b_j(i)$ is $N(t-1,i+j)-1$, so N obeys the recurrence equation

$$(22)\quad N(t,i) = 1+2\sum_{j\ge 1}[1-(1-p)^{2v-i-j+1}]^{t-1}b_j(i) + \sum_{j} b_j(i)[N(t-1,i+j)-1].$$

Simplifying, we get

$$(23)\quad N(t,i) = 1+b_0(i)[N(t-1,i)-1]$$
$$+\sum_{j\ge 1} b_j(i)\{2[1-(1-p)^{2v-i-j+1}]^{t-1}+N(t-1,i+j)-1\}.$$

Equation (20) gives the boundary condition $N(0,i) = 1$.

With (23), the average time can be computed in time $O(tv^2)$. Figure 2 shows the results of calculations for $v = 100$.

The recurrences in this paper were checked by comparing their solutions with the actual number of nodes generated by programs for the algorithms. Each recurrence was solved algebraically for $1 \le t \le 3$, $1 \le v \le 3$ using Macsyma. (A modification of (23) was checked for $1 \le t \le 6$, $1 \le v \le 6$, $tv \le 12$.) Also, each of the 2^{2tv} SAT problems was generated and solved with a program that counted the number of nodes that were generated. A problem with i literals has probability $p^i(1-p)^{2tv-i}$. Multiplying the count by the probability and summing over all problems gives a formula for the average number of nodes. (This formula is a polynomial in p with integer coefficients.) The formulas generated from the recurrences were identical with the ones generated by the programs.

6. Related recurrences. The number of nodes that results when a clause of length k is allowed to contribute at most k nodes can be obtained by replacing $2a_{j_n}$ in the previous derivations with $a_{j_n} + a'_{j_n}$, where a' is defined by (12) with the $k = j$ term omitted from the sum. Let $T(t,i)$ be the solution to

(24)

$$T(t,i) = 1 + \sum_{j\ge 1}[1-(1-p)^{2v-i-j+1}]^{t-1}\,[b_j(i)+b_{j+1}(i)] + \sum_j b_j(i)\,[T(t-1,i+j)-1],$$

with boundary condition $T(0,i) = 1$. The average number of nodes for the alternate counting method is $T(t) = T(t,0)$.

The recurrence for the number of solutions (in compressed form) is given by (23) with the constant term dropped. Let $S(t,i)$ be the solution to

$$S(t,i) = \sum_j b_j(i)S(t-1,i+j), \tag{25}$$

with the boundary condition $S(0,i) = 1$. The average number of solutions is given by $S(t) = S(t,0)$.

7. Comparison with simple backtracking.

THEOREM 1. *The average number of nodes generated by simple backtracking is greater than or equal to the average number of nodes generated by clause order backtracking for any distribution where the probability of a predicate is not changed by permuting the variable names in a single clause.*

Proof. Consider an arbitrary node q in a possible backtrack tree. Refer to this node with a string of T's and F's, where the ith symbol says which way to set the ith variable. (The important point is that the node can be described without reference to which variables are set on the path to the node.) For each predicate P consider whether or not node q occurs in the clause order backtrack tree for P. If node q does not occur, it is either because one of the clauses used to determine the order of assigning values to variables becomes *false* or because one of the other clauses becomes *false*. For the comparison between clause order backtracking and simple backtracking, construct a predicate P' in the following way. If node q does not occur in the clause order backtrack tree for P, then P' is the same as P. Otherwise, let k be the number of variables on the path from the root to node q, let v_i be the ith variable set by clause order backtracking on the path from the root to q, and let c_i be the clause that introduced v_i into the clause order backtrack tree. Form predicate P' from P by making the following change in variable names in the indicated clauses. For i from 1 to k, interchange variable names v_i and i in all clauses after c_i. For any node q, this mapping from P to P' is well defined, each P' is the image of a unique P, and P and P' have the same probability in the model for generating random clauses. Finally, if node q occurs in the clause order backtrack tree for P, then it also occurs in the simple backtrack tree for P'. Summing over all predicates P, we find that the probability of node q occurring in the clause order backtracking tree is no larger than the probability of it occurring in the simple backtracking tree. (For most distributions the node is less likely to be in the clause order tree, because a node q that does not appear in the clause order backtrack tree for P may still appear in the simple backtrack tree for P'.) $\square$

8. Asymptotics. Let $U(t)$ be an upper bound on $N(t,i)-1$ so that $U(t) \geq N(t,i)-1$ for $0 \leq i \leq v$ and $t \geq 0$. Since all the coefficients in (23) are positive

$$N(t,i)-1 \leq b_0(i)U(t-1)+\sum_{j\geq 1} b_j(i)\{2[1-(1-p)^{2v-i-j+1}]^{t-1}+U(t-1)\}. \tag{26}$$

Using

$$\sum_{j\geq 1} b_j(i) = 2p(v-i)(1-p)^v(1+p)^{v-i-1}, \tag{27}$$

and

$$1-(1-p)^{2v-i} \geq 1-(1-p)^{2v-i-j+1} \qquad \text{for } j \geq 1 \tag{28}$$

gives

$$\begin{aligned} N(t,i)-1 \leq{} & [1-(1-p)^v(1+p)^{v-i}]U(t-1) \\ & +2p(v-i)(1-p)^v(1+p)^{v-i-1}\{2[1-(1-p)^{2v-i}]^{t-1}+U(t-1)\}, \end{aligned}$$

which can be written as

$$\begin{aligned} N(t,i)-1 \leq{} & 4p(v-i)(1-p)^v(1+p)^{v-i-1}[1-(1-p)^{2v-i}]^{t-1} \\ & +\{[2p(v-i)-1-p](1-p)^v(1+p)^{v-i-1}+1\}U(t-1). \end{aligned} \tag{29}$$

Replacing the i's on the right side of (29) with the value that maximizes each part and replacing two factors of $(1+p)^{-1}$ with 1 gives

$$N(t,i)-1 \leq 4pv(1-p^2)^v[1-(1-p)^{2v}]^{t-1}+\{1+[2pv-1](1-p^2)^v\}U(t-1), \tag{30}$$

where we have assumed that $2pv-1$ is nonnegative. (When $2pv-1$ is negative, a slight modification of the current derivation shows that $N(t,i)$ is small, so that case is not considered further.)

Now let $U(t)$ be the solution to the recurrence

$$U(t) = 4pv(1-p^2)^v[1-(1-p)^{2v}]^{t-1}+\{1+[2pv-1](1-p^2)^v\}U(t-1), \tag{31}$$

with boundary condition $U(0)=0$. Equation (31) and the boundary for (23) imply that $U(t)$ is an upper bound on $N(t,i)-1$. Let $a = 1+(2pv-1)(1-p^2)^v$ and $b = 1-(1-p)^{2v}$. Then,

$$U(t) = 4pv(1-p^2)^v\frac{a^t-b^t}{a-b}. \tag{32}$$

Since $(a^t - b^t)/(a-b) = a^{t-1} + a^{t-2}b + a^{t-3}b^2 + \cdots + ab^{t-2} + b^{t-1}$, $a \geq b \geq 0$, and $a \geq 1$, we have

$$U(t) \leq 4ptv(1-p^2)^v a^{t-1}. \tag{33}$$

Also,

$$U(t) \leq \frac{4pv(1-p^2)^v}{a-b} a^t. \tag{34}$$

Figure 3 shows the contours from the approximation

$$N(t,i) \leq 1 + \frac{4pv(1-p^2)^v}{1+p} \frac{[1+(2pv-1)(1-p^2)^v/(1+p)]^t - [1-(1-p)^{2v}]^t}{(2pv-1)(1-p^2)^v/(1+p)+(1-p)^{2v}}. \tag{35}$$

This approximation is (32) modified by including the $(1+p)^{-1}$ factors that were dropped in deriving (32). When (35) gives a small result, it appears to have the main features of the real results (except in the lower right region where p is small and t is large). The $(1+p)^{-1}$ factors that are included in (35) but omitted from (32) have a noticeable effect in the upper part of Fig. 3, but the following derivations show that the factors are not significant when $v^{1/2}$ is large.

The right side of (35) is large when a is above 1, and it is small when a is near 1. Thus, it is small when pv is near 1/2 (so that $2pv - 1$ is small) and when p^2v is large (so that $2pv(1-p^2)^v$ is small). When p is larger than v^{-1} but smaller than $v^{-1/2}$, the bound is exponentially large (provided t is not small).

9. The small p case. From (33) we obtain

$$N(t,i) \leq 1 + 4ptv(1-p^2)^v[1+(2pv-1)(1-p^2)^v]^t. \tag{36}$$

When p is small, the $(1-p^2)^v$ term is not important. Replacing it with the upper limit of 1 gives

$$N(t,i) \leq 1 + 4ptv(2pv)^t. \tag{37}$$

The number of nodes is no more than v^n when

$$v^n \geq 1 + 4ptv(2pv)^t. \tag{38}$$

Rearranging and taking logarithms gives

$$\ln(2pv) \leq \frac{n \ln v - \ln(4ptv) - O(v^{-n})}{t}. \tag{39}$$

Since $p \leq 1$, the number of nodes is no more than v^n when

$$p \leq \frac{1}{2v} e^{[(n-1)\ln v - \ln t - O(1)]/t}. \tag{40}$$

10. The large p case. From plugging in the values of a and b into (34) we obtain

$$N(t,i) \leq 1 + \frac{4pv(1-p^2)^v}{(2pv-1)(1-p^2)^v + (1-p)^{2v}} [1+(2pv-1)(1-p^2)^v]^t. \tag{41}$$

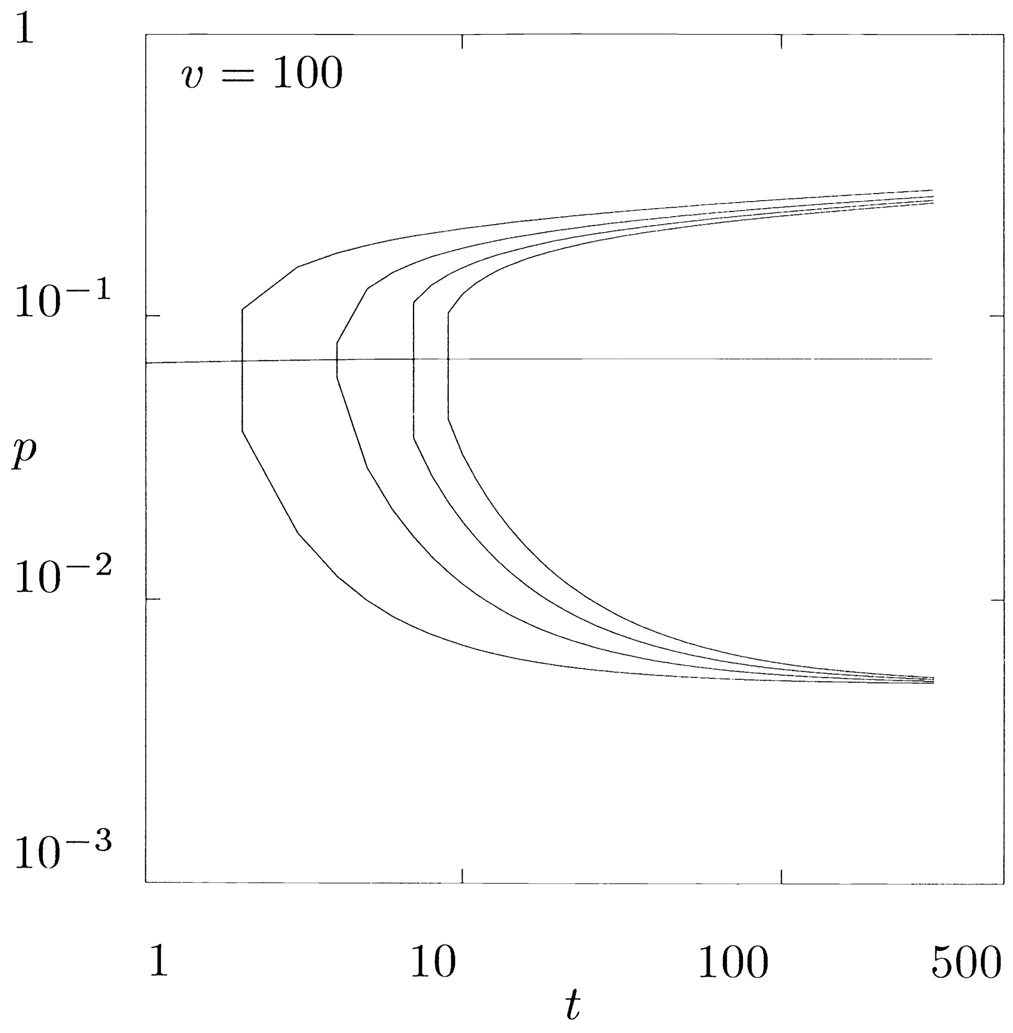


FIG. 3. *The contours for the approximation to the average number of nodes.*

Dropping terms in a way that leads to an even large bound gives

$$\begin{aligned}N(t,i) &\le 1 + \frac{4pv(1-p^2)^v}{(2pv-1)(1-p^2)^v}[1+2pv(1-p^2)^v]^t\\ &\le 1 + \frac{2}{1-1/(2pv)}[1+2pv(1-p^2)^v]^t\\ &\le 1 + \left[1+O\left(\frac{1}{pv}\right)\right][1+2pv(1-p^2)^v]^t.\end{aligned}$$

The number of nodes is no more than v^n when

(42) $$v^n \ge 1 + 2\left[1+O\left(\frac{1}{pv}\right)\right][1+2pv(1-p^2)^v]^t.$$

Writing this as $v^n - 1 \geq 2[1 + O(1/(pv))][1 + 2pv(1-p^2)^v]^t$ and taking logarithms gives $[n \ln v - \ln(1 - v^{-n})]/t \geq \ln[1 + 2pv(1-p^2)^v] + \ln 2 + \ln[1 + O(1/(pv))]$, which can be written as

$$e^{[n \ln v - O(1)]/t} - 1 \geq 2pv(1-p^2)^v. \tag{43}$$

Expanding the exponential in a power series gives

$$\frac{n \ln v}{2ptv}\left[1 + O\left(\frac{n \ln v}{t}\right)\right] \geq (1-p^2)^v. \tag{44}$$

Taking logarithms gives

$$\ln p + \ln t + \ln v - \ln \ln v - O\left(\frac{n \ln v}{t}\right) \leq v \ln(1-p^2). \tag{45}$$

Dividing by v and replace $\ln(1-p^2)$ with $-p^2 - O(p^4)$ gives

$$p^2 \geq \frac{\ln p + \ln t + \ln v - \ln \ln v}{v} - O(p^4) - O\left(\frac{n \ln v}{tv}\right). \tag{46}$$

For any $\delta > 0$ and large v, (46) is satisfied by

$$p \geq \sqrt{\frac{(1+\delta)\ln t + (\ln v)/2}{v}}. \tag{47}$$

When $\ln[(\ln t)/(\ln v)] < \ln v$ (i.e., when $t < v^{\ln v}$), the bound on p is also satisfied with $\delta = 0$.

To obtain conditions where the average time is barely exponential (bounded by $(1+\epsilon)^v$ for small positive ϵ), replace $n \ln v$ with ϵ in (3) and (46).

Acknowledgment. We wish to thank the referees for their careful reading of the paper and their helpful suggestions.

REFERENCES

[1] C. A. Brown and P. W. Purdom, *An average time analysis of backtracking*, SIAM J. Comput., 10 (1981), pp. 583–593.

[2] K. Bugrara, Y. Pan, and P. Purdom, *Exponential average time for the pure literal rule*, SIAM J. Comput., 18 (1988), pp. 409–418.

[3] S. A. Cook, *The complexity of theorem-proving procedures*, in the Proceedings of the 3rd ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1971, pp. 151–158.

[4] J. Franco, *On the Occurrence of Null Clauses in Random Instances of Satisfiability*, Tech. Report 291, Department of Computer Science, Indiana University, Bloomington, IN, 1989.

[5] ———, *Elimination of infrequent variables improves average case performance of satisfiability algorithms*, SIAM J. Comput., 20 (1991), pp. 1119–1127.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979, pp. 38–44, 48–50.

[7] K. Iwama, *CNF satisfiability test by counting and polynomial average time*, SIAM J. Comput., 18 (1989), pp. 385–391.

[8] P. W. Purdom, *Search rearrangement backtracking and polynomial average time*, Artificial Intelligence, 21 (1983), pp. 117–133.

[9] P. W. Purdom, *A survey of average time analyses of satisfiability algorithms*, J. Inform. Process., 13 (1990), pp. 449–455. An earlier version appeared as *Random Satisfiability Problems*, in Proceedings Algorithms and Complexity, The Institute of Electronics, Information and Communication Engineers, Tokyo, 1989, pp. 253–259.

[10] P. W. PURDOM AND C. A. BROWN, *The pure literal rule and polynomial average time*, SIAM J. Comput., 14 (1985), pp. 943–953.

[11] ———, *Polynomial-average-time satisfiability problems*, Inform. Sci., 41 (1987), pp. 23–42.

[12] P. W. PURDOM, C. A. BROWN, AND E. L. ROBERTSON, *Backtracking with multi-level dynamic search rearrangement*, Acta Informatica, 15 (1981), pp. 99–113.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 318–331, April 1993

FACTORING RATIONAL POLYNOMIALS OVER THE COMPLEX NUMBERS*

CHANDERJIT BAJAJ†, JOHN CANNY‡, THOMAS GARRITY§, AND JOE WARREN¶

Abstract. NC algorithms are given for determining the number and degrees of the factors, irreducible over the complex numbers **C**, of a multivariate polynomial with rational coefficients and for approximating each irreducible factor. NC is the class of functions computable by logspace-uniform boolean circuits of polynomial size and polylogarithmic depth. The measures of size of the input polynomial are its degree, coefficient length, number of variables (d, c, and n, respectively). If n is fixed, we give a deterministic NC algorithm. If the number of variables is not fixed, we give a random (Monte-Carlo) NC algorithm in these input measures to find the number and degree of each irreducible factor.

After reducing to the two-variable, square-free case, we apply the classical algebraic geometry fact that the absolute irreducible factors of $(P(z_1, z_2) = 0)$ correspond to the connected components of the real surface (or *complex* curve) $P(z_1, z_2) = 0$ minus its singular points. In finding the number of connected components of the surface $P = 0$, the surface is projected to the the z_2-plane. The singular points of $P(z_1, z_2)$ lie over the projection's critical values. The inverse image of a grid isolating the critical values in the z_2-plane lifts to a one-dimensional real curve skeleton on the surface $(P = 0)$ whose number of connected components is precisely the number of connected components of $P = 0$ minus its singular points. The connectivity of this curve skeleton is constructed symbolically using Sturm sequences associated with the various polynomials defining these maps. Given the number of irreducible factors and their degrees, the actual factors can be reconstructed using the recent result of Neff [*Proceedings of the* 31*st Annual Symposium on Foundations of Computer Science*, pp. 152–162] on finding zeros of one-variable polynomials in NC.

Key words. polynomials, factoring, NC, algebraic geometry

AMS(MOS) subject classification. 12D05

1. Introduction.

Factoring polynomials is a basic problem in symbolic computation with applications as diverse as theorem proving and computer-aided design. Our goal is to approximate the factors, irreducible over the complex numbers, of a multivariable polynomial with rational coefficients in deterministic NC with respect to the polynomial's degree and coefficient size, assuming that the number of variables is fixed. Further, if the number of variables is not fixed, we will find the number of irreducible factors and each of their degrees in random NC with respect to the polynomial's degree, coefficient size and number of variables. The key is that this is a topological problem, not an algebraic one. We will show that the number of irreducible factors is exactly equal to the number of connected components of a certain semialgebraic set.

Since factoring is held as a touchstone problem in computer algebra, there have been a number of recent successes in factoring various types of polynomials over different fields. Methods for factoring polynomials with rational coefficients over the rational numbers are well known. Kaltofen [16] and Lenstra, Lenstra, and Lovasz [20] establish that factoring polynomials in a fixed number of indeterminates over the field of rational numbers **Q** is in polynomial time.

For factoring rational polynomials over **C**, Noether [23], Davenport and Trager [6], and Heintz and Sieveking [13] each give methods that require time exponential in the

*Received by the editors May 17, 1989; accepted for publication (in revised form) November 25, 1991.

†Department of Computer Science, Purdue University, West Lafayette, Indiana 47907. The work of this author was supported in part by Army Research Office contract DAAG29-85-C0018 and Office of Naval Research contract N00014-88-K-0402.

‡Computer Science Division, University of California, Berkeley, California 94720. The work of this author was supported in part by a David and Lucille Packard Fellowship.

§Department of Mathematics, Rice University, Houston, Texas 77251.

¶Department of Computer Science, Rice University, Houston, Texas 77251. The work of this author was supported in part by National Science Foundation grant IRI 88-10747.

degree of the input polynomial. DiCrescenzo and Duval [7] and Duval [8] give geometric methods of factorization based on algebraic geometry. Kaltofen [15] describes an NC method for testing whether a rational polynomial is irreducible over **C**. The method involves computing approximate roots and their corresponding minimum polynomials. The first polynomial time algorithm for factoring over **C** seems to have been given by Chistov and Grigoryev [4]. Until recently, it has been an open problem whether approximating the factors, irreducible over **C**, of a rational polynomial is in NC. Kaltofen [18], using techniques quite different from ours, has also derived an algorithm for approximating these factors in NC.

Recall that NC is the class of functions computable by log-space uniform Boolean circuits of polynomial size and polylogarithmic depth. Thus the running time of an NC algorithm will be polylogarithmic, allowing a polynomial number of processors that work in parallel. Given a polynomial P with rational coefficients, the input size is measured by the number of variables n degree d coefficient size c and the number of nonzero ooefficients s. We show that the general problem of computing number and degrees of the factors is in random NC in these measures, in the Monte-Carlo sense (definitely fast, probably correct). If the number of variables is fixed, or if the polynomial P is dense, we give a deterministic NC solution for also approximating the irreducible factors. Finally, if the polynomial is represented as a straight-line program of length p our algorithm runs in random NC plus the time to evaluate the polynomial at an integer point. By the parallelization result of Valiant et al. [27], any straight-line program of size p and degree d can be converted into an equivalent program of polynomial size, and polylogarithmic depth in d and p, which can be evaluated in NC. Although this result applies to the real number model, it extends easily to bit complexity since every node of the SLP represents a polynomial in the input variables and constants. If we have a bound on the degree of these polynomials, this also bounds the size of any intermediate coefficient when we evaluate the program numerically.

However, the conversion to a low-depth straight-line program is itself not in NC, and seems intrinsically sequential because of constant evaluation, which is P-complete. So we cannot run our algorithm in random NC for straight-line program polynomials unless we are given a program of low depth.

The paper can be divided into two main parts. In the first part, in §§2 and 3 we show how to find the number and the degree of the irreducible factors of a two-variable, square-free polynomial with rational coefficients. In the second main part, in §§4 and 5, we show how to reduce the general case of a multivariable polynomial to a two-variable, square-free polynomial, and then how to approximate the actual irreducible factors.

In §2, we recall a fact of the classical algebraic geometry: namely, that the number of factors of a two-variable, square-free polynomial $P(z_1, z_2)$ is precisely the number of connected components of the real, two-dimensional surface $P = 0$ minus the singular points of P. This is the key, allowing us to translate the original algebraic problem of factoring to the topological one of finding connected components of a special semi-algebraic set. Section 3, the technical heart of the paper, finds the number of these connected components. The two-dimensional surface $P(z_1, z_2) = 0$, lying in $\mathbf{C}^2$ or equivalently $\mathbf{R}^4$, can be projected to the z_2 plane, with a finite number of critical values. The singular points of the surface lie over critical points of the projection. In §3.1, the projection is described. In §3.2, we create a grid of horizontal and vertical lines in the z_2-plane with the property that inside each box of the grid there is at most one critical point of the projection. The inverse image of this grid on the surface $P(z_1, z_2) = 0$ forms a curve skeleton. By Theorems 3 and 4, the number of connected components of this curve skeleton will be precisely the number of connected components of $P(z_1, z_2) = 0$ minus its singular points. Since this curve skeleton is a graph, if we can compute its adjacency matrix in NC, we can find the number of connected components in NC. Section 3.3 shows how to find this matrix by using sign sequences of various Sturm sequences, all of which, by Theorem 6, can be computed in NC. This will give the number of irreducible factors of a two-variable square-free polynomial with rational coefficients. We will see that the degree of each factor has already been determined, by applying Bezout's theorem.

In §4, we show how to reduce the problem of factoring a multivariable polynomial with rational coefficients to that of factoring a square-free, two-variable polynomial. All we do is note that the proof of Bertini's theorem in [21] actually describes an algorithm that runs in NC. Finally, in §5, using the recent wonderful result of Neff [22] that the roots of a one-variable rational polynomial can be approximated in NC, we actually show how to approximate the irreducible factors.

2. Connectivity and factorization.

2.1. Preliminaries. Let $P_i(z_1, \ldots, z_n) \in \mathbf{C}[z_1, \ldots, z_n]$ for $i = 1, \ldots, k$ be polynomials with complex coefficients in n variables. Let $V(P_1, \ldots, P_k)$ denote the set of common zeros of these polynomials in $\mathbf{C}^n$

$$V(P_1, \ldots, P_k) = \{z \in \mathbf{C}^n | P_i(z) = 0,\ i = 1, \ldots, k\}.$$

This is an example of an *algebraic* set. For a single polynomial P, the set $S = V(P)$ is called a *hypersurface*. A hypersurface S is said to be *irreducible* if the zero set of the polynomial $P(z_1, \ldots, z_n)$ is irreducible over $\mathbf{C}$. More generally, an algebraic set is irreducible if it cannot be expressed as a finite union of proper algebraic subsets. An irreducible algebraic set is called a *variety*.

For the rest of the paper, we assume that P is square-free (irreducible factors have multiplicity one). Note that if the original P is not square-free, we may compute the square-free part of P by computing

$$P/GCD\left(P, \frac{\partial P}{\partial z_1}\right),$$

where P is monic in z_1. This computation may be performed in NC using the greatest common divisor algorithm of [1].

The key observation in §2.2 will be that there is a fundamental relationship between the *singular* points of an algebraic set and its irreducible components.

DEFINITION. Let $S = V(P)$ be a hypersurface with P a square-free polynomial. The set of *singular points* of S, denoted $Sing(S)$, is defined by

$$Sing(S) = S \cap V\left(\frac{\partial P}{\partial z_1}, \ldots, \frac{\partial P}{\partial z_n}\right). \qquad (1)$$

For example, an irreducible algebraic plane curve has at most a finite number of singular points. More generally, the singular set can be defined for any algebraic set, but we will not give a definition here. Intuitively, the singular points of an algebraic set are the points where the set is not smooth (smooth points have neighborhoods diffeomorphic to some $\mathbf{C}^k$) or where tangent lines are ill-defined.

2.2. Topology of zero sets of reducible polynomials. This section is the key to translating the algebraic problem of factoring to the topological problem of counting connected components.

Removing the singular set from an algebraic set may split it into several connected components. Here connectivity is equivalent to pathwise connected in the usual (metric) topology. As the following theorems show, these components correspond exactly to the irreducible components of the curve.

THEOREM 2.1. *The set S is irreducible if and only if $S - Sing(S)$ is connected.*

THEOREM 2.2. *The irreducible components of set S are exactly the closures of the connected components of $S - Sing(S)$.*

Both theorems are extremely classical and are consequences of the next two lemmas.

LEMMA 2.3. *Let the set S have distinct irreducible components $S_1, S_2, \dots, S_k$. Then for any i and j, $S_i \cap S_j \subseteq Sing(S)$.*

For hypersurfaces, this is a straightforward calculation. For the general case, this is Theorem 6 in [26, Chap. 2, §2].

LEMMA 2.4. *If S is irreducible, and Y is any proper algebraic subset of S, then $S - Y$ is connected.*

This is Corollary 4.16 of [21].

3. Computing connected components. We have reduced the problem of finding the number of irreducible factors of $P(z_1, z_2)$ to the problem of counting the number of connected components of the surface $V(P) - Sing(V(P))$. In the first part of this section, we discuss an algorithm for counting the number of connected components. We then discuss how the algorithm can be executed in parallel.

3.1. Projecting to the plane. In this subsection, we treat the algebraic set $V(P(z_1, z_2))$ as a branched cover of the z_2 plane, showing that there will only be a finite number of critical values (which will be defined in a minute) and, more importantly, that the singular points of P must lie over critical values. In §§3.2 and 3.3, we will construct a grid, isolating the critical values, in the z_2 plane whose inverse image on $V(P)$ will be a graph with the same number of connected components as $V(P) - Sing(V(P))$. This reduces the problem to constructing the adjacency matrix of this graph.

We express the complex coordinates z_1 and z_2 in terms of their real and imaginary parts: $z_1 = x_1 + y_1\mathbf{i}$ and $z_2 = x_2 + y_2\mathbf{i}$. The polynomial $P(z_1, z_2) = P(x_1, y_1, x_2, y_2)$ can also be expressed in terms of its real and imaginary parts: $P(x_1, y_1, x_2, y_2) = P_1(x_1, y_1, x_2, y_2) + P_2(x_1, y_1, x_2, y_2)\mathbf{i}$. Thus, $V(P)$ can be expressed as the two-dimensional real surface $V(P_1, P_2)$ in $\mathbf{R}^4$. Let $\pi : \mathbf{R}^4 \to \mathbf{R}^2$ be the projection map that takes those points (x_1, y_1, x_2, y_2) lying on $V(P_1, P_2)$ and maps them to (x_2, y_2). By a change of variables, we can assume that $P(z_1, z_2)$ has a z_1^d term, where d is the degree of P, implying that P does not have any factors univariate in z_2. Thus, π must be finite-to-one everywhere.

DEFINITION. If $F : \mathbf{R}^n \to \mathbf{R}^m$ is a differentiable map, $p \in \mathbf{R}^n$ is a *critical point* of F if the Jacobian dF of F is not surjective at p. The image of a critical point is a *critical value*.

In complex algebraic geometry, the critical points are called *ramification points* and the critical values, *branch points*. A point that is not a critical point is called a *regular point*, and the preimage of a *regular value* consists of regular points only.

This projection map has only a finite number of critical points.

LEMMA 3.1. *There are only a finite number of critical points in the projection map π from the surface $V(P_1(x_1, y_1, x_2, y_2), P_2(x_1, y_1, x_2, y_2))$ to the (x_2, y_2) plane.*

This lemma can be stated more precisely.

LEMMA 3.2. *The critical points of the projection map π are the points in the intersection of $V(P)$ and the surface $\frac{\partial P_1}{\partial z_1}$.*

The proofs of both lemmas are contained in §§4 and 5 in [19, Chap. 2]. The critical values of the projection map are the zeros of the one variable polynomial $R(z_2) = Res_{z_1}(P, \frac{\partial P}{\partial z_1})$. This expression is frequently referred to as the *discriminant*.

Note that the singular points of the polynomial P, the points where both partial derivatives vanish, must be contained in the critical points.

3.2. Reduction to curve skeleton. In this subsection we reduce the problem of finding the number of connected components of $V(P(z_1, z_2)) - Sing(V(P))$, which is the number of irreducible factors of P, to finding the number of connected components of a real one-dimensional curve skeleton (or graph) on the surface $V(P) - Sing(V(P))$.

In §3.1 we showed that the projection map π from $V(P)$ to the z_2 or (x_2, y_2) plane has only a finite number of critical values. Let G be a grid of a finite number of horizontal and vertical lines in the (x_2, y_2) plane, so that each critical value is in at most one cell of the grid. For now, we assume such a grid exists. We will give more details about how to construct such a grid later in this section. For now, the key point is that G isolates the critical values of the projection map.

Let K be the inverse image of the grid G on the surface $V(P)$. Note that K is one dimensional and, since no critical values lie on G, lies on $V(P) - Sing(V(P))$. K can be interpreted as defining a graph whose vertices are those points on K lying over the vertices of the grid G. Two vertices in this graph are adjacent if their corresponding points on K are connected.

The following two theorems will show that the number of connected components of the curve skeleton K is the same as the number of connected components of $V(P) - Sing(V(P))$. Theorem 3 will also yield the degree of each irreducible factor of $P(z_1, z_2)$.

THEOREM 3.3. *The number of vertices of the curve skeleton K, over any vertex of the grid G, in each connected component of $V(P) - Sing(V(P))$ is exactly the degree of that component.*

Proof. Let $P(z_1, z_2) = \Pi Q_i(z_1, z_2)$, where each Q_i is an irreducible factor of P. Each connected component of $V(P) - Sing(V(P))$ is of course $V(Q_i) - Sing(V(P))$, for some i.

A vertex of the grid G is a point a in the z_2 plane. The inverse image of this vertex is given by the zeros of the one-variable polynomial $P(z_1, a)$. The inverse image in each component are the zeros of $Q_i(z_1, a)$. By the Fundamental Theorem of Algebra, each component will have exactly degree of Q_i points, which is also the degree of the component. We do not have to worry about multiple roots of these polynomials since the grid G does not pass through any critical values. $\square$

THEOREM 3.4. *Any path in $V(P) - Sing(V(P))$ connecting two points in K is homotopic (i.e., can be continuously deformed) to a path in K.*

Proof. Let σ be a path in $V(P) - Sing(V(P))$ connecting two points of K. By slightly deforming σ, we can assume that the projection of σ by π misses any critical values in the z_2 plane. In each cell of the grid G, deform the path $\pi(\sigma)$ to the actual grid G, so that the area swept out by the deformation does not contain any critical values. This is possible since each cell will contain at most one critical value. Since the area swept out by the deformation does not contain a critical value, the deformation can be lifted to $V(P) - Sing(V(P))$. Thus the path σ can be continuously deformed to a path on the skeleton K. $\square$

We next describe the actual construction of the grid G. In the last section we show that the critical points of the projection map π are the points in $V(P, \frac{\partial P}{\partial z_1})$. Then the critical values are the zeros of the one variable polynomial,

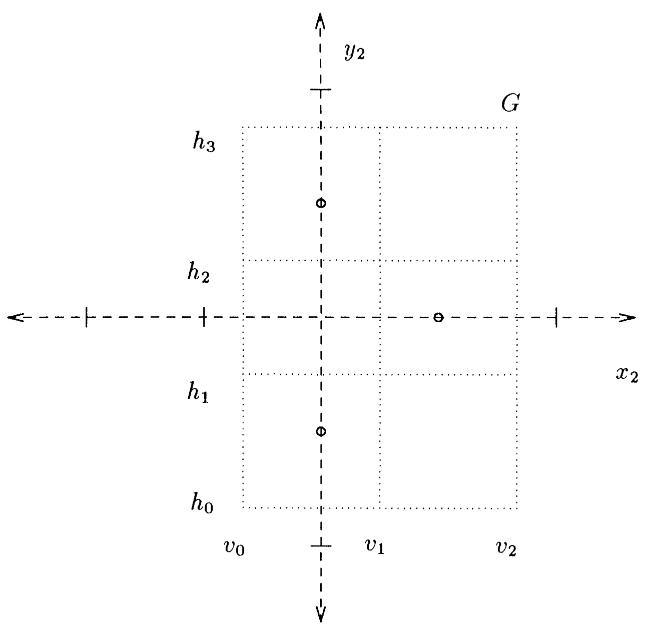


FIG. 1. *A grid plane whose cells contain at most one critical point.*

$$R(z_2) = Res_{z_1}\left(P, \frac{\partial P}{\partial z_1}\right), \tag{2}$$

where $Res_{z_1}(P, Q)$ is the resultant of the polynomials of P and Q treated as one-variable polynomials in z_1. The edges of G are chosen to be parallel to the x_2 and y_2 axes.

The vertical edges are the lines $x_2 = v_i$ with the $(v_0 < v_1 < \cdots)$ real constants chosen so that the open interval (v_i, v_{i+1}) contains at most one of the distinct real components of the complex zeros of $R(z_2)$. The horizontal edges are the lines $y_2 = h_i$ with the $(h_0 < h_1 < \cdots)$ real constants chosen so that the open interval (h_i, h_{i+1}) contains at most one of the distinct imaginary components of the complex zeros of $R(z_2)$. Figure 1 illustrates this situation where $R(z_2) = (z_2 - 1)(z_2 + \mathbf{i})(z_2 - \mathbf{i})$.

The lines of G form rectangular cells in the z_2 plane, intersecting in vertices. Note that each cell in the grid contains at most one critical value.

The grid G may now be used to construct the curve skeleton K directly on $V(P)$. In fact K, the inverse image of G under the projection π onto the z_2 plane is the one-dimensional curve $V(P) \cap \pi^{-1}(G)$. As described earlier, the vertices of this graph are the points on $V(P)$ lying over each vertex (v_k, h_l) in G. These points are the complex roots of the univariate polynomial $P(z_1, v_k + \mathbf{i}h_l)$. The edges of the graph correspond to algebraic curve segments of K. Figure 2 illustrates three curve segments over two vertices s and t of the grid G, adjacent on a vertical grid line. The curve segments have been projected onto the x_1y_2 plane.

Thus to find the number of connected components of $V(P) - Sing(V(P))$, we must find the number of connected components of the graph K. To determine the connectivity of K, we need only the adjacency information between points of K, not the actual curve segments. We will next describe a fast parallel method for computing this adjacency information.

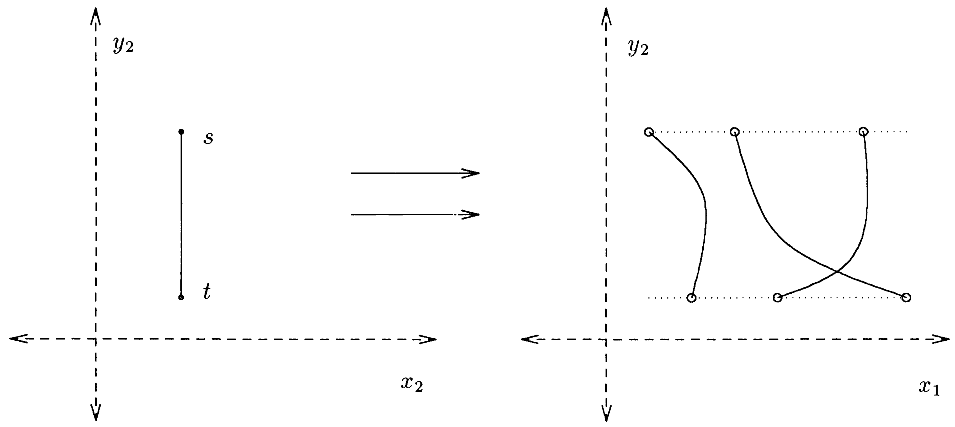


FIG. 2. *Curve segments on S joining vertices of K.*

3.3. Construction of curve skeleton. In this section we show how to construct the adjacency matrix for the curve skeleton K in NC with respect to the degree of the polynomial and the coefficient size. The key will be the symbolic representation by sign sequences of roots of various polynomials, allowing us to keep track of the order of the roots, which will be needed for the matrix.

In §3.3.1, we quickly review Sturm sequences for one-variable polynomials and then give a generalization for multivariable polynomials. In §3.3.2, we show explicitly how to construct the adjacency matrix of the curve skeleton.

3.3.1. Sturm sequences. Sturm sequences are classical. Let $p(x)$ be a one-variable polynomial. Consider the following sequence $p_0(x), \ldots, p_n(x)$ of polynomials:

$$\begin{aligned} p_0 &= p \\ p_1 &= dp(x)/dx \\ &\vdots \quad \vdots \\ p_k &= q_{k-1}p_{k-1} - p_{k-2} \\ &\vdots \\ &p_n \end{aligned} \tag{3}$$

where p_k is simply the negative of the remainder obtained by dividing p_{k-2} by p_{k-1}. Since $p(x)$ is a polynomial, the last term p_n must be a constant. If $p(x)$ is square-free, p_n must be nonzero. Sturm sequences can be computed in NC [1].

The importance of Sturm sequences lies in that they provide an easy way of determining how many real roots a polynomial has between two points.

THEOREM 3.5. *Let $p(x)$ be a univariate real polynomial with Sturm sequence $(p_0(x), \ldots, p_n(x))$. Let a and b be real numbers that are not roots of $p(x)$. Then the number of real roots of $p(x)$ between a and b is equal to the number of sign changes in the sequence $(p_0(a), \ldots, p_n(a))$ minus the number of sign changes in the sequence $(p_0(b), \ldots, p_n(b))$.*

The proof can be found in many places, such as [14, Chap. 6].

We will also need the following multivariable version of Sturm sequences. Let Σ be a collection of rational polynomials $(p_1, \ldots, p_m)$ in n variables.

THEOREM 3.6. *Let $p_1(x_1, \ldots, x_n) = 0, \ldots, p_m(x_1, \ldots, x_n) = 0$ be a system of rational coefficient polynomial equations having a finite number of solution points. Denote the l real solution points not at infinity as $\alpha_j \in \mathbf{R}^n, j = 1, \ldots, l$. Let $q_1(x_1, \ldots, x_n), \ldots, q_k(x_1, \ldots, x_n)$ be a set of polynomials. Then the set of sign sequences of $q_1(\alpha_j), \ldots, q_k(\alpha_j), j = 1, \ldots, l$ can be computed in* NC *if m is fixed.*

This theorem is a corollary of Lemma 2.4 in [3].

3.3.2. Parallel adjacency calculation. We now discuss how to compute the grid G in the z_2 plane and the adjacency information for K in NC with respect to the degree of the original polynomial and the coefficient size.

Let $R(z_2)$ be the polynomial whose zeros are the critical values of the projection map from $V(P)$ to the z_2 plane, defined by (2). Without loss of generality, we assume that $R(z_2)$ is square-free (if not, make it so). We write $R(z_2)$ in terms of its real and imaginary parts:

$$R(x_2, y_2) = R_1(x_2, y_2) + \mathbf{i}R_2(x_2, y_2).$$

The complex zeros of R are at the simultaneous real zeros of R_1 and R_2. Let

$$U(x_2) = Res_{y_2}(R_1, R_2), \qquad H(y_2) = Res_{x_2}(R_1, R_2). \tag{4}$$

The real zeros of U contain the x_2-coordinates of the critical values and the zeros of H contain the y_2-coordinates. Again, we ensure that U and H are square-free.

Since U is square-free, the solutions v_i to the equation

$$\frac{dU(x_2)}{dx_2} = 0 \tag{5}$$

generate vertical lines that separate the critical points. Likewise, the solutions h_i to the equation

$$\frac{dH(y_2)}{dy_2} = 0 \tag{6}$$

generate horizontal lines that separate the critical points. Finally, if A is a constant so that all roots of both U and H are greater than $-A$ and less than A, then the grid G will consist of the lines from (5) and (6) and

$$x_2 = \pm A$$

$$y_2 = \pm A.$$

This gives a symbolic description of G. We next use this description of the grid with Sturm sequences to compute the adjacency information for graph of the curve skeleton K.

We describe a method for computing this adjacency information in the x_2 direction in G. The y_2 direction is similar. Let (v_i, h_j) and (v_i, h_{j+1}) be two adjacent vertices in G. These vertices lie on the grid line $x_2 = v_i$. Over each of these vertices lies d points in $V(P)$. These points form the vertices of K. In $\mathbf{R}^4$, the intersection of $x_2 = v_i$ and

$V(P)$ define d algebraic curve segments in K. These curve segments form the edges in K, joining pairs of vertices in K, each lying over a distinct grid vertex.

We do not explicitly attempt to construct and follow the curve segments. Instead, we symbolically compute the adjacency information. Project $(V(x_2 - v_i) \cap V(P))$ onto the x_1y_2-plane via resultants. As shown in the next section, this projection can be chosen so that only nodal singularities are introduced into the curve. To determine adjacency information, we need only locate and detect the relative position of these nodes with respect to the vertices of K, since at each node the order of vertices changes. Since we are interested in nothing more than the relative order of these points, these calculations can be done via Sturm sequences. For example, in Fig. 3, denote the endpoints of the segment (v_i, h_j) and (v_i, h_{j+1}) by s and t. Over s, assume that there are four vertices s_1, s_2, s_3, and s_4 in K. Likewise, over t there are four vertices t_1, t_2, t_3, and t_4. The projected curve segments link the s_i to the t_j and the position of the nodes determines which s vertices are connected to which t vertices.

Specifically, consider the three polynomials:

$$T(x_1, x_2, y_2) = Res_{y_1}(P_1, P_2).$$

$$\frac{dU(x_2)}{dx_2} \tag{7}$$

$$N(x_2, y_2) = Res_{x_1}(T, \tfrac{\partial T}{\partial x_1}).$$

$V(T)$ is the projection of $V(P)$ to x_1, x_2, y_2 space. Allowing x_1 and y_2 to be free variables for $\frac{dU(x_2)}{dx_2}$, the intersection of $V(\frac{dU(x_2)}{dx_2})$ with $V(P)$ yields curves lying in planes parallel to the x_1y_2 plane and through the vertical grid lines.

Now allow y_2 to be a free variable for $\frac{dU(x_2)}{dx_2}$, the points on $V(N, dU/dx_2)$, restricted to the line $x_2 = v_i$, are the images of the nodes of $V(T)$ projected to the y_2 axis. Thus, allowing x_1 to be a free variable, $V(N, dU/dx_2)$ consists of lines in the x_1y_2 plane, parallel to the x_1 axis, containing nodes of the projected plane curve (the dotted horizontal lines in Fig. 3).

Compute the sign sequences of the following polynomials at the common zeros of the system (7):

- The Sturm sequence of dU/dx_2.
- The Sturm sequence of dH/dy_2.
- The Sturm sequence with respect to y_2 of $N(x_2, y_2)$.
- The Sturm sequence with respect to x_1 of $\frac{\partial T}{\partial x_1}$.

By Theorem 3.6, these sign assignments can be computed in NC with respect to the size of the input polynomials.

To compute adjacencies for K we proceed as follows: As y_2 increases, the number of sign alternations of the Sturm sequence for dH/dy_2 increases monotonically. We first sort all the sign assignments according to the number of sign alternations in this Sturm sequence within each sign assignment. This partitions all the zeros of (7) into classes according to y_2 coordinate. Each of these classes provides adjacency information for a particular slice $y_2 = h_i$.

Next we sort within each class according to the number of sign alternations of the Sturm sequence of dU/dx_2. This gives us a collection of classes that lie on the same horizontal grid segment between two adjacent vertical grid lines.

Then sort within classes according to number of sign alternations of the Sturm sequence of $N(x_2, y_2)$. The sign assignments with a zero correspond to the image in the (x_2, y_2) plane of the nodes of the curve seqments.

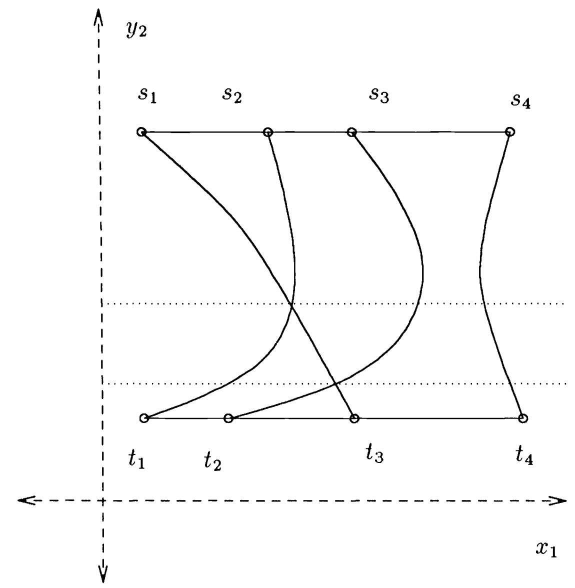


FIG. 3. *Effect of nodes on adjacency calculations.*

Finally, we sort the sign assignments according to number of alternations of the Sturm sequence of $\partial T/\partial x_1$. This orders the points with the same x_2, y_2-coordinates along lines parallel to the x_1 axis. One of these sign assignments will have a zero assignment to the polynomial $\partial T/\partial x_1$, and this is the sign assignment of the node point itself. From the position of this sign assignment in the ordering, we infer the relative position of the node point along the dotted line and therefore among the branches of the curve in K. In Fig. 3, we are ordering the points along the dotted lines.

To generate the graph K, we label the d vertices of K over a given gridpoint with $1, \ldots, d$. These labels come from the x_1 ordering of the corresponding points in K. Each node can be represented as a permutation (an exchange of two adjacent elements) of the indices of the curve branches that cross at the node. To determine the permutation as we move in y_2 past k nodes, we compose the permutations of the nodes. The composition can be done in NC by composing adjacent (in y_2 ordering) permutations, then composing adjacent pairs of these, etc. The final permutation gives the change in ordering from one gridpoint to the next and provides the d edges joining corresponding vertices of K.

One performs similar calculations to compute adjacency information in the horizontal direction.

3.3.3. Projections introducing only nodal singularities. We need to prove the assumption used in the §3.3.2 that a smooth space curve defined by the intersection of two surfaces can be projected to a plane curve with at worst nodes as singularities. The following is no doubt well known.

LEMMA 3.7. *Let C be a space curve defined by the intersection of two algebraic surfaces. Then we can choose a projection map in* NC *with respect to the degrees of the polynomials defining the surfaces so that the image of C is a plane curve with at worst ordinary nodes as singularities.*

Proof. Recall that a projection is defined as follows: choose a point p off of both the curve C and a plane P. Let q be any point on C. Then the unique line defined by p and

q intersects the plane P in exactly one point. The projection maps q to this intersection point.

By [21, pp. 132–135] or [12, Th. IV.3.10], a point will give rise to a nonnodal projection if it lies on a multisecant of the curve (i.e., a secant intersecting the curve in more than two places), a tangent of the curve, or a secant with coplanar tangent lines. In these references, it is shown that the set of nonnodal projection points forms a proper algebraic subset. It can be checked that the polynomials defining this algebraic set are bounded by the degrees of the polynomials defining the surfaces. Thus by [25], and using arguments similar to those that will be given for Theorems 4.2 and 4.3 in §4, we can, in NC with respect to the degrees of the polynomials defining the surfaces, choose a point of projection.

One technical note is needed. Both [21] and [12] work over the complex numbers. But a proper algebraic subset of $\mathbf{C}^n$ cannot contain all of the underlying real points. Thus we are indeed guaranteed a good point of projection.

4. Reduction to bivariate factorization. All of the previous work depended on the original polynomial being in two variables. In this section we show how to reduce the problem of factoring a multivariable polynomial to the two-variable case.

There have been a number of papers giving reductions from multivariate to bivariate factorization. The first appeared in Heintz and Sieveking [13], and made use of Bertini's theorem, as will we. This was a randomized irreducibility test that worked for sparse multivariate polynomials. The idea was extended to factorization in [10]. In [16] a reduction was given which is in deterministic polynomial time if the number of variables is fixed, or if the polynomials are dense. Kaltofen [17] later gave a different randomized reduction for the sparse case. These randomized reductions work for polynomials represented as straight-line programs as well as sparse polynomials. An NC reduction for the dense case was given in [15].

For the complex case, we give a new randomized reduction that requires fewer bits per random coefficient $O(\log d)$ than the previous methods $O(d)$ for [17] and $O(d^2)$ for [10]. Thus our reduction also runs in deterministic NC if the number of variables is fixed, or if the polynomials are dense. For sparse polynomials, the reduction is in random NC in the degree d number of variables n coefficient size c and number of nonzero terms s. For straight-line program polynomials, the parallel running time is the sum of a polylogarithmic function of measures d, n, c, plus the time to evaluate the polynomial at an integer point.

Given a polynomial $P(x_1, \ldots, x_n)$, we assume that it is square-free. We wish to develop a constructive version of Bertini's theorem, which states that the intersection of an irreducible variety with a generic plane will be an irreducible plane curve. Luckily the proof given in Mumford as Theorem 4.17 in [21] is constructive. Next we describe algebraically the set of intersecting planes that violate Bertini's theorem and then, using Schwartz's lemma [25], show how to choose an intersecting plane satisfying Bertini's theorem.

We use the following theorem (which is Theorem 4.17 in [21]).

THEOREM 4.1. *Let $X \subset \mathbf{P}^n$ (complex projective n-space) be an r-dimensional projective variety and let $M^{n-r-1} \subset \mathbf{P}^n$ be a linear space disjoint from X. Let $p : X \to \mathbf{P}^n$ be the projection from M and let*

$$B = (x \in \mathbf{P}^r \mid p \text{ not smooth over } x)$$

so that

$$(X - p^{-1}B) \to \mathbf{P}^r - B$$

is a finite-sheeted covering space. Let $l \subset \mathbf{P}^r$ be any line that meets B transversely. Then

$$p^{-1}(l - l \cap B) \to l - l \cap B$$

is also a connected covering space, hence p^{-1} is an irreducible curve.

In fact by Corollary (4.18) in [21], the above line l can be chosen so that $p^{-1}(l)$ will intersect the variety X transversely. For our case, X will be an irreducible hypersurface $V(P(x_0, \dots, x_n))$, where $P(x_0, \dots, x_n)$ is the homogenized version of a polynomial $P(x_1, \dots, x_n)$. Thus $V(P(x_0, \dots, x_n))$ is the projective closure of the affine hypersurface $V(P(x_1, \dots, x_n))$. Then M^{n-r-1} will be for us simply a point off of $V(P(x_0, \dots, x_n))$.

Following [21], we will find three points that span the plane $p^{-1}(l)$. The first point, p_0 will be the point of projection. Any point off of the hypersurface $V(P(x_0, \dots, x_n))$ will work.

As in the theorem, let B be the set of points, in the hyperplane $\mathbf{P}^{n-1}$, over which the projection from p_0 is not smooth. In our terminology, B is the set of critical values. We now need to find a line l in the hyperplane $\mathbf{P}^{n-1}$ that is transverse to B. But this is easy. In $\mathbf{P}^{n-1}$, choose a point p_1 off of B and then project B from this point to some $\mathbf{P}^{n-2}$. Let B' be the critical values in $\mathbf{P}^{n-2}$ of this projection map, and let our third point p_2 be a point off of B'. The line l will be defined by the points p_1 and p_2 and the plane $p^{-1}(l)$ will be spanned by p_0, p_1, and p_2. Again, the justification for these choices is in [21].

The following choices must be made. First we must find a point p_0 off of a degree ($d = \deg P$) hypersurface in $\mathbf{P}^n$. Then we must find a point p_1 off of B, which is a degree $d(d-1)$ hypersurface in $\mathbf{P}^{n-1}$. Note that the degree of B is $d(d-1)$ since it is given by the resultant of the polynomial P and some first partial of P. Finally we must find the point p_2 off of the set B', which is a hypersurface of degree $d(d-1)(d(d-1)-1)$ in $\mathbf{P}^{n-2}$. Luckily it is not hard to choose points off of a proper algebraic set. Further, since the set of bad points is a proper algebraic set, we can assume, even though the statement of the above theorem is for projective space, that we are in an affine space $\mathbf{C}^n$ (i.e., we can dehomogenize, since we will be losing only the points on the hyperplane at infinity, which is a proper algebraic subset of degree one).

THEOREM 4.2. *Let $P(x_1, \dots, x_n)$ be an irreducible polynomial of degree d. Let E be a finite subset of $\mathbf{C}$. Then the probability that the bivariate polynomial $Q(x, y)$ defining the plane curve, given by the intersection of $V(P)$ by a plane spanned by three points in E^n, is reducible is less than $(d^4 - 2d^3 + d^2 + d + 1)/|E|$, where $|E|$ is the cardinality of E.*

Proof. We make use of Schwartz's lemma [25] that the number of points in the set E^n (E a finite subset of $\mathbf{C}$) that lie in an algebraic set $Z \subset \mathbf{C}^n$ of degree d is at most $d|E|^{n-1}$.

The set of points off of which we want to choose our three points is the union of $V(P)$, B, B', and the hyperplane at infinity; hence, has degree $d + d(d-1) + d(d-1)(d(d-1)-1)$, which is $d^4 - 2d^3 + d^2 + d + 1$. The result follows. □

COROLLARY 4.3. *Let $P(x_1, \dots, x_n)$ be a polynomial of degree d with k irreducible, distinct factors. Let E be a finite subset of $\mathbf{C}$. Then the probability that the bivariate polynomial $Q(x, y)$ defining the plane curve given by the intersection of $V(P)$ by a plane spanned by three points in E^n does not have k factors with corresponding degrees is less than $d^4 - 2d^3 + d^2 + d + 1/|E|$.*

This follows because the points can be chosen exactly as in Theorem 4.2.

In order to achieve a probability of failure less than ϵ, we make sure that the following holds true: $|E| > d^4 - 2d^3 + d^2 + d + 1/\epsilon$. Choosing integer values for elements of E therefore requires $(4 \log d + \log \frac{1}{\epsilon})$ bits. For a deterministic algorithm, we take $|E| = d^4 - 2d^3 + d^2 + d$.

Finally, note that by Bertini's theorem almost every reduction to two variables will work. Thus if we do not fix the number of variables, our algorithm will run in random (Monte-Carlo) NC.

5. Factorization information.

We now want to approximate each factor of the polynomial $P(z_1, \dots, z_n)$, which is possible due to the recent work of Neff [22] on approximating the roots of a one-variable polynomial with rational coefficients in NC. The arguments used are very straightforward, so we will only sketch the proof.

Let our polynomial $P(z_1, \dots, z_n) = \prod P_i(z_1, \dots, z_n)$, where each P_i is irreducible of degree d_i. We can assume, after a change of coordinates, that P and the P_i are monic in the variable z_n. There are then $(\frac{(d_i+n)!}{(d_i)!(n)!} - 1)$ unknown coefficients for each P_i. We will now determine how to approximate these coefficients by solving an associated system of linear equations: $AX = B$, where A will be an integral invertible matrix, X will be a column vector of coefficients for P_i, and B will be a column vector of algebraic numbers.

Let $a_1, a_2, \dots, a_{n-1}$ be integers. Using [22], approximate the roots of the one-variable polynomial $P(a_1, a_2, \dots, a_{n-1}, z_n)$. Assume for a moment that we can determine which roots are associated with which irreducible factor P_i and can thus approximate the one-variable polynomial $P_i(a_1, a_2, \dots, a_{n-1}, z_n)$. Then given an integer a_n, we can approximate the algebraic number $b = P(a_1, a_2, \dots, a_{n-1}, a_n)$. By choosing $(\frac{(d_i+n)!}{(d_i)!(n)!} - 1)$-tuples of integers $(a_1, a_2, \dots, a_{n-1}, a_n)$ and treating the coefficients of the P_i as unknowns, we can approximate the coefficients by solving a linear system $AX = B$. Here B is the column vector of the algebraic numbers b from the various $P_i(a_1, a_2, \dots, a_{n-1}, a_n)$ and A is the square matrix arising from evaluating all monomials of degree d_i in n variables at the points $(a_1, a_2, \dots, a_{n-1}, a_n)$. We must choose our tuple so that the matrix A is invertible, but this is clearly no problem.

There is one difficulty with this method. We do not yet know how to determine which roots of $P(a_1, a_2, \dots, a_{n-1}, z_n)$ are associated to which factor P_i. We do know how to do this in the two-variable case. For a polynomial $P(z_1, z_2) = \prod P_i(z_1, z_2)$, we can determine which roots of $P(a_1, z_2)$ are associated to which factors P_i, since this is precisely the information that is contained in the connectedness of the earlier constructed curve skeleton, provided that we choose the point a_1 to be on the grid in the z_1 plane, which we can do by enlarging the grid. The general case is now easy. Given two tuples $(a_1, a_2, \dots, a_{n-1})$ and $(b_1, \dots, b_{n-1})$, intersect $P(z_1, \dots, z_n) = 0$ with the plane parallel to the z_n axis containing the points $(a_1, a_2, \dots, a_{n-1}, 0)$ and $(b_1, \dots, b_{n-1}, 0)$. This reduces the problem of associating roots of P to the two-variable case, which we can do. Of course, we cannot intersect $P(z_1, \dots, z_n) = 0$ with any plane, but this is not a true difficulty. In the previous section we have an algebraic description of the planes that do not intersect $P(z_1, \dots, z_n) = 0$ correctly. Thus we simply must choose our $(n-1)$ tuples so that resulting planes perform properly.

Acknowledgments. We would like to thank Jim Renegar for his comments and discussion of this work. We would also like to thank the referee for many suggestions for improving the exposition of the paper. An earlier version of this work was presented at ISSAC '89, in Portland, Oregon.

REFERENCES

[1] A. BORODIN, J. VON ZUR GATHEN, AND J. HOPCROFT, *Fast parallel matrix and* GCD *computations*, Inform. and Control, 52 (1982), pp. 241–256.

[2] J. Canny, *A new algebraic method for robot motion planning and real geometry*, in Proceedings of the 28th Symposium on Foundations of Computer Science, 1987, pp. 39–48.

[3] ———, *Some algebraic and geometric computations in* PSPACE, in Proceedings of the 20th Symposium on Theory of Computing, 1988, pp. 460–467.

[4] A. L. Chistov and D. Y. Grigoryev, *Subexponential-Time Solving Systems of Algebraic Equations* I., Steklov Institute, LOMI preprint, 1983, E-9-83.

[5] P. Ciarlet, *Introduction to Numerical Linear Algebra and Optimisation*, Cambridge Texts in Applied Mathematics, Cambridge University Press, London, 1989.

[6] J. Davenport and Trager, *Factorization over finitely generated fields*, in Proceedings of the 1981 ACM Symposium on Symbolic Algebraic Computation, 1981, pp. 200–205.

[7] C. DiCrescenzo and D. Duval, *Computations on curves*, Eurosam'84, LICS 174, 1984, pp. 100–107.

[8] D. Duval, *Diverses Questions Relatives au Calcul Formel Avec Des Nombres Algebriques*, Thèse, L'Université Scientifique, Technologique et Médicale de Grenoble, Grenoble, France, 1987.

[9] R. Dvornicich and C. Traverso, *Newton Symmetric Functions and the Arithmetic of Algebraically Closed Fields*, in Proc. AAECC-5, Springer Lecture Notes Computer Science, No. 356, 1987, pp. 216–224.

[10] J. von zur Gathen, *Irreducibility of multivariate polynomials*, J. Comput. System Sci. No. 31, 1985, pp. 225–264.

[11] P. Griffiths and J. Harris, *Principles of Algebraic Geometry*, John Wiley and Sons, New York, 1978.

[12] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, Berlin, New York, 1977.

[13] J. Heintz and M. Sieveking, *Absolute primality of polynomials is decidable in random polynomial time in the number of variables*, in Proceedings of the 1981 International Conference on Automata, Languages, and Programming, Springer Lecture Notes in Computer Science, Vol. 115, 1981, pp. 16–28.

[14] P. Henrici, *Applied and Computational Complex Analysis*, John Wiley and Sons, New York, 1988.

[15] E. Kaltofen, *Fast parallel absolute irreducibility testing*, J. Symbolic Comput., 1 (1985), pp. 57–67.

[16] ———, *Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization*, SIAM J. Comput., 14 (1985), pp. 469–489.

[17] ———, *Effective Hilbert irreducibility*, Inform. and Control, 66 (1985), pp. 123–137.

[18] ———, *Effective Noether irreducibility forms and applications*, in Proceedings of the 23rd Annual Symposium on Theory of Computing, 1990, pp. 54–63.

[19] K. Kendig, *Elementary Algebraic Geometry*, Springer-Verlag, Berlin, New York, 1977.

[20] A. Lenstra, H. Lenstra, and L. Lovasz, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[21] D. Mumford, *Algebraic Geometry* I: *Complex Projective Varieties*, Springer-Verlag, Berlin, New York, 1970.

[22] C. A. Neff, *Specified precision polynomial root isolation is in* NC, in Proceedings of the 31st Annual Symposium on Foundations of Computer Science, 1990, pp. 152–162.

[23] E. Noether, *Ein algebraisches Kriterium fur absolute Irreduzibilitat*, Math. Ann., 85 (1922), pp. 26–33.

[24] V. Pan, *Fast and efficient algorithms for sequential evaluation of polynomial zeros and of matrix polynomials*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, 1985.

[25] J. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. Assoc. Comput. Mach., 27 (1980), pp. 701–717.

[26] I. Shafarevich, *Basic Algebraic Geometry*, Springer-Verlag, Berlin, New York, 1974.

[27] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 332–348, April 1993

OPTIMAL STOCHASTIC ALLOCATION OF MACHINES UNDER WAITING-TIME CONSTRAINTS*

E. G. COFFMAN, JR.†, LEOPOLD FLATTO†, AND PAUL E. WRIGHT†

Abstract. The nonpreemptive scheduling of $n \geq 1$ stochastic jobs is considered to minimize the expected number of parallel machines needed to meet given waiting-time constraints. The number of machines available is unlimited. The running times of the jobs are denoted $T_1, \ldots, T_n$ and are taken to be independent samples of an exponentially distributed random variable T with mean 1. Job waiting times are to be bounded stochastically by a nonnegative random variable W, independent of $T_1, \ldots, T_n$. At time zero, a timer is started with an initial value W, and job scheduling begins. When the timer expires, all jobs still waiting for a machine are assigned to available machines. Only the distributions of W and the job durations $T_1, \ldots, T_n$ are known to the scheduler in advance.

A scheduling policy is defined which is proven to be optimal when W has an exponential distribution, and is asymptotically optimal as $n \to \infty$, when W is a constant (hard-deadline). In the exponential case, an explicit formula for the cost function is derived. The uniqueness question is also resolved. The paper concludes with a partial analysis of the general hard-deadline problem, which leads to a policy that we think is optimal. A proof of optimality, however, remains an open problem.

Key words. Bellman equation, optimal scheduling, Markovian decision process

AMS(MOS) subject classifications. 90B22, 90B35, 93E20

1. Introduction.

We consider the nonpreemptive scheduling of $n \geq 1$ stochastic jobs so as to minimize the expected number of parallel machines needed to meet given waiting-time constraints. The number of machines available is unlimited. The running times of the jobs are denoted $T_1, \ldots, T_n$ and are taken to be independent samples of a random variable T.

Job waiting times are to be bounded stochastically by a nonnegative random variable W, independent of T, with distribution $C(t)$. Thus at time zero a timer is started with the initial value W and job scheduling begins. To simplify terminology, we equip the timer with a bell which rings when the interval W expires. When the bell rings, all jobs not currently running and still waiting are assigned to available machines. We emphasize that, except when W is a constant, the value of the timer is not available to the scheduling policy before the bell rings; but its distribution is known. Similarly, job lengths are also not known in advance, but their common distribution is known. The problem is to find, within the class of nonpreemptive policies, a policy which minimizes the expected cost, with cost defined as the number of distinct machines used throughout a schedule.

The analysis of later sections suggests that this *machine minimization* problem is quite difficult for general distributions of T and W. A similar, but more precise statement can be made about the corresponding combinational optimization problem, where the values of $T_i, 1 \leq i \leq n$ and W are all given in advance and the maximum number of machines used throughout a schedule is to be minimized. This problem is strongly NP-complete, a result which follows easily from the strong NP-completeness of one-dimensional bin-packing [1].

This paper simplifies the general problem by assuming that T has an exponential distribution. In the usual way, this allows us to formulate a Markov decision process without having to include elapsed job running times as part of the state variable. For convenience we take the mean of the distribution as the time unit, $E(T) = 1$. For general

*Received by the editors January 8, 1991; accepted for publication (in revised form) February 17, 1992.
†AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

$C(t)$, the optimization problem remains difficult. However, for $C(t)$ also exponential, we give in §§3 and 4 a complete characterization of optimal policies.

In particular, it is shown that, for some $1 \leq k \leq n$, the following policy is optimal. The policy begins by assigning jobs to k machines. Thereafter, until the bell rings, jobs are run only on these machines; whenever a machine finishes a job, it is assigned a waiting job, if any. If there are any jobs still waiting when the bell rings, then all such jobs are assigned to new machines. This is called the *constant-k* policy; it is analyzed for general $C(t)$ in §2 in preparation for the results of later sections. Note that the cost of any constant-k schedule is simply the maximum of k and the number of unfinished jobs when the bell rings.

Section 5 treats the hard-deadline model, where W is a known constant. This section first shows that the best constant-k policy is asymptotically optimal as $n \rightarrow \infty$. A Bellman equation is then developed for the general case. Numerical evidence and partial results derived from the Bellman equation suggest the structure that an optimal policy is likely to have. We present such a policy, but a proof of its optimality has so far eluded us.

In the context of stochastic scheduling theory, machine minimization relates to the *makespan minimization problem*, where the system of machines is assumed fixed and the latest job finishing time is to be minimized in expected value. The latter problem has been studied extensively under a variety of assumptions, including machines of different speeds, precedence constraints among jobs, and different distributions for the T_i (see [2], [3], [4], [5] and the references therein for the literature on this problem). The model here concerns the complementary, equally important problem of allocating resources to meet customer demands on waiting times.

In addition to the usual job-shop applications cited in the literature of stochastic scheduling theory, there are equivalent problems in computer/communication system design. For example, at a network node the machines become channels or trunks, and W bounds the message delay prior to the start of transmission. Machine minimization also falls within the theory of stochastic real-time scheduling. A distinctive feature of the model here is that job running times are not known in advance. Also, the waiting time constraint is imposed on the delays prior to being assigned machines, rather than the job finishing times.

2. The constant-k policy. This section assumes a general clock-time distribution $C(t) = Pr\{W \leq t\}$. For given n and $k, 1 \leq k \leq n$, let $F(n,k)$ denote the expected cost incurred by the constant-k policy. Then $F(n) = \min_{1 \leq k \leq n} F(n,k)$ denotes the expected cost incurred by a best constant-k policy. Recall that the constant-k policy initially assigns k of the given n jobs to k machines. Of the remaining $m = n - k$ jobs, let R count those that will eventually be served before the bell rings. When the bell rings, there still remain $m - R$ jobs requiring $m - R$ new machines, in addition to the k already in use. Thus, the total number M of machines used by the constant-k policy is given by $M = m - R + k = n - R$, so that

$$F(n,k) = E[M] = n - E[R]. \tag{2.1}$$

In particular, when $k = n$, we have $R = 0$ so that $F(n,n) = n$.

The distribution for R is denoted by $\pi_i = Pr\{R = i\}, 0 \leq i \leq m = n - k$. For $i < m, Pr\{R = i|W = t\}$ is simply the probability that, in a Poisson process with rate k, i events occur in $[0,t)$. Thus, we have a mixture of truncated Poisson distributions

$$\pi_i = \begin{cases} \displaystyle\int_0^\infty e^{-kt}\frac{(kt)^i}{i!}dC(t), & 0 \le i < m, \\ \displaystyle\int_0^\infty \sum_{j\ge m} e^{-kt}\frac{(kt)^j}{j!}dC(t), & i = m. \end{cases} \tag{2.2}$$

Introduce the Laplace transform $\hat{c}(s) = \int_0^\infty e^{-st}dC(t)$ and rewrite (2.2) as

$$\pi_i = \begin{cases} p_i - p_{i+1}, & 0 \le i < m, \\ p_m, & i = m, \end{cases}$$

with

$$p_i = \int_0^\infty \sum_{j\ge i} e^{-kt}\frac{(kt)^j}{j!}dC(t) = \sum_{j\ge i}\frac{(-k)^j}{j!}\hat{c}^{(j)}(k), \tag{2.3}$$

where $\hat{c}^{(j)}(k)$ is the jth derivative of $\hat{c}(s)$ evaluated as $s = k$. We have

$$\begin{aligned} \mathbf{E}[R] &= \sum_{i\ge 0} i\pi_i = \sum_{i=0}^{m-1} i(p_i - p_{i+1}) + mp_m = \sum_{i=1}^{m} ip_i - \sum_{i=1}^{m}(i-1)p_i \\ &= \sum_{i=0}^{m} p_i, \end{aligned}$$

so (2.1) and (2.3) give

$$F(n,k) = n - \sum_{i=1}^{m}\sum_{j\ge i}\frac{(-k)^j}{j!}\hat{c}^{(j)}(k). \tag{2.4}$$

This formula simplifies when C is a completely monotone distribution or the difference of two such distributions. In this case, C has a density $c(t)$ that can be expressed as

$$c(t) = \int_0^\infty e^{-ts}d\alpha(s) \tag{2.5}$$

for some (signed) measure $\alpha(s)$ which is finite on $[0,\infty)$. Since $\int_0^\infty c(t)dt = 1$, the measure $\alpha(s)$ must satisfy $\int_0^\infty d\alpha(s)/s = 1$. As an example of (2.5) we observe that weighted sums of exponentials (hyperexponential distributions) are completely monotone. In this case $\alpha(s)$ consists of a sequence of point masses, one for each exponential.

From (2.5) we get

$$\begin{aligned} \int_0^\infty e^{-kt}t^j c(t)dt &= \int_0^\infty\int_0^\infty e^{-(k+s)t}t^j dt\, d\alpha(s) \\ &= \int_0^\infty \frac{j!}{(k+s)^j}d\alpha(s); \end{aligned}$$

so by (2.3), $\sum_{i=1}^{m} p_i$ becomes

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j\geq i}\int_0^\infty e^{-kt}\frac{(kt)^j}{j!}c(t)dt &= \int_0^\infty \sum_{i=1}^{m}\sum_{j\geq i}\frac{k^j}{(k+s)^{j+1}}d\alpha(s)\\ &= \int_0^\infty \sum_{i=1}^{m}\left(\frac{k}{k+s}\right)^i\frac{d\alpha(s)}{s}\\ &= \int_0^\infty k\left[1-\left(\frac{k}{k+s}\right)^m\right]\frac{d\alpha(s)}{s^2}.\end{aligned}$$

Then (2.4) reduces to

$$F(n,k) = n - k\int_0^\infty \left[1-\left(\frac{k}{k+s}\right)^{n-k}\right]\frac{d\alpha(s)}{s^2}. \tag{2.6}$$

Note that $1/s^2$ in the integrand causes no difficulties, by virtue of the integrability condition on the measure and the fact that the bracketed term vanishes at $s = 0$.

In general, little insight is provided by (2.4); the computation of $F(n)$ becomes a numerical problem. However, useful information is available from the asymptotic behavior of $F(n,k)$ as $n \to \infty$. Asymptotic estimates can be obtained without reference to (2.4), using only elementary properties of the Poisson distribution. We need the following intuitive, preliminary result.

LEMMA 2.1. *Let $a(\lambda)$ be any function satisfying $a(\lambda)/\lambda \to \mathcal{B}, 0 < \mathcal{B} < \infty$ as $\lambda \to \infty$. Let Y_λ denote a Poisson random variable with parameter λ, and define $\overline{Y}_\lambda$ as Y_λ truncated at $a(\lambda)$, i.e., $\overline{Y}_\lambda = \min\{Y_\lambda, a(\lambda)\}$. Then as $\lambda \to \infty$,*

$$\mathbf{E}[\overline{Y}_\lambda] \sim \begin{cases}\lambda & \text{if } \mathcal{B} \geq 1,\\ a(\lambda) & \text{if } \mathcal{B} \leq 1.\end{cases} \tag{2.7}$$

Proof. If $\mathcal{B} < 1$, choose $\varepsilon > 0$ so that $a(\lambda) < \lambda(1-\varepsilon)$ for all λ sufficiently large. On the set $\{|Y_\lambda - \lambda| < \lambda\varepsilon\}, \overline{Y}_\lambda = a(\lambda)$. Thus, since $\mathbf{E}[\overline{Y}_\lambda] \leq \min\{\lambda, a(\lambda)\} = a(\lambda)$, we have for all λ sufficiently large,

$$a(\lambda)Pr\{|Y_\lambda - \lambda| < \lambda\varepsilon\} \leq \mathbf{E}[\overline{Y}_\lambda] \leq a(\lambda).$$

But $\lim_{\lambda\to\infty} Pr\{|Y_\lambda - \lambda| < \lambda\varepsilon\} = 1$ is a standard property of the Poisson distribution, so (2.7) follows for $\mathcal{B} < 1$.

If $\mathcal{B} > 1$ choose ε so that $a(\lambda) > \lambda(1+\varepsilon)$, and if $\mathcal{B} = 1$ choose ε so that $a(\lambda) > \lambda(1-\varepsilon)$, for all λ sufficiently large. Then in either case $\overline{Y}_\lambda \geq \lambda(1-\varepsilon)$ when $|Y_\lambda - \lambda| < \lambda\varepsilon$, so that

$$\lambda(1-\varepsilon)Pr\{|Y_\lambda - \lambda| < \lambda\varepsilon\} \leq \mathbf{E}[\overline{Y}_\lambda] \leq \lambda,$$

and (2.7) follows for $\mathcal{B} > 1$ as above.

THEOREM 2.1. *For any constant $0 < \theta \leq 1$,*

$$\lim_{n\to\infty}\frac{F(n,\lfloor n\theta\rfloor)}{n} = 1 - \left\{\theta\int_0^{(1-\theta)/\theta} tdC(t) + (1-\theta)\left[1 - C\left(\frac{1-\theta}{\theta}\right)\right]\right\}. \tag{2.8}$$

Proof. From (2.1)

$$\frac{F(n,k)}{n} = \int_0^\infty \left[1 - \frac{\mathbf{E}[R|W=t]}{n}\right] dC(t). \tag{2.9}$$

For $k = \lfloor n\theta \rfloor, [R|W=t]$ is a Poisson random variable with parameter $\lfloor n\theta \rfloor t$ truncated at $n - \lfloor n\theta \rfloor$. By Lemma 2.1, as $n \to \infty$,

$$\frac{\mathbf{E}[R|W=t]}{n} \sim \begin{cases} \theta t & \text{if } t \le \dfrac{1-\theta}{\theta}, \\ 1-\theta & \text{if } t \ge \dfrac{1-\theta}{\theta}. \end{cases} \tag{2.10}$$

Let $n \to \infty$ in (2.9). Since $0 \le \mathbf{E}[R|W=t] \le n$, the dominated convergence theorem along with (2.10) implies (2.8). □

The expression for

$$L(\theta) \equiv \lim_{n\to\infty} \frac{F(n, \lfloor n\theta \rfloor)}{n}$$

is easy to minimize if $C(t)$ has a continuous density $c(t), t \ge 0$. In this case, let $\theta = 1/(1+u)$ and set

$$\hat{L}(u) = L\left(\frac{1}{1+u}\right), u = \frac{1-\theta}{\theta}, 0 \le u < \infty.$$

Then (2.8) becomes

$$\hat{L}(u) = 1 - \frac{1}{1+u}\int_0^u t\, dC(t) - \frac{u}{1+u}[1 - C(u)], \tag{2.11}$$

where $\hat{L}(0) = \hat{L}(\infty) = 1$ and $\hat{L}(u) > 0$ for all $u \ge 0$. We are assuming that $C(t)$ has a continuous derivative $C'(t) = c(t)$ for $t \ge 0$, so

$$\hat{L}'(u) = \frac{1}{(1+u)^2}\left(\int_0^u tc(t)dt - [1 - C(u)]\right). \tag{2.12}$$

Now $\int_0^u tc(t)dt$ is monotone increasing from zero to $\mathbf{E}[W]$ and $1 - C(u)$ is monotone decreasing from 1 to zero. Hence $\hat{L}(u)$ has a unique minimum at $u = u_0$, where $\hat{L}'(u_0) = 0$. The above analysis leads to the following consequence of Theorem 2.1.

COROLLARY 2.1. *If W has a continuous density* $c(t)$, *then* $L(\theta)$ *has a unique minimum at* $\theta_0 = 1/(1+u_0)$, *where* u_0 *is the solution to*

$$\int_0^{u_0} tc(t)dt = [1 - C(u_0)]. \tag{2.13}$$

Hence, by (2.8),

$$L(\theta_0) = C\left(\frac{1-\theta_0}{\theta_0}\right) = C(u_0).$$

For distributions C without continuous densities, it may still be possible to obtain $\inf_{0\le\theta\le 1} L(\theta)$. An important example, to which we return in §5, occurs when W is a constant.

Trivially, the constant=k policy supplies the upper bound $E(n) \le F(n, \lfloor n\theta_0 \rfloor)$, where θ_0 minimizes $L(\theta)$ and $E(n)$ is the expected cost of an optimal policy. This fact is exploited in §5, where the asymptotic optimality of the constant-k policy is proved for the case of constant clock times.

3. Exponential clock. This section shows that optimal policies are constant-k policies when W is exponentially distributed. We let $\mu > 0$ be the parameter of the distribution, $C(t) = 1 - e^{-\mu t}, t \ge 0$. Before getting into the analysis we introduce two problem reductions.

First, it is easy to see that we may confine ourselves to policies that always assign a waiting job, if any, to a processor that has just completed a job, i.e., there is always an optimal policy in the class defined by this property. This observation applies no matter what clock and service time distributions are assumed. But under our exponential assumptions, if we ignore events with probability zero (in particular, simultaneous job completions), it is clear that this "replenishing" property is in fact mandatory in an optimal policy whenever there are two or more waiting jobs. If there is only one job waiting at the time some processor completes a job, then the waiting job can be assigned to this processor immediately or at any subsequent time no later than W.

Our second observation exploits the memoryless property of the exponential distributions for W and T, and it states that the decisions of an optimal policy can be restricted to job completion times. Furthermore, for the Markov decision process underlying our optimization problem, a sufficient state is (n, k), where n is the number of unfinished jobs and k is the number of jobs currently assigned to processors. Based on this fact, it is easy to develop a Bellman equation for the cost $E(n, k)$ of an optimal policy starting in state (n, k). When in state $(n, k), 1 \le k < n$, either initially or at some job completion time, an optimal policy can assign a waiting job to an available machine and then proceed optimally, or it can wait until the next job departure or until the bell rings, whichever comes first. As part of the first choice, the term "proceed optimally" includes those situations where $j > 1$ jobs are allocated before continuing to run jobs in state $(n-j, k+j)$. If the second choice is made, a job will depart first with probability $k/(k + \mu)$, while the bell will ring first with probability $\mu/(k + \mu)$. If a job leaves first, then the machine on which the completion occurs is replenished with a waiting job, if any. Then for any given $n \ge 1$ we can write

$$\begin{aligned} &E(n,k) = \min\left\{E(n,k+1), \frac{k}{k+\mu}E(n-1,k) + \frac{\mu n}{k+\mu}\right\}, \qquad 1 \le k < n < \infty \\ &E(n,n) = n, \end{aligned} \tag{3.1}$$

where the first and second terms in braces account for the first and second choices, respectively. A routine induction on $n - k$ shows that (3.1) has a unique solution for any $n \ge 1$. The cost of an optimal policy is computed from $E(n) = \min_{1\le k\le n} E(n, k)$.

With respect to (3.1) the following terminology is useful. We say that state (n, k) is *stable* if and only if

$$E(n,k) = \frac{k}{k+\mu}E(n-1,k) + \frac{\mu}{k+\mu}n < E(n,k+1). \tag{3.2}$$

That is, an optimal policy in state (n,k) must continue running jobs in that state. State (n,k) is *unstable* if and only if

$$E(n,k) = E(n,k+1) < \frac{k}{k+\mu}E(n-1,k) + \frac{\mu}{k+\mu}n, \tag{3.3}$$

so that an optimal policy must assign at least one more job before continuing to run jobs. State (n,k) is *semistable* if and only if

$$E(n,k) = E(n,k+1) = \frac{k}{k+\mu}E(n-1,k) + \frac{\mu}{k+\mu}n, \tag{3.4}$$

so an optimal policy in state (n,k) has both of the above options.

It is appropriate to define the state (n,n) as stable, since no new assignments are possible in this state. Note also that (3.2) and $E(n,n) = n$ imply that $(n, n-1), n > 1$ is stable, i.e., the worst possible cost $(n > 1)$ is never incurred unless forced by the bell.

Our objective is a result (Theorem 3.1) that not only determines an optimal policy, but also decides the uniqueness question and classifies each state as stable, semistable, or unstable. For this purpose we need three lemmas. The existence of an optimal constant-k policy follows easily from the first lemma, as the second lemma will show.

LEMMA 3.1. *If $k \le n-1$ and state (n,k) is stable or semistable, then $(n-1,k)$ is stable.*

Proof. The result is immediate for $k = n-1$; so assume that $k < n-1$. We first observe that (3.2), (3.4), and the assumptions of the lemma imply

$$E(n,k) = \frac{k}{k+\mu}E(n-1,k) + \frac{\mu}{k+\mu}n \le E(n,k+1). \tag{3.5}$$

From the Bellman equation (3.1),

$$E(n,k+1) \le \frac{k+1}{k+1+\mu}E(n-1,k+1) + \frac{\mu}{k+1+\mu}n. \tag{3.6}$$

Now substitute the right-hand side of (3.6) in (3.5) and rearrange to obtain

$$E(n-1,k+1) \ge \frac{1}{(k+1)(k+\mu)}[k(k+1+\mu)E(n-1,k) + \mu n]. \tag{3.7}$$

But $n > E(n-1,k)$. So on substitution into (3.7), we get $E(n-1,k+1) > E(n-1,k)$, which means that $(n-1,k)$ is stable. □

Next, we relate $E(n,k)$ to $F(n,k)$, defined in §2 for the constant-k policy. Observe that $E(n,k) \le F(n,k)$ for all n and k, by definition of $E(n,k)$.

LEMMA 3.2. (a) *If (n,k) is stable or semistable, then $E(n,k) = F(n,k)$, and, furthermore, the constant-k policy is the unique optimal policy among those policies which begin running jobs in state (n,k).*

(b) *Conversely, if $E(n,k) = F(n,k)$, then (n,k) is either stable or semistable.*

(c) $E(n,k) = \min_{k\le j\le n} F(n,j)$.

Proof. To prove part (a), consider an optimal policy for any fixed n. Clearly, the policy must begin by running some subset of $l > 1$ jobs, where the corresponding state (n,l) is stable or semistable. Thereafter, provided the bell does not ring, the next state transition will occur at a job completion and have the form $(n,l) \to (n-1,l)$, where we have accounted for the fact that the machine just finishing a job is replenished. But state

$(n-1, l)$ is stable by Lemma 3.1, so the policy must continue running jobs in this state with one fewer waiting job. Repeating this reasoning shows that the decisions of the given optimal policy must be precisely those of a constant-k policy with $k = 1$. Part (a) follows easily. For part (b), we may assume that $k < n$. We are assuming $E(n, k) = F(n, k)$, so we can write

$$\frac{k}{k+\mu}E(n-1,k)+\frac{\mu}{k+\mu}n$$
$$\geq E(n,k) = F(n,k) = \frac{k}{k+\mu}F(n-1,k)+\frac{\mu}{k+\mu}n \geq \frac{k}{k+\mu}E(n-1,k)+\frac{\mu}{k+\mu}n,$$

and hence

$$E(n,k) = \frac{k}{k+\mu}E(n-1,k)+\frac{\mu}{k+\mu}n.$$

But this means (n, k) is stable or semistable so part (b) is proved.

For part (c) we may again assume $k < n$. Define the integer l such that $E(n, k) = E(n, k+1) = \cdots = E(n, l) < E(n, l+1)$ where $k \leq l < n$. Thus (n, l) is stable. By part (a),

$$E(n,k) = E(n,l) = F(n,l) \geq \min_{k\leq j\leq n} F(n,j). \tag{3.8}$$

For fixed n and $k \leq j \leq n$, let $F(n, j)$ attain its minimum at $j = j_0$. One possible policy starting out from state (n, k) is to transit to (n, j_0) and then apply the *constant-*j_0 policy. The expected cost of this policy is $F(n, j_0)$, so that

$$E(n,k) \leq F(n,j_0) = \inf_{k\leq j\leq n} F(n,j). \tag{3.9}$$

Inequalities (3.8) and (3.9) give $E(n, k) = \min_{k\leq j\leq n} F(n, j)$ as desired. □

To deal with the uniqueness problem, we analyze an explicit formula for $F(n, k)$. Choosing $\alpha(s)$ of (2.5) to have unit mass concentrated at $s = \mu$, so as to obtain the density $c(t) = \mu e^{-\mu t}, t \geq 0$, we find easily from (2.6),

$$F(n,k) = n - \frac{k}{\mu}\left[1-\left(\frac{k}{k+\mu}\right)^{n-k}\right], \qquad 1 \leq k \leq n. \tag{3.10}$$

Let

$$k_n = \max\{1 \leq k \leq n : F(n,k) = F(n)\} \tag{3.11}$$

denote the largest value of k for which $F(n, k)$ attains its minimum as a function of k with n held fixed. From (3.10), we obtain $F(n, k) < F(n, n)$ for $1 \leq k \leq n-1$, so that $k_n < n$ for $n > 1$.

LEMMA 3.3. *For fixed n, $F(n, k)$ is unimodal in k and attains its minimum at either one or two consecutive values of k, i.e., at just k_n, or at just k_n and $k_n - 1$.*

Proof. Replace k in (3.10) by the continuous variable x, $1 \leq x \leq n$, and with n fixed, write $F(x) = n - f(x)/\mu$, where

$$f(x) = x\left[1-\left(\frac{x}{x+\mu}\right)^{n-x}\right]. \tag{3.12}$$

We show that $f''(x) < 0$ and hence $F''(x) > 0, 1 \leq x \leq n$; the lemma then follows easily. Let $h(x) = (n-x)\log x/(x+\mu)$ so that $f(x) = x[1-e^{h(x)}]$. To prove that $f''(x) < 0$, it is enough to show that $2h'(x) + xh''(x) > 0$ in

$$f''(x) = -e^{h(x)}[2h'(x) + x(h'(x))^2 + xh''(x)].$$

Routine calculations give

$$2h'(x) + xh''(x) = \frac{(n-x)\mu^2}{x(x+\mu)^2} - 2\left\{\log\left(1 - \left[\frac{\mu}{x+\mu}\right]\right) + \frac{\mu}{x+\mu}\right\}.$$

The first term is clearly nonnegative for $1 \leq x \leq n$, and the bracketed term is negative, since $\log(1-z) + z < 0$ for $z = \mu/(x+\mu) < 1$. Thus $f''(x) < 0$ and $F''(x) > 0$, $1 \leq x \leq n$. □

We are now in position for the main result.

THEOREM 3.1. (a) *For any* $n \geq 1$,

$$(3.13) \qquad E(n,k) = \begin{cases} F(n,k) & \text{if } k \geq k_n, \\ F(n,k_n) & \text{if } k \leq k_n. \end{cases}$$

(b) (n,k) *is stable for* $k \geq k_n$ *and unstable for* $k \leq k_n - 2$. *If, for fixed* n, $F(n,k)$ *has a unique minimum at* $k = k_n \geq 2$, *then* $(n, k_n - 1)$ *is also unstable; otherwise it is semistable.*

(c) *If, for fixed* n, $F(n,k)$ *has a unique minimum at* k_n, *then the constant-k policy with* $k = k_n$ *is the unique optimal policy. Otherwise there are exactly two optimal policies, viz, the constant-k policies with* $k = k_n$ *and* $k = k_n - 1$.

Proof. Part (a) follows directly from Lemmas 3.2 and 3.3. To prove part (b), we may assume that $k < n$. For $k \geq k_n$, we have $E(n,k) = F(n,k) < F(n,k+1) = E(n,k+1)$. Here (n,k) is stable for $k \geq k_n$. For $k \leq k_n - 2, E(n,k) < F(n,k)$. By Lemma 3.2(a), (n,k) is thus unstable. Similarly, if $F(n,k)$ attains a unique minimum at k_n, then $E(n,k_n-1) < F(n,k_n-1)$, and so (n,k_n-1) is also unstable. Otherwise, $E(n,k_n-1) = F(n,k_n-1)$ and by Lemma 3.2(b) (n,k_n-1) is either stable or semistable. As $E(n,k_n-1) = E(n,k_n)$, we conclude that (n,k_n-1) is semistable.

For part (c), we observe that the initial states for which $E(n)$ is attained must either be stable or semistable. Hence by part (b) and Lemma 3.2(a), the only optimal policies executing initially in these states are the corresponding constant-k policies.

4. Analysis of k_n. This section studies k_n as a function of both n and μ. We begin by fixing μ and describing the behavior of k_n as n increases.

THEOREM 4.1. k_n *is an increasing function of* n.

Proof. State (n,k_n) is stable and, by Lemma 3.1, so is $(n-1,k_n)$. It follows from Theorem 3.1(b) that $k_n \geq k_{n-1}$. □

Corollary 2.1, together with a uniform version of Theorem 2.1, furnishes asymptotic estimates for the present model. With $C(t) = 1 - e^{-\mu t}, t \geq 0$, these results imply that as $n \to \infty, k_n \sim \mu/(\mu + x(\mu))n$ and $E(n) \sim [1 - e^{-x(\mu)}]n$, where $x(\mu)$ is the unique positive root of $e^x - x - 1 = \mu$. We do not present the details of the proof, as a more detailed analysis is possible for this case which yields the following stronger result.

THEOREM 4.2. *As* $n \to \infty$,

$$(4.1) \qquad k_n = \frac{\mu}{\mu + x(\mu)}n + O(1)$$

and

$$E(n) = [1 - e^{-x(\mu)}]n + O(1). \tag{4.2}$$

Proof. For given n, k_n is the largest k for which

$$H(n,k) \equiv \mu[n - F(n,k)] = k\left[1 - \left(\frac{k}{k+\mu}\right)^{n-k}\right], \qquad 1 \le k \le n,$$

attains its maximum. Let $H(n, n\theta) = nf_n(\theta)$, where $k = n\theta$ and

$$f_n(\theta) = \theta\left[1 - \left(1 + \frac{\mu}{n\theta}\right)^{n(\theta-1)}\right], \qquad 0 < \theta \le 1.$$

Then k_n may be expressed as $k_n = n\theta_n + O(1)$, where $f_n'(\theta_n) = 0$ defines θ_n.

Write

$$f_n(\theta) = \theta\left[1 - e^{n(\theta-1)\log(1+(\mu/n\theta))}\right],$$

and then expand $\log(1 + (\mu/n\theta))$ in powers of $\mu/(n\theta)$. It is easily verified that, for any constant $\gamma, 0 < \gamma < 1$,

$$f_n(\theta) = f(\theta) + O\left(\frac{1}{n}\right), \quad f_n'(\theta) = f'(\theta) + O\left(\frac{1}{n}\right) \tag{4.3}$$

uniformly on $[\gamma, 1]$, where

$$f(\theta) = \theta[1 - e^{-h(\theta)}], \quad h(\theta) = \mu(\theta^{-1} - 1). \tag{4.4}$$

Differentiation formulas give $f'' > 0$ on $(0, 1]$ with $f'(0+) > 0$, and $f'(1) < 0$. Hence $f'(\theta) = 0$ has a unique root in $(0, 1]$, say, at θ_*. Choose $0 < \gamma < \theta_*$. By (4.3), for n sufficiently large, $0 < \gamma < \theta_n$ and $f'(\theta_*) - f'(\theta_n) = O(1/n)$. An application of the mean value theorem to f' then gives $\theta_n = \theta_* + O(1/n)$; so from $k_n = n\theta_n + O(1)$, we get $k_n = n\theta_* + O(1)$. By (4.4), $\theta_* = \mu/[\mu + h(\theta_*)]$; so

$$f'(\theta_*) = 1 - e^{-h(\theta_*)} + \theta_* h'(\theta_*)e^{-h(\theta_*)} = 1 - \left(1 + \frac{\mu}{\theta_*}\right)e^{-h(\theta_*)} = 0 \tag{4.5}$$

implies $e^{h(\theta_*)} - h(\theta_*) - 1 = \mu \Rightarrow h(\theta_*) = x(\mu)$. Together with $k_n = n\theta_* + O(1)$, this proves (4.1).

Substituting $k_n = n\theta_* + O(1)$ into (4.3) shows that

$$H(n, k_n) = nf_n\left(\frac{k_n}{n}\right) = nf_n\left(\theta_* + O\left(\frac{1}{n}\right)\right) = n\left[f(\theta_*) + O\left(\frac{1}{n}\right)\right];$$

so

$$\begin{aligned} E(n) &= n - \frac{1}{\mu}H(n,k_n) = \left[1 - \frac{f(\theta_*)}{\mu}\right]n + O(1) \\ &= \left[1 - \frac{\theta_*}{\mu}(1 - e^{-h(\theta_*)})\right]n + O(1). \end{aligned} \tag{4.6}$$

By (4.5), $1 - e^{-h(\theta_*)} = \mu e^{-h(\theta_*)}/\theta_*$; so on substitution into (4.6) we prove (4.2). □

Finally, we describe k_n as a function of μ. For $n = 1, 2$, we readily check that $k_n(\mu) = 1$; so we assume $n > 2$. First, we need the following property of

$$H(n,k,\mu) = k\left[1 - \left(\frac{k}{k+\mu}\right)^{n-k}\right].$$

LEMMA 4.1. *For given n and $1 \le k \le n-2$, the equation $H(n,k,\mu) = H(n,k+1,\mu)$ has exactly one root. Denoting this root by μ_k, we have*

$$H(n,k,\mu) > H(n,k+1,\mu), \qquad 0 < \mu < \mu_k,$$
$$H(n,k,\mu) < H(n,k+1,\mu), \qquad \mu > \mu_k.$$

Proof. Let $H'(n,k,\mu)$ denote the derivative with respect to μ. Then

$$H'(n,k,\mu) = \left(\frac{k}{k+\mu}\right)^{n-k+1}(n-k) > 0. \tag{4.7}$$

From the formulas for H and H',

$$H(n,k,0) = 0, \quad H(n,k,\infty) = k, \quad H'(n,k,0) = n-k. \tag{4.8}$$

We conclude that $D(n,k,\mu) = H(n,k,\mu) - H(n,k+1,\mu), 1 \le k \le n-2$, is positive for all μ sufficiently small and negative for all μ sufficiently large. Let μ_k denote the smallest zero of D. Then $D(n,k,\mu) > 0, 0 < \mu < \mu_k$, and it remains to show that $D(n,k,\mu) < 0, \mu > \mu_k$. For $\mu > \mu_1$, the function $(x+\mu_k)/(x+\mu)$ is strictly increasing in $x > 0$; so

$$\frac{H'(n,k+1,\mu)}{H'(n,k+1,\mu_k)} = \left(\frac{k+1+\mu_k}{k+1+\mu}\right)^{n-k} > \left(\frac{k+\mu_k}{k+\mu}\right)^{n-k+1} = \frac{H'(n,k,\mu)}{H'(n,k,\mu_k)}.$$

Also $D(n,k,\mu) > 0$ on $(0,\mu_k)$ implies $D'(n,k,\mu_k) \le 0$, which means

$$H'(n,k+1,\mu_k)/H'(n,k,\mu_k) \ge 1;$$

so

$$H'(n,k+1,\mu) > \frac{H'(n,k+1,\mu_k)}{H'(n,k,\mu_k)}H'(n,k,\mu) \ge H'(n,k,\mu),$$

whereupon an integration yields $D(n,k,\mu) < 0$ for $\mu > \mu_k$. □

THEOREM 4.3. *Let $\mu_k, 1 \le k \le n-2$, be defined as in Lemma 4.1, and define $\mu_0 = 0, \mu_{n-1} = \infty$. Then $\mu_0 < \mu_1 < \mu_2 < \cdots < \mu_{n-2} < \mu_{n-1}$ and in $(\mu_{k-1}, \mu_k), 1 \le k \le n-1$, we have $k_n(\mu) = k$, i.e., as μ increases, $k_n(\mu)$ increases from 1 to $n-1$ and assumes all intermediate values.*

Proof. Fix n and let $1 \le k \le n-1, 0 < \mu < \infty$. Since $F(n,k,\mu) = n - H(n,k,\mu)/\mu$, we conclude from (3.10) and (4.8) that $F(n,1,\mu) < F(n,2,\mu) < \cdots < F(n,n-1,\mu)$ for all μ sufficiently small, and $F(n,1,\mu) > F(n,2,\mu) > \cdots > F(n,n-1,\mu)$ for all μ sufficiently large. Thus, $k_n(\mu) = 1$ and $k_n(\mu) = n-1$ for all μ sufficiently small and large, respectively. This gives $F(n,1,\mu) < \min_{2 \le k \le n-1} F(n,k,\mu)$ for μ sufficiently small and the reverse inequality for μ sufficiently large. Let $\hat{\mu}_1$ be the smallest μ for which

$F(n,1,\mu) = \min_{2\leq k\leq n-1} F(n,k,\mu)$, so that $k_n(\mu) = 1, 0 < \mu < \hat{\mu}_1$. By Lemma 3.3 we must then have $F(n,1,\hat{\mu}_1) = F(n,2,\hat{\mu}_1) < \cdots < F(n,n-1,\hat{\mu}_1)$. Together with Lemma 4.1, this implies $\hat{\mu}_1 = \mu_1$ and $F(n,1,\mu) > F(n,2,\mu), \mu > \mu_1$, and thus for $\mu > \mu_1$, that k rendering $F(n,k,\mu)$ smallest will be at least 2, and exactly 2 near μ_1. Repeating the above reasoning yields the theorem. □

We remark that for given n and $\mu \in (\mu_{k-1}, \mu_k), F(n,j,\mu)$ assumes a minimum at $j = k$ only. For such μ, the constant-k policy is the only optimal policy. For $\mu = \mu_k, 1 \leq k \leq n-2, F(n,j,\mu)$ assumes a minimum at $j = k, k+1$. In this case, there are exactly two optimal policies, viz the constant-k and $k+1$-policies.

5. Constant clock. It is easy to anticipate that the constant-clock case is likely to be more difficult than the exponential-clock case. We only have to observe that (n,k) is no longer a sufficient state for the Markov decision process; with a constant clock we need in addition to (n,k) an indication of how long the clock has to run, or equivalently, how long it has already run. However, we can describe the asymptotic behavior of an optimal policy. We do this by proving a lower bound on the expected cost incurred by any policy, and then observing that the expected cost of the best constant-k policy converges to this lower bound as $n \to \infty$. Thus, while the constant-k policies are not optimal in general, they are asymptotically optimal in the above sense.

After the asymptotic result we will present conjectures and partial results for the general problem. The analysis will identify what lies at the heart of the open optimization problem. This material is further motivated by the fact that the numerical evaluation of the Bellman equation (see (5.6)) gives convincing evidence that the conjectures are true.

A lower bound. Let $C(t)$ be an arbitrary distribution, and define the function

$$\Phi(u) = \frac{C(u) - \int_u^\infty t\,dC(t)}{1+u}, \qquad u \geq 0. \tag{5.1}$$

It is easy to see that $\Phi(u)$ is bounded and positive for all u sufficiently large, so $\sup_{u\geq 0} \Phi(u) > 0$. The result below is our sole generalization to other than exponential service times.

THEOREM 5.1. *Let the distribution of T have a monotone nondecreasing hazard-rate function, and assume as before that $E[T] = 1$. Then $E(n)/n \geq \sup_{u\geq 0} \Phi(u)$.*

Proof. Let M denote the maximum number of processors used throughout the schedule. An area argument yields the inequality

$$\sum_{i=1}^{n} T_i \leq MW + \sum_{i=1}^{K} T_i',$$

where $K \leq M$ is the number of active machines just before the bell rings, and where the $T_i', 1 \leq i \leq K$, denote the residual times of the jobs running on these machines ($\sum_{i=1}^K T_i'$ is defined to be zero when $K = 0$). Taking expectations,

$$n = E\left[\sum_{i=1}^{n} T_i\right] \leq E[MW] + E\left[\sum_{i=1}^{K} T_i'\right]. \tag{5.2}$$

To simplify (5.2), we prove next that

$$E\left[\sum_{i=1}^{K} T_i'\right] \leq E[K],$$

a result that would be trivial if K were independent of T_i' for all i. But the inequality is easy to prove in any case. Let $\mathcal{X}(A)$ be the indicator function of event A, and define $\mathcal{X}_k = \mathcal{X}(K = k), 0 \le k \le n$. Also, let $T_i'', 1 \le i \le K$, be the elapsed time of the job whose remaining time is T_i'. Then

$$(5.3)\qquad E\left[\sum_{i=1}^{K} T_i'\right] = E\left[\sum_{k=1}^{n} \mathcal{X}_k \sum_{i=1}^{k} T_i'\right] = \sum_{k=1}^{n}\sum_{i=1}^{k} E[\mathcal{X}_k T_i']$$

and

$$E[\mathcal{X}_k T_i'] = EE[\mathcal{X}_k T_i' | \mathcal{X}_k, T_i''] = E[\mathcal{X}_k E(T_i' | \mathcal{X}_k, T_i'')].$$

But given T_i'', T_i' is independent of $\mathcal{X}_k$; so $E[\mathcal{X}_k T_i'] = E[\mathcal{X}_k E(T_i' | T_i'')]$. Also, $E[T_i' | T_i''] \le E[T] = 1$, since T's distribution is hazard-rate-nondecreasing. Then $E[\mathcal{X}_k T_i'] \le E[\mathcal{X}_k]$, whereupon substitution into (5.3) gives

$$E\left[\sum_{i=1}^{K} T_i'\right] \le \sum_{k=1}^{K} kE[\mathcal{X}_k] = E[K],$$

as desired.

Since $K \le M$, we can now return to (5.2) and write

$$(5.4)\qquad n \le E[MW] + E[M] = E[M(W+1)].$$

But for any $u \ge 0$, we have

$$(5.5)\qquad \begin{aligned} E[M(W+1)] &= E[M(W+1)\mathcal{X}(W \le u)] + E[M(W+1)\mathcal{X}(W > u)] \\ &\le (1+u)E[M] + n\int_u^{\infty} (1+t)dC(t). \end{aligned}$$

The desired bound on $E(n) = E[M]$, valid for optimal schedules, follows easily from (5.1), (5.4), and (5.5). □

It follows at once from Theorem 5.1 that $E(n) \le n$ is bounded from below by a linear function of n for all n sufficiently large. Returning to our exponential service-time distribution, we can now prove the optimality result promised earlier.

THEOREM 5.2. *Let the clock time be a constant s. Then the constant-k policy is asymptotically optimal in the sense that, with*

$$k = \left\lfloor \frac{n}{1+s} \right\rfloor,$$

$$\frac{n}{1+s} \le E(n) \le F(n,k) = \frac{n}{1+s} + o(n) \text{ as } n \to \infty.$$

Proof. The lower bound follows from Theorem 5.1. The asymptotic expression for $F(n,k)$ is obtained directly from Theorem 2.1, since the right-hand side of (2.8) equals $1/(1+s)$ when $\theta = 1/(1+s)$. □

The general problem. Assuming a constant clock time, and exponentially distributed service times T with $E[T] = 1$, a Markov decision process can be defined on the set of states (n, k, s), where n and k have their earlier meanings and where s is the time remaining on the clock. Define $E(n, s) = \min_{1\leq k\leq n} E(n, k, s)$ as the expected cost incurred by an optimal policy in scheduling n jobs with s time units on the clock and all machines initially available. $E(n, k, s)$ can be computed from the following Bellman equation:

(5.6)
$$E(n,k,s) = \inf_{0\leq t\leq s}\left\{e^{-kt}E(n,k+1,s-t) + \int_0^t ke^{-ku}E(n-1,k,s-u)du\right\}, \qquad k < n,$$
$$E(n,n,s) = n.$$

The second of these equations is obvious. The recurrence in the first equation is derived as follows. In state (n, k, s) the scheduler must choose one of a continuum of actions $P_t, 0 \leq t \leq s$. P_t means: Begin the k jobs on k machines at time zero. If none of these jobs finishes by time t, then start a new job on a $(k+1)$th machine and proceed optimally thereafter. If at least one job finishes by time t, say this occurs first at time u, then replenish the just finishing machine with a new job and proceed optimally thereafter. The expected number of machines used under P_t is the expression inside the brackets of (5.6). Minimizing over t yields the cost of the optimal policy.

The following elementary properties of $E(n, k, s)$ are intuitive and easily derived from the Bellman equation. We omit the proofs.

LEMMA 5.1. (a) $E(n, k, 0) = n, E(n, k, \infty) = k$.

(b) *$E(n, k, s)$ is nondecreasing in k for fixed n and s, strictly increasing in n for fixed k and s, and strictly decreasing in s for fixed n and $k < n$.*

Graphs of $E(n, k, s)$ suggested by numerical evaluations of the Bellman equation are sketched in Fig. 1. The key properties of the curves in Fig. 1 are expressed in the following assertion.

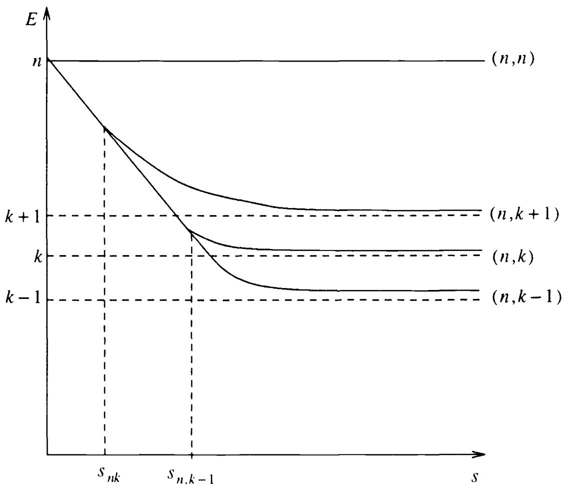


FIG. 1. *Graph of $E(n, k, s)$.*

CONJECTURE. *For each $n > 1$ there exist nonnegative numbers $s_{nk}, 1 \leq k \leq n-1$, such that*

(5.7) $E(n, k+1, s) = E(n, k, s), s \leq s_{nk}$; $E(n, k+1, s) > E(n, k, s)$, $s > s_{nk}$, (5.8) $$s_{n,n-1} < s_{n,n-2} < \cdots < s_{n1}.$$

Since a state in which $k = n-1$ must be stable, as in §3, we have $s_{n,n-1} = 0$. Unfortunately, we have been unable to prove this assertion; it remains a conjecture supported by numerical evaluations of (5.6). The remainder of this section proves the following result, which shows how (5.7) and (5.8) determine the structure of an optimal policy.

THEOREM 5.3. *The relations in* (5.7) *and* (5.8) *imply*

(5.9) $$s_{n-1,k} < s_{n,k}, \quad 1 \le k \le n-2,$$

and hence the following structure of an optimal policy.

Let the initial state have $n > 1$ jobs to be scheduled, an unlimited number of machines available, and a clock time s such that $s_{nk} < s \le s_{n,k-1}$ for some $1 \le k \le n-1$. Then assign jobs to k machines and start running jobs in state (n, k, s). Continue running jobs in any given state until one of the following three events occur: (1) *The bell rings: in this case assign the $n-k \ge 0$ waiting jobs to new machines.* (2) *A machine completes a job in some state (n', k', s'), $k' < n'$; in this case replenish the machine and continue in state $(n'-1, k', s')$.* (3) *A state (n', k', s'), $k' < n'$, is reached in which the remaining clock time has reduced to $s' = s_{n'k'}$; in this case assign a waiting job to a new machine and continue in state $(n', k'+1', s')$.*

We remark that (2) is a direct consequence of Theorem 5.3 and (3) a direct consequence of the assertion made in (5.7), (5.8).

Before proving the theorem, we need a preliminary result. Define

(5.10) $$B(n,k,s) = e^{ks}E(n,k+1,s) - \int_0^s ke^{k\tau}E(n-1,k,\tau)d\tau.$$

Introducing $B(n,k,s)$ into (5.6) gives

(5.11) $$E(n,k,s) = E(n,k+1,s) - e^{-ks}\left[B(n,k,s) - \inf_{0\le t\le s} B(n,k,t)\right].$$

LEMMA 5.2. *The relations in* (5.7) *and* (5.8) *imply that $E(n,k,s)$ is continuously differentiable with respect to s for all $1 \le k < n < \infty$ and $0 < s < \infty$. In addition,*

(5.12) $$E'(n,k,s) + kE(n,k,s) \le kE(n-1,k,s),$$

with equality for $s \ge s_{nk}$.

Proof. The proof is by induction on $n-k$, the number of waiting jobs. The assertions of the lemma are readily verified for $n-k \le 1$; so assume $n-k \ge 2$. Differentiating (5.10) we get

(5.13) $$B'(n,k,s) = e^{ks}[E'(n,k+1,s) + kE(n,k+1,s) - kE(n-1,k,s)].$$

In view of (5.11), we may rewrite (5.7) as

(5.14) $$B'(n,k,s) < 0 \text{ for } s < s_{nk}, \text{ and } B(n,k,s) > B(n,k,s_{nk}) \quad \text{for } s > s_{nk},$$

so that $B'(n,k,s_{nk}) = 0$. Then (5.13), (5.14), and the inductive hypothesis give

(5.15) $$E'(n,k,s) \le -kE(n,k,s) + kE(n-1,k,s) \quad \text{for } s \le s_{nk},$$

with equality at $s = s_{nk}$, where $E'(n, k, s)$ is interpreted as a left derivative at $s = s_{nk}$. By (5.11) and (5.14),

$$E(n,k,s) = E(n,k+1,s) - e^{-ks}[B(n,k,s) - B(n,k,s_{nk})]$$
$$= E(n,k,s_{nk})e^{-k(s-s_{nk})} + \int_{s_{nk}}^{s} ke^{-k(s-\tau)}E(n-1,k,\tau)d\tau, \qquad s \geq s_{nk}.$$

Differentiating, we get

$$(5.16) \qquad E'(n,k,s) = -kE(n,k,s) + kE(n-1,k,s), \qquad s \geq s_{nk},$$

where $E'(n, k, s)$ is interpreted as a right derivative at $s = s_{nk}$. The lemma follows from (5.15) and (5.16). □

Proof of Theorem 5.3. From (5.12) and (5.13) we obtain for $0 \leq s \leq s_{n-1,k}$,

$$B'(n,k,s) = e^{ks}[E'(n,k+1,s) + kE(n,k+1,s) - kE(n-1,k+1,s)]$$
$$\leq e^{ks}[(k+1)E(n-1,k+1,s) - (k+1)E(n,k+1,s)]$$
$$+[kE(n,k+1,s) - kE(n-1,k+1,s)]$$
$$= e^{ks}[E(n-1,k+1,s) - E(n,k+1,s)] < 0.$$

Thus $B(n, k, s)$ is strictly decreasing for $0 < s \leq s_{n-1,k}$, and we conclude from (5.11) that $s_{nk} > s_{n-1,k}$.

Finally, it is easily seen that the relations in (5.7)–(5.9) imply the optimality of the policy given in the theorem. □

6. Final remarks. The main contributions have been a solution to the stochastic machine minimization problem under exponential assumptions, a detailed analysis of the constant-k policy, and a proof that this policy is asymptotically optimal in the case of exponential service and hard deadlines. The most tantalizing open problem is finding a proof of the optimality of the policy defined in Theorem 5.3 for hard deadlines. The optimality of the constant-k policy for distributions $C(t)$ with nonincreasing hazard rates is also a conjecture worth investigating. However, in addition to more general distributional assumptions, there are many other interesting avenues of further research, e.g., machines of different speeds, and jobs with precedence constraints are useful extensions of the structure of the problem.

A special case of some importance is the assumption of constant service times. Here, it can be proved that the constant-k policy is optimal for all clock-time distributions. In general, the analysis is in the same spirit as that in §§3 and 4. However, the peculiarities of the problem and the greater generality of the result alter and extend the arguments in certain ways. As a result, a comprehensive analysis is well beyond the scope of this paper. The authors plan to include this analysis in a future publication.

REFERENCES

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

[2] M. Pinedo and L. Schrage, *Stochastic shop scheduling*: *A survey*, in Deterministic and Stochastic Scheduling, M. Dempster, J. K. Lenstra, and A. Rinooy-Kan, eds., D. Reidel, Dordrecht, Holland, 1982.

[3] R. R. WEBER, *Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime*, J. Appl. Prob., 19 (1982), pp. 167–182.

[4] G. WEISS, *Multiserver stochastic scheduling*, in Deterministic and Stochastic Scheduling, M. Dempster, J. K. Lenstra, and A. Rinooy-Kan, eds., D. Reidel, Dordrecht, Holland, 1982.

[5] G. WEISS AND M. PINEDO, *Scheduling tasks with exponential services times on nonidentical processors to minimize various cost functions*, J. Appl. Prob., 17 (1980), pp. 187–202.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 349–355, April 1993

AN ON-LINE SCHEDULING HEURISTIC WITH BETTER WORST CASE RATIO THAN GRAHAM'S LIST SCHEDULING*

GÁBOR GALAMBOS† AND GERHARD J. WOEGINGER‡

Abstract. The problem of on-line scheduling a set of independent jobs on m machines is considered. The goal is to minimize the makespan of the schedule. Graham's List Scheduling heuristic [R. L. Graham, *SIAM J. Appl. Math.*, 17(1969), pp. 416–429] guarantees a worst case performance of $2 - \frac{1}{m}$ for this problem. This worst case bound cannot be improved for $m = 2$ and $m = 3$. For $m \geq 4$, approximation algorithms with worst case performance at most $2 - \frac{1}{m} - \varepsilon_m$ are presented, where ε_m is some positive real depending only on m.

Key words. combinatorial problems, scheduling, worst case bounds, on-line algorithms

AMS(MOS) subject classifications. 90B35, 90C27

1. Introduction.

We consider the problem of on-line scheduling a list $\{J_1, \ldots, J_n\}$ of n jobs nonpreemptively on m identical machines $\{M_1, \ldots, M_m\}$. Each job J_i has a fixed processing time p_i. The jobs may be processed in any order. Our goal is to minimize the makespan, i.e., the maximum completion time over all jobs in a schedule. This problem obviously is NP-hard. So we are interested in heuristics that produce "rather" good approximate solutions. The quality of a heuristic H is measured by its worst case ratio

$$R^H(m) = \limsup\{C^H(L)/C^*(L) : L \text{ is a list of jobs}\}, \tag{1}$$

where $C^H(L)$ denotes the makespan produced by the heuristic on m machines and the list L of jobs and $C^*(L)$ denotes the corresponding makespan in some optimum schedule. The jobs are given to us *on-line*, that means we get the jobs one by one and must immediately decide by which machine the job is processed. Once a job is assigned to its machine, we are not allowed to move the job to another machine.

In 1969, Graham [2] suggested a simple heuristic to solve this on-line problem, *List Scheduling* (*LS* for short). This heuristic always assigns a job to the machine that has the minimum load at the moment. Graham showed that *LS* constructs a schedule with makespan at most $2 - \frac{1}{m}$ times the optimum makespan. Until now there was no heuristic known with better worst case ratio. This was mentioned as an open problem by Faigle, Kern, and Turán [1].

In this paper, for $m \geq 4$ machines we present a heuristic that has a slightly better worst case performance guarantee than *LS*. Contrary to the simple formulation of Graham's heuristic, the behaviour of our heuristic depends on the number m of machines and the processing time of the job that has to be scheduled next. Thus, by exploiting more information than *LS* does, we are able to improve on the antique $2 - \frac{1}{m}$ worst case bound. For the average performance, numerical tests indicate that the gain in the worst case ratio is *not* paid by a loss in the average performance. For $m = 2$ and $m = 3$, List Scheduling cannot be beaten, as its worst case bounds are best possible. For $m \geq 4$, no on-line heuristic can guarantee a better worst case performance than $1 + \sqrt{2}/2 \approx 1.707$.

*Received by the editors May 22, 1991; accepted for publication (in revised form) October 4, 1991. This paper was supported by a grant from the Hungarian Academy of Sciences (OTKA Nr. 2037) and by the Christian Doppler Laboratorium für Diskrete Optimierung.

†József Attila University, Department of Applied Computer Sciences, Árpád tér 2, H-6720 Szeged, Hungary.

‡TU Graz, Institut für Theoretische Informatik, Klosterwiesgasse 32/II, A-8010 Graz, Austria.

The paper is organized as follows. Section 2 deals with lower bounds for on-line scheduling algorithms. Section 3 gives a number of basic definitions and presents our heuristic. Section 4 gives the worst case analysis of the heuristic and the discussion in §5 finishes the paper.

2. Lower bounds for on-line scheduling.

2. Lower bounds for on-line scheduling. All we do in this section is to prove the following theorem. This result is also contained in the discussion of [1].

THEOREM 2.1. *There is no on-line scheduling algorithm with worst case guarantee better than*

(i) $3/2$ *for* 2 *machines*,
(ii) $5/3$ *for* 3 *machines*,
(iii) $1+\sqrt{2}/2$ *for* $m \geq 4$ *machines.*

Proof. Claims (i) and (ii) are easily proved by using the list $\mathcal{L}_2 = \langle 1,1,2\rangle$ for $m=2$ and the list $\mathcal{L}_3 = \langle 1,1,1,3,3,3,6\rangle$ for $m=3$. The exact arguments are analogous to those used for $m \geq 4$ and omitted.

For $m \geq 4$, we consider a sequence consisting of m times a job of length 1, followed by m times a job of length $1+\sqrt{2}$ and ending with a single job of length $2+2\sqrt{2}$. If the algorithm puts two or more of the jobs with length 1 on the same machine, it does not get any other job. It produced a makespan of 2, whereas the optimum makespan is only 1 and we are finished. Otherwise, it puts one job with length 1 on every machine. If in the following the algorithm puts two or more of the jobs with length $1+\sqrt{2}$ on the same machine, we are finished again. Instead of an optimum makespan $2+\sqrt{2}$, it came up with a $3+2\sqrt{2}$ makespan. Consequently, every machine gets one job with processing time 1 and one with processing time $1+\sqrt{2}$ job. But now the single final job completes the proof. □

3. Some definitions and the heuristic.

3. Some definitions and the heuristic. We will denote by C^* the makespan in the optimal schedule and by C^H the makespan in the schedule constructed by our heuristic. As the algorithm gets the jobs one by one, the values of C^* and C^H vary during the algorithm. To simplify notation, we will identify each job with its length. The *load* L_i of a machine M_i is the sum of processing times over all jobs assigned to it. When we describe the algorithm, we will use $L^{\max}$ to denote the maximum load and $L^{\min}$ to denote the minimum load over all machines.

We assume that $m \geq 4$ holds (as for $m \leq 3$ List Scheduling cannot be beaten). To define the algorithm, we introduce two real numbers $\alpha(m)$ and $\beta(m)$, where $0 \leq \alpha(m) \leq 1/3$ and $1 \leq \beta(m) \leq 5/4$ holds. Moreover, we will assume that $(\beta(m)+1)/\beta(m)^3 > \alpha(m)+1$ holds. These two numbers depend on m and their exact value will be specified later. To simplify presentation we write α and β instead of $\alpha(m)$ and $\beta(m)$. For nonnegative reals x, y we define the symmetric relation

$$x \sim y \quad \Longleftrightarrow \quad \frac{y}{\beta} \leq x \leq \beta y, \tag{2}$$

and we say that in this case x is *similar* to y. For S a set of nonnegative reals, we say that $\sim(S)$ holds, if and only if every two elements in S are similar (equivalently we can say, $\sim(S)$ holds if and only if the smallest and the largest elements of S are similar to each other).

Before we present our heuristic, we want to give the reader some intuition of the underlying ideas. We begin with recapitulating the analysis of List Scheduling. Let $L_1 \leq L_2 \leq \ldots \leq L_m$ be the loads of the machines and let x be a new job to be scheduled.

We use $C^* \geq (\sum L_i + x)/m \geq L_1 + x/m$ and $C^* \geq x$ to derive

$$L_1 + x \leq C^* - x/m + x \leq \left(2 - \frac{1}{m}\right) C^*. \tag{3}$$

This inequality is tight if $C^* = x$ holds and all L_i are equal. To reach a better worst case performance, we must avoid at least one of these two conditions. We have no influence on the length of job x, consequently we try to keep the L_i as "unequal" as possible. To measure "unequality" of loads we apply the similarity relation introduced above. Unfortunately, it is not always possible to keep the loads unequal (for example if the first m jobs to be scheduled all have equal length). To circumvent this difficulty, we use a threshold α; if the quotient of smallest load to largest load is less than α, the next job may be put on the smallest machine. If α is chosen appropriately (with respect to β and m), we will show that after such actions the ratio C^H/C^* is not too large.

Finally, we are ready to present our heuristic called *Refined List Scheduling* (*RLS* for short).

```
(1) Reorder the machines such that $L_1 \leq L_2 \leq \ldots \leq L_m$ holds.
    Let $x$ be a new job given to the algorithm.
(2) If $\not\sim (L_1 + x, L_2, L_3, \ldots, L_m)$ then put $x$ on $M_1$ and goto (1).
(3) Else if $L_1 > \alpha L_m$ then put $x$ on $M_2$ and goto (1).
(4) Else if $L_1 \leq \alpha L_m$ then
    (4.1) Put $x$ on $M_1$.
    (4.2) While $L^{\min} \sim L^{\max}$ do: put all new jobs on $M_1$.
    (4.3) Goto (1).
```

The heuristic terminates when no more jobs are given to it. Observe that new jobs are given to the algorithm only in steps (1) and (4.2). In step (4.2) M_1 generally is *not* the machine with minimum load anymore; the heuristic is putting jobs on M_1 until it has the maximum load.

4. The worst case analysis. In this section we will derive a sequence of claims and lemmas that lead to a number of different worst case bounds. Having derived all these bounds, we will fix the values of α and β so as to minimize the maximum of all these bounds.

LEMMA 4.1. *Each time the heuristic reenters step* (1), $\not\sim (L_1, \ldots, L_m)$ *holds.*

Proof. The proof is a simple inductive argument. When the heuristic reenters step (1) the first time, there is only one nonempty machine and the claim follows. If the heuristic last performed step (2), the claim holds by definition; if it last performed step (3), the claim holds by induction, as step (3) does not increase $L^{\min} = L_1$. If it reenters from step (4), $L^{\min} \not\sim L^{\max}$ holds. □

LEMMA 4.2. *When the heuristic leaves step* (2), *the inequality*

$$(L_1 + x)/C^* < (2m - 2 + \beta)/(m - 1 + \beta) \tag{4}$$

holds.

Proof. By Lemma 4.1, before x was assigned to M_1 the machine loads were not similar. This implies

$$C^* \geq \left(\sum L_i + x\right)/m > \frac{1}{m}(m - 1 + \beta)L_1 + x/m, \tag{5}$$

where the first inequality follows from averaging over all loads and the second inequality uses $L_i \geq L_1$ for $i = 1 \ldots m-1$ and $L_m > \beta L_1$. Using $C^* \geq x$, we simply derive the claimed result. □

LEMMA 4.3. *When the heuristic leaves step* (3), *the inequality*

$$(L_2 + x)/C^* < \Big(m + (m-2)\beta - (m-1)\alpha\Big)/(m-1) \tag{6}$$

holds.

Proof. As L_2 is the second smallest load, it is less or equal to the average value of the $m-1$ largest loads. This gives

$$L_2 \leq (L_2 + \ldots + L_m)/(m-1) \leq (mC^* - L_1 - x)/(m-1). \tag{7}$$

Moreover, we show that

$$L_1 + x \leq \beta C^* \tag{8}$$

by considering the following two cases. If $L_2 > C^*$, $L_1 + x < C^*$ must hold (again by the standard averaging argument) and this implies the inequality because of $\beta \geq 1$. On the other hand, if $L_2 \leq C^*$ holds, we may use that $L_1 + x$ and L_2 are similar and derive the inequality. The two displayed inequalities together with $L_1 > \alpha L_m \geq \alpha C^*$ lead to

$$\begin{aligned} L_2 + x &\leq (mC^* - L_1 - x)/(m-1) + x \\ &= mC^*/(m-1) - L_1 + (L_1 + x)(m-2)/(m-1) \\ &< C^*\Big(m + (m-2)\beta - (m-1)\alpha\Big)/(m-1), \end{aligned} \tag{9}$$

and the proof is complete. □

LEMMA 4.4. *If $L^{\min} \sim L^{\max}$ holds in step* (4.2), *then*

$$L^{\max}/C^* \leq m\beta/(m-1+\beta). \tag{10}$$

Proof. By load averaging, $C^* \geq \sum L_i/m$ holds. Now we simply use $L_i \geq L^{\max}/\beta$ for $i \leq m-1$ and finish the proof. □

LEMMA 4.5. *Assume the algorithm enters step* (4). *Denote by a_i the job assigned last to machine M_i. Then for every $i \geq 2$, $a_i \geq (1-\alpha\beta^2)L_i$ must hold.*

Proof. Assume the contrapositive. Then we consider that machine M_j with $a_j < (1-\alpha\beta^2)L_j$ that got its a_j latest among all these contradicting machines. The job a_j was not assigned during step (2), as step (2) assigns jobs to only the smallest machine and by assumption

$$L_j - a_j > \alpha\beta^2 L_j \geq \alpha\beta L_m \geq L_1 \tag{11}$$

holds. Therefore, a_j was treated either in step (3) or in step (4). We will distinguish between these two cases. We typify loads at the moment before a_j is assigned by a prime, loads without prime concern the moment cited in the formulation of the claim. The two moments are typified analogously. Finally, in our arguments we will use a machine M_d with $M_1 \neq M_d \neq M_j$.

First we scrutinize the step (3) case. In this case, M_j must have been the second smallest machine when a_j was assigned to it. The only possible smaller machine at this moment is M_1. Moreover by the definition of M_j, no other machine received a job between the primed and the unprimed moment. At the primed moment, L'_j and L'_d are similar, and at the unprimed moment the loads are similar again. This means $\beta L'_j \geq L'_d$ and $\beta L'_d \geq L'_j + a_j$ and we derive

$$(\beta^2 - 1)L'_j \geq a_j. \tag{12}$$

At the primed moment $L'_1 > \alpha L^{\max}$ holds, at the unprimed moment $L_1 \leq \alpha L^{\max}$ holds. The load of M_1 does not decrease between these two moments, thus $L^{\max}$ must increase. The only possibility for this is $L^{\max} = L'_j + a_j$. This gives $L_1 \leq \alpha(L'_j + a_j)$. In addition, $L'_1 + a_j \geq L'_j/\beta$ holds, since the two numbers are similar. These two facts together with $L_1 \geq L'_1$ imply

$$(\alpha + 1)a_j \geq \left(\frac{1}{\beta} - \alpha\right) L'_j. \tag{13}$$

But now the two displayed inequalities contradict the assumption $(\beta + 1)/\beta^3 > \alpha + 1$.

Next we examine the step (4) case. As the heuristic entered step (4), $L_1 \leq \alpha L^{\max} \leq \alpha\beta L_j$ holds. At the primed moment, $L'_j \sim L'_1$ implies that $L'_1 \geq (L_j - a_j)/\beta$ holds. Using $L_1 \geq L'_1$, this leads to

$$a_j \geq (1 - \alpha\beta^2)L_j, \tag{14}$$

and our proof is complete. □

LEMMA 4.6. *When the heuristic leaves step* (4.2), *then*

$$L^{\max}/C^* \leq \beta^2/(2 - 2\alpha\beta^2) + 1. \tag{15}$$

Proof. Let a_i, $1 \leq i \leq m$ denote the last job assigned to M_i before the heuristic enters the while-loop (hence, a_1 is equal to x). As $L_1 + x \sim L_m$, we have $L_1 + x \geq L_m/\beta$. Together with Lemma 4.5 and $L_1 \leq \alpha L_m$, this gives

$$a_i \geq \left(\frac{1}{\beta} - \alpha\beta\right) L_m \tag{16}$$

for all $1 \leq i \leq m$. Finally, we define y to be the last job assigned to machine M_1 before the heuristic leaves step (4.2) and note that after this assignment $L^{\max}$ is the load of machine M_1.

We distinguish two cases. If $y \geq (\frac{1}{\beta} - \alpha\beta)L_m$ holds, we use the pigeon hole principle: There are $m + 1$ jobs with length greater or equal $(\frac{1}{\beta} - \alpha\beta)L_m$, hence the optimum schedule must put two of them on the same machine. This gives $C^* \geq (\frac{2}{\beta} - 2\alpha\beta)L_m$. Using $C^* \geq y$ and the fact that $L^{\max} - y \sim L_m$ we have

$$L^{\max} \leq \beta L_m + y \leq \beta^2 C^*/(2 - 2\alpha\beta^2) + C^*. \tag{17}$$

Otherwise, if $y < (\frac{1}{\beta} - \alpha\beta)L_m$ holds, we derive in an analogous way that $C^* \geq (\frac{1}{\beta} - \alpha\beta)L_m + y$ and $C^* \geq y$. This gives

$$L^{\max} - C^* \leq (\beta - \frac{1}{\beta} + \alpha\beta)L_m \leq (\beta^2 - 1 + \alpha\beta^2)L^{\min}. \tag{18}$$

But now $L^{\min} \leq C^*$ and $\beta^2 - 1 + \alpha\beta^2$ is smaller or equal to $\beta^2/(2 - 2\alpha\beta^2)$, if $0 \leq \alpha \leq 1/3$ and $1 \leq \beta \leq 5/4$. So we end in all cases with the claimed bound. □

TABLE 1
Approximate values of α and β for different m.

m	α	β	$R^{RLS}(m)$	$R^{LS}(m)$	ε_m	L.B.
2	—	—	—	1.50000	—	1.50000
3	—	—	—	1.66667	—	1.66667
4	0.27419	1.02837	1.74472	1.75000	0.0053	1.70710
5	0.25549	1.06111	1.79034	1.80000	0.0097	1.70710
6	0.24439	1.08295	1.82197	1.83333	0.0114	1.70710
7	0.23696	1.09863	1.84523	1.85714	0.0119	1.70710
8	0.23161	1.11047	1.86308	1.87500	0.0119	1.70710
9	0.22756	1.11975	1.87722	1.88889	0.0117	1.70710
10	0.22439	1.12722	1.88869	1.90000	0.0113	1.70710
...	...	...	...	...	...	...
100	0.20005	1.19009	1.98812	1.99000	0.0019	1.7071
∞	0.19743	1.19743	2.00000	2.00000	0.0000	1.7071

THEOREM 4.7. *For $m \geq 4$, the worst case performance of RLS is less than or equal to $2 - \frac{1}{m} - \varepsilon_m$, with ε_m some small positive real depending on m.*

Proof. Summarizing, Lemmas 4.2, 4.3, 4.4, and 4.6 give the following worst case bounds. In each inequality C^* denotes the momentary optimum makespan:

$$(19)\qquad (L_1 + x)/C^* < (2m - 2 + \beta)/(m - 1 + \beta),$$

$$(20)\qquad (L_2 + x)/C^* < \Big(m + (m-2)\beta - (m-1)\alpha\Big)/(m-1),$$

$$(21)\qquad L^{\max}/C^* \leq m\beta/(m - 1 + \beta),$$

$$(22)\qquad L^{\max}/C^* \leq \beta^2/(2 - 2\alpha\beta^2) + 1.$$

Let us examine the worst case performance of *RLS*. A worst case situation occurs after some job x was assigned to its machine. This happens either in step (2), in step (3), in the middle of step (4), or in the end of step (4); the four inequalities above give upper bounds on the worst case ratios in these four scenarios. Consequently, our goal is to minimize for each $m \geq 4$ the maximum over the four right-hand side values under the restrictions

$$(23)\qquad 0 \leq \alpha \leq 1/3, \qquad 1 \leq \beta \leq 5/4, \qquad (\beta + 1)/\beta^3 > \alpha + 1.$$

We denote this minimum (that is an upper bound on the worst case ratio of *RLS*) by $R^{RLS}(m)$. First, it is easy to see that for $\beta \leq 2$, the right-hand side of (21) is less than or equal to the right-hand side of (19) and so we need not consider it in the optimization problem. For the remaining three values, we derive by using standard calculus that the minimum is taken if all three values are equal. Some substitutions lead to

$$(24)\ \beta^4(2m-3) + 2\beta^3(m^2 - 2m + 2) + \beta^2(4m - m^2 - 3) - 2\beta(m-1) = 2(m-1)^2.$$

For $m \leq 3$ this equality does not have a feasible solution. For each $m \geq 4$, this equality has a solution with $1 < \beta < 5/4$, since its value taken for $\beta = 1$ is negative and its value for $\beta = 5/4$ is positive. The solutions for some small values of m are stated in the third column of Table 1. As m tends to infinity, β tends from below to the positive real root of $2\beta^3 - \beta^2 - 2 = 0$. The restriction $(\beta + 1)/\beta^3 > \alpha + 1$ that was essential in the proof of Lemma 4.5, is of no consequence for the minimization problem.

Finally, we determine from the right-hand side of inequality (19) the value of $R^{RLS}(m)$. As $\beta > 1$ holds, we have $R^{RLS}(m) < 2 - \frac{1}{m}$. This means that *RLS* beats the worst case performance of List Scheduling for $m \geq 4$. Table 1 summarizes some of the results of this section. $\square$

5. Discussion. In this paper we derived an on-line algorithm that beats Graham's List Scheduling in the measure of worst case performance (for $m \geq 4$ machines). For two or three machines, the worst case guarantee of List Scheduling is optimal.

However, asymptotically our analysis did not improve the heuristic LS, since ε_m tends to zero as m tends to infinity. So the first open problem is at hand. Are there on-line scheduling algorithms with worst case ratio smaller than $2 - \delta$ for all m and some fixed $\delta > 0$? Secondly, the following question may be interesting as well. For four machines, we gave a lower bound of 1.7071 and an upper bound of 1.7447. How can we narrow the gap between these two bounds?

Acknowledgement. We thank Hannes Hassler and Hans Kellerer for valuable discussions, and we thank Wolfgang Gutjahr for pointing out an error in an earlier version of this paper.

REFERENCES

[1] U. Faigle, W. Kern, and Gy. Turán, *On the performance of on-line algorithms for partition problems*, Acta Cybernet., 9 (1989), pp. 107–119.

[2] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 356–378, April 1993

THEORETICAL ASPECTS OF VLSI PIN LIMITATIONS*

ROBERT CYPHER†

Abstract. This paper presents a formal model of the pin requirements of a parallel computer. This model is then used to obtain bounds on the pin requirements of different parallel machines and on the tradeoffs between pins and time. Specifically, two new bounds are established on the relationship between pin requirements and the time needed to sort or permute data. This paper also gives new lower bounds on the pin requirements of mesh-connected, cube-connected-cycles, shuffle-exchange, and Ajtai–Komlós–Szemerédi (AKS) computers, and it gives new upper bounds on the pin requirements of shuffle-exchange and AKS computers. All of these bounds are tight to within a constant factor. Finally, the bounds on pin requirements are used to prove a tight lower bound on the time required to implement the AKS sorting algorithm on a shuffle-exchange or cube-connected-cycles computer.

Key words. pin limitations, VLSI theory, parallel sorting

AMS(MOS) subject classifications. 68Q35, 68P10, 68M10, 68Q05

1. Introduction. An enormous number of different interconnection networks have been proposed for distributed-memory parallel computers. These networks differ in both their ability to support various algorithms and in their hardware costs. One useful measure of hardware cost is the area required when the entire parallel computer is laid out on a single sheet of silicon. This measure has been well studied, and the very-large-scale integration (VLSI) area requirements of many interconnection networks are known. Also, tradeoffs between area and time for problems such as permuting and sorting data have been established [28], [29].

Actual parallel machines are typically laid out on a number of separate chips, each of which has a limited number of pins through which connections can be made to other chips. In some cases the number of pins available per chip is a more serious limitation than the amount of area available per chip [6], [10], [11], [15], [25]. As a result, the pin requirements of a parallel computer are another important measure of its cost. Many researchers have studied pin requirements but a theory comparable to that relating to VLSI area requirements has not been created.

A large amount of research has been devoted to studying the pin requirements of switch-based interconnection networks. For example, the pin requirements of the Omega network were studied by Ciminiera and Serra [6]; Snir [25]; Franklin and Dhar [10]; Franklin, Wann, and Thomas [11]; McMillen and Siegel [20]; Knauer, O'Neill, and Huang [15]; and Szymanski [27]. Other switch-based networks studied include the crossbar [10], [11], [27], the augmented data manipulator [20], and various partial concentrators [7]. Snir introduced a general model for VLSI pin requirements [25] and proved matching upper and lower bounds on the pin requirements of Omega networks, two-dimensional mesh-connected computers, and trees. Snir also proved a lower bound on the pin requirements of shuffle-exchange computers, which was subject to certain restrictions, and an upper bound on the pin requirements of mesh-connected computers of arbitrary dimension [25].

*Received by the editors April 29, 1991; accepted for publication (in revised form) January 7, 1992. A preliminary version of this work was presented at the 6th MIT Conference on Advanced Research in VLSI, Cambridge, Massachusetts, April 1990. This research was supported in part by a National Science Foundation Graduate Fellowship.

†Department K54/802, IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120.

A number of researchers have studied different but related problems. Ullman defined the notion of "flux" to capture the amount of locality in a network and he showed the importance of this measure to sorting and artificial intelligence problems [31]. Li and Maresca introduced a parallel computer called the "polymorphic torus" and compared this machine to mesh-connected computers and hypercubes in terms of flux and performance [19]. They introduced the notion of the flux–time product, and they calculated this product for a number of machines and algorithms. Hong and Kung studied the input/output complexity of the fast Fourier transform and other problems [14], and Aggarwal and Vitter studied the input/output complexity of the fast Fourier transform and of permuting and sorting problems [1].

Three types of results are presented in this paper. First, two new lower bounds on the tradeoffs between pin and time requirements for permuting and sorting data are proved. These lower bounds are analogous to the AT^2 (where A is chip area and T is time) lower bounds that have been established for the VLSI area model. These lower bounds apply to all interconnection networks, so they provide an objective measure of the cost/performance tradeoffs of an interconnection topology.

Second, new lower and upper bounds on the pin requirements of various types of parallel computers are proved. Specifically, tight lower bounds on the pin requirements of mesh-connected computers of arbitrary (constant) dimension [21], cube-connected-cycles computers [22], shuffle-exchange computers [26], and Ajtai–Komlós–Szemerédi (AKS) computers [2], [18] are proved, as are tight upper bounds on the pin requirements of shuffle-exchange computers and AKS computers. The lower bound for mesh-connected computers extends a lower bound proved by Snir for two-dimensional mesh-connected computers [25]. The upper bound for shuffle-exchange computers solves an open problem posed by Snir [25], and the lower bound for shuffle-exchange computers generalizes a lower bound proved by Snir [25]. The bounds on AKS computers show that the AKS computer is suboptimal in terms of pin and time requirements for permuting and sorting data. This result is unexpected, because the AKS computer has been proved to be optimal in terms of the VLSI area model [3].

Third, the AKS sorting algorithm is proved to require $\Theta(\log^2 N)$ time to sort N items when it is implemented on an N-processor cube-connected-cycles or shuffle-exchange computer. Previously it was unknown whether the AKS sorting algorithm could be used to obtain an optimal $O(\log N)$-time sorting algorithm for these types of computers. Surprisingly, this bound on the running time of the AKS algorithm makes critical use of the bounds on pin requirements proved earlier in the paper.

The remainder of this paper is organized as follows. Section 2 presents a formal model of the VLSI pin requirements of a parallel machine. Section 3 reviews some previously known results for the pin requirements of parallel computers. Lower bounds on the time and pin requirements for permuting and sorting are presented in §4. Sections 5–8 contain results for mesh-connected computers, cube-connected-cycles computers, shuffle-exchange computers, and AKS computers, respectively. Finally, bounds on the time required to implement the AKS sorting algorithm on cube-connected cycles and shuffle-exchange computers are presented in §9.

2. The VLSI-pin-requirements model. A distributed-memory parallel machine M will be viewed as an undirected graph $M = (V, E)$, where the nodes V represent processors and the edges E represent communication links. Each communication link can transmit one word in unit time. The parameter $N = |V|$ will be the number of processors in M, and the parameter C will be the number of chips to which M is partitioned. It will be assumed throughout that $4 \leq C \leq N$. To minimize the pin requirements for

M, the N processors must be assigned to the C chips so that no chip has a large number of wires connecting the processors on the chip with the processors off the chip. The only restriction in assigning processors to chips will be that each chip has fewer than $N/2$ processors. This restriction is a result of the limited area of a chip, and it prevents the trivial solution of assigning all the processors to a single chip, which requires 0 pins. The above notions are formalized below.

Given any positive integer I, let $[I] = \{0, 1, \cdots, I-1\}$. Let $M = (V, E)$ be an undirected graph. The collection of sets $P = P_0, \cdots, P_{k-1}$ is a *partition* of M if the following properties hold:

1. For each $i \in [k]$, $P_i \subset V$.
2. $\bigcup_{i\in[k]} P_i = V$.
3. For each $i \in [k]$ and $j \in [k]$, where $i \neq j$, $P_i \bigcap P_j = \emptyset$.

The sets $P_0, \cdots, P_{k-1}$ will often be referred to as *chips*. Given a set $S \subseteq V$, the *boundary* of S in M, written Bound(S, M), is the set of edges in E connecting vertices in S with vertices not in S. The *girth* of M partitioned by P, written Girth(M, P), is $\max_{i\in[k]} |\text{Bound}(P_i, M)|$.

A partition $P = P_0, \cdots, P_{k-1}$ of M is *legal* if $\max_{i\in[k]} |P_i| < |V|/2$ and $k \leq C$. The pin requirements of M will be denoted by the parameter Q, where Q is the minimum, over all legal partitions P of M, of Girth(M, P). Given a problem to be solved, the parameter T is the minimum, over all algorithms for the problem, of the algorithm's time requirements on M.

3. Known results. This section briefly reviews some lower and upper bounds on the pin requirements of parallel computers that are either known or that follow directly from known results.

3.1. Mesh-connected computers. The N processors in a d-dimensional mesh-connected computer [21] may be thought of as being arranged in a d-dimensional array. Each side of the array is of length u, where $u = N^{1/d}$ is an integer. The position of each processor in the array is specified by a unique integer vector of the form $(a_{d-1}, \cdots, a_0)$, where $0 \leq a_i < u$ for all i, $0 \leq i < d$. Mesh-connected computers can be defined either with or without wraparound connections. In a mesh-connected computer *without* wraparounds, each processor $(a_{d-1}, \cdots, a_0)$ is connected to the at most $2d$ processors of the form $(a_{d-1}, \cdots, a_{i+1}, a_i \pm 1, a_{i-1}, \cdots, a_0)$, provided that they exist. In a mesh-connected computer *with* wraparounds, each processor $(a_{d-1}, \cdots, a_0)$ is connected to $2d$ processors of the form $(a_{d-1}, \cdots, a_{i+1}, (a_i \pm 1) \bmod u, a_{i-1}, \cdots, a_0)$.

An upper bound on the pin requirements of mesh-connected computers can be obtained by viewing the processors as forming a d-dimensional cube and partitioning the cube into (nearly) cubical blocks with (approximately) N/C processors each. The number of pins required by this partition is based on the surface-to-volume ratio of the cubical blocks and is given in the following theorem due to Snir [25].

THEOREM 3.1. *If M is a d-dimensional mesh-connected computer with or without wraparounds and if d is a constant, then $Q = O((N/C)^{(d-1)/d})$.*

The following theorem, proved by Snir [25], shows that the above bound is the best bound possible when $d = 2$. However, Snir's proof technique makes critical use of a property that is unique to the case $d = 2$ (namely, that a single chip cannot receive processors from opposite faces of the cube) and does not generalize to higher values of d.

THEOREM 3.2. *If M is a 2-dimensional mesh-connected computer with or without wraparounds, then $Q = \Omega((N/C)^{1/2})$.*

3.2. Hypercube computers. A hypercube computer [21], [24] consists of $N = 2^n$ processors, each of which has a unique label in the range 0 through $N - 1$. Each pair of processors i and j are connected if and only if the binary representations of i and j differ in exactly one bit position.

An upper bound on the pin requirements of a hypercube can be obtained by partitioning the hypercube into smaller dimensional hypercubes with (approximately) N/C processors each. Each processor will then be connected to $O(\log N - \log(N/C)) = O(\log C)$ processors[1] on different chips, so $Q = O(N \log C/C)$. This result, which is very natural and has been discovered by several researchers [13], [19], [24], is stated in the following theorem.

THEOREM 3.3. *If M is a hypercube computer with N processors, then it holds that $Q = O(N \log C/C)$.*

This partition into smaller dimensional hypercubes can be shown to be optimal by using the following lemma (called an *isoperimetric inequality*) proved by Hart [12]. For any nonnegative integer i, let $h(i)$ denote the number of ones in the binary representation of i. For example, $h(0) = 0$ and $h(23) = 4$. Also, for any positive integer s, let $g(s) = \sum_{i=0}^{s-1} h(i)$.

LEMMA 3.4. *Let $M = (V, E)$ be a hypercube computer with N processors, and let $S \subseteq V$, where $|S| = s > 0$. Then $|\mathrm{Bound}(S, M)| \geq s \log N - 2g(s)$.*

In order to use Lemma 3.4 to prove that the upper bound given by Theorem 3.3 is optimal, we will need the following bound on the value of $g(s)$.

LEMMA 3.5. *For all integers $s \geq 1$, $g(s) \leq s \lceil \log s \rceil /2$.*

Proof. All integers in the range 0 through $s - 1$ can be represented with $\lceil \log s \rceil$ bits. In the sequence $0, 1, \cdots, s - 1$, the jth bit, $0 \leq j < \lceil \log s \rceil$, appears periodically as 2^j zeros followed by 2^j ones. Because the jth bit first appears as 2^j zeros, it has the value one for at most half of the integers in the sequence $0, 1, \cdots, s - 1$. Thus $g(s) = \sum_{i=0}^{s-1} h(i) \leq s \lceil \log s \rceil /2$. $\square$

Combining Lemmas 3.4 and 3.5 yields the following theorem.

THEOREM 3.6. *If M is a hypercube computer with N processors, then it holds that $Q = \Omega(N \log C/C)$.*

Proof. Let $P = P_0, \cdots, P_{k-1}$ be any legal partition of M, let $s = \max_{i \in [k]} |P_i|$, and let j be such that $|P_j| = s$. Note that $k \leq C$, so the average size of the sets P_i is at least N/C and $s \geq N/C$. Also, because P is a legal partition of M, $s < N/2$. Let $f(x) = x(\log N - \lceil \log x \rceil)$. From Lemmas 3.4 and 3.5 $|\mathrm{Bound}(P_j, M)| \geq f(s)$. There are two cases. If $N/4 \leq s < N/2$, then $\log N - \lceil \log s \rceil \geq 1$ and $f(s) \geq N/4 = \Omega(N \log C/C)$. Otherwise, $N/C \leq s < N/4$ and $\log N - \log s \geq 2$, so $\log N - \lceil \log s \rceil \geq \log N - \log s - 1 \geq (\log N - \log s)/2$. Let $e(x) = x(\log N - \log x)/2$. Then the derivative $e'(x) = (\ln N - \ln x - 1)/2 \ln 2$, so when $\ln x < \ln N - 1$, $e'(x) > 0$. Thus $f(s) = s(\log N - \lceil \log s \rceil) \geq e(s)$ and $e(s)$ is minimized when $s = N/C$, at which point $e(s) = (N/2C)(\log C) = \Omega(N \log C/C)$. Thus in either case $Q \geq |\mathrm{Bound}(P_j, M)| \geq f(s) = \Omega(N \log C/C)$. $\square$

3.3. Cube-connected-cycles computers. A cube-connected-cycles (CCC) computer [22] consists of $N = n2^n$ processors, where $n = 2^m$ for some integer m. Each processor has a unique label of the form (r, c), where $0 \leq r < 2^n$ and $0 \leq c < n$. Let $a \oplus b$ denote the bitwise exclusive OR of a and b. Each processor (r, c) is connected to processor $(r \oplus 2^c, c)$ through a *cube edge* and to processors $(r, (c \pm 1) \bmod n)$ through *lateral edges*.

[1]Throughout this paper $\log N$ will denote the base 2 logarithm of N and $\ln N$ will denote the natural logarithm of N.

Snir [25] proved tight lower and upper bounds on the pin requirements of Omega networks, which are multistage switch-based interconnection networks. Snir's upper bound is based on a partition of an Omega network into a collection of smaller Omega networks. By associating each switch in an Omega network with a pair of processors that are connected by a cube edge in a CCC, Snir's upper bound on the pin requirements of an Omega network can be used to obtain an upper bound on the pin requirements of a CCC. In fact, the resulting partition of a CCC is similar to partitions given by a number of authors, including Hong and Kung [14]; Knauer, O'Neill, and Huang [15]; and Aggarwal and Vitter [1]. The pin requirements of this partition of a CCC are given in the following theorem.

THEOREM 3.7. *If M is a cube-connected-cycles computer with N processors, then $Q = O(N/C \log(4N/C))$.*

It should be noted that Snir's lower bound on the pin requirements of an Omega network does not imply a similar lower bound on the pin requirements of a CCC, because each switch in an Omega network must be assigned to a single chip, whereas a pair of processors in a CCC that are connected by a cube edge can be assigned to different chips.

One interesting consequence of Theorems 3.6 and 3.7 is that if $C = N^x$, where $0 < x < 1$ is a constant, then the pin requirements of a hypercube are a factor of $\Omega(\log^2 N)$ greater than those of a CCC even though the degree of a hypercube is only a factor of $\Theta(\log N)$ greater than the degree of a CCC.

4. Lower bounds for permuting and sorting.

Having defined the VLSI-pin-requirements model, we are able to establish bounds on relationships among the parameters N, C, Q and T. In this paper we will focus on the permutation and sorting problems. For both problems it is assumed that N items are to be permuted or sorted and that each item consists of a constant number of words. The only operations that are allowed to be performed on items are duplication of the item and examination of the item's destination or key value, so algorithms that encode the contents of an item are prohibited. Also, it is assumed that each processor originally holds one item and that after the permutation or sort each processor again holds one item. When an average-case time analysis is used, it is assumed that all of the $N!$ different permutations or orderings of the sort keys are equally likely. In the permutation problem only the time spent permuting the data is counted; the time spent determining how to accomplish the permutation is ignored. The following theorems establish lower bounds on the time and pin requirements of permuting and sorting. Theorem 4.1 is closely related to a lower bound on the input/output complexity of permuting and sorting that was obtained by Aggarwal and Vitter [1] and to a lower bound on area that was proven by Vuillemin [32].

THEOREM 4.1. *If M is a parallel machine with N processors and if T is the average- or worst-case time required to permute or sort N items on M, then $TCQ = \Omega(N)$.*

Proof. The lower bound for the average-case time analysis will be proved here. The lower bound for the worst case follows immediately from the lower bound for the average case. View M as a graph $M = (V, E)$, and let $P = P_0, \cdots, P_{C-1}$ be any legal partition of M. Let $j \in [C]$ be such that for all $i \in [C]$, $|P_j| \geq |P_i|$, and let $S = P_j$. Note that the average size of the sets P_i is at least N/C so $|S| \geq N/C$. Also, because P is a legal partition of M, $|S| < N/2$.

Note that for any permutation or sort, if k items must be sent from processors in S to processors outside of S, then the time required for that permutation or sort is at least $k/|\text{Bound}(S, M)|$. Also, to implement all $N!$ of the different permutations or sorts, a total of $|S|(N - |S|)(N - 1)!$ items must be sent from processors in S to processors outside of S. Because $N/C \leq |S| < N/2$, $|S|(N-|S|)(N-1)! > (N/2C)N!$. Therefore, the total time required to implement all $N!$ different permutations or sorts is at least $N!(N/2C|\text{Bound}(S,M)|)$, and the average time required for a permutation or sort is at least $N/2C|\text{Bound}(S,M)|$. Because $Q \geq |\text{Bound}(S,M)|$, it follows that $T \geq N/2CQ$ and $TCQ = \Omega(N)$. □

Although Theorem 4.1 gives an important lower bound on the time and pin requirements for permuting and sorting, a tighter lower bound can be obtained in some cases. This other lower bound is proved in the following theorem.

THEOREM 4.2. *If M is a parallel machine with N processors and if T is the average- or worst-case time required to permute or sort N items on M, then* $TCQ \log Q/\log C = \Omega(N)$.

Proof. The lower bound for the average-case time analysis will be proved here. The lower bound for the worst case follows immediately from the lower bound for the average case. If $4 \leq C < 256$, then from Theorem 4.1 $TCQ = \Omega(N)$, which implies that $TCQ \log Q/\log C = \Omega(N)$. Therefore, it will be assumed that $C \geq 256$.

The idea of the proof is that at most TCQ items can move between chips in time T. Thus if it could be shown that on the average $\Omega(N)$ items each have to visit $\Omega(\log C/\log Q)$ chips, then TCQ would have to be $\Omega(N \log C/\log Q)$, which implies that $TCQ \log Q/\log C = \Omega(N)$. This argument can be used if the processors are fairly evenly distributed among the chips, but it fails if many of the processors are assigned to a small number of chips. However, in that case an argument like the one used in Theorem 4.1 can be used to obtain the desired result. Thus a two-case analysis will be used on the basis of how evenly the processors are assigned to the chips.

View M as a graph $M = (V,E)$, and let $P = P_0, \cdots, P_{C-1}$ be any legal partition of M. Without loss of generality, assume that for all $i,j \in [C]$ if $i < j$, then $|P_i| \geq |P_j|$. Let $s = |P_0|$, and for each i, where $1 \leq i \leq s$, let y_i be the number of chips having at least i processors (see Fig. 1). More formally, $y_i = C$ if $|P_{C-1}| \geq i$, and $y_i = \min\{j \in [C] \mid |P_j| < i\}$ otherwise. Note that

$$\sum_{i=1}^{s} y_i = N.$$

(This follows from the fact that a chip with j processors will be counted in exactly j of the y_i's.) Let k be the smallest integer such that

$$\sum_{i=1}^{k} y_i \geq N/2.$$

The definition of k is illustrated in Fig. 2, where the area of the shaded region consisting of the first k rows is at least $N/2$. Two cases will be considered on the basis of the value of y_k.

First, consider the case for which $y_k \geq C^{1/2}$. Let Wires(P,M) be the set of all edges in E that connect nodes that are in different sets in P, and for any $i,j \in V$ let WireDist(i,j,P,M) be the minimum, over all paths between i and j in M, of the number of edges in the path that are in the set Wires(P,M). Then, for an item to go from processor i to processor j it must cross at least WireDist(i,j,P,M) wires between chips. Note that given any processor i and any distance d, at most $Q^d + 1$ chips contain a processor j such that WireDist$(i,j,P,M) \leq d$. In particular, for any processor i at most $\lfloor C^{1/4} + 1 \rfloor$ chips contain a processor j such that WireDist$(i,j,P,M) \leq \log C/4 \log Q$.

Let R be a set consisting of k of the processors from each chip, or all of the processors from a chip if the chip has fewer than k processors. For example, the shaded region in

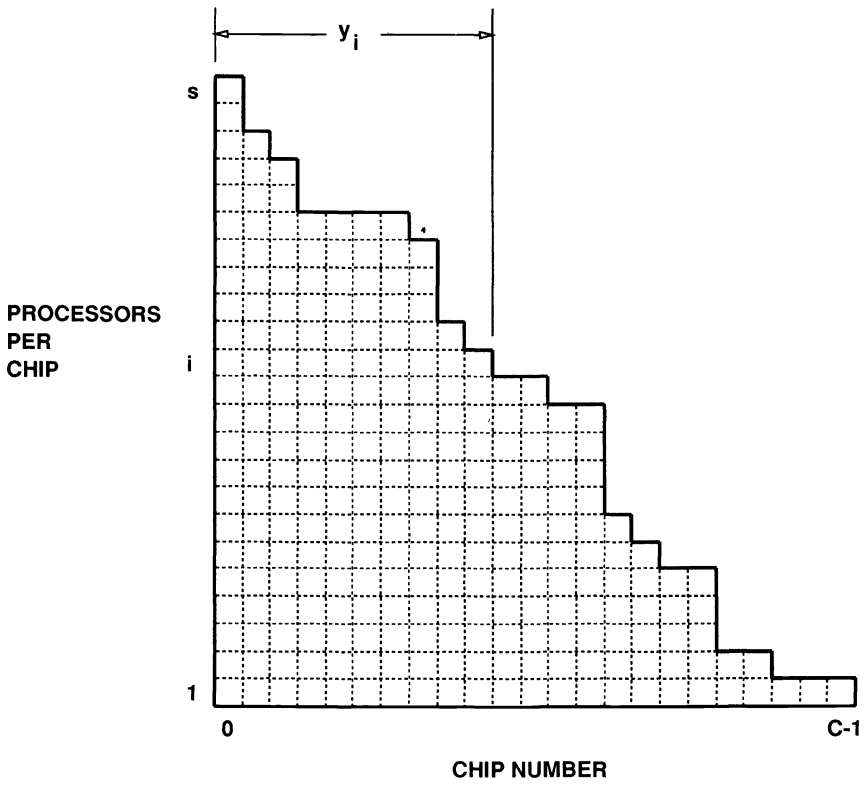


FIG. 1. *Number of processors per chip as a function of chip number. Each square represents one processor.*

Fig. 2 is such a set R. Note that

$$N \geq |R| = \sum_{i=1}^{k} y_i \geq N/2.$$

It is clear that $ky_k \leq |R|$, so $k \leq |R|/y_k \leq N/C^{1/2}$. As a result, for any positive integer z any set of z chips can contain at most $zk \leq zN/C^{1/2}$ processors in R. If we set $z = \lfloor C^{1/4} + 1 \rfloor$ and recall that $C \geq 256$, it follows that for any processor i there are at most $\lfloor C^{1/4} + 1 \rfloor N/C^{1/2} \leq (C^{-1/4} + C^{-1/2})N \leq 3N/8$ processors $j \in R$ such that WireDist$(i, j, P, M) \leq \log C/4 \log Q$. Therefore, for any given processor i there are at least $|R| - 3N/8 \geq N/8$ processors $j \in R$ such that WireDist$(i, j, P, M) > \log C/4 \log Q$.

As a result, the sum over all $N!$ permutations or sorts of the total number of times that items must move between chips to implement the permutation or sort is at least $(\log C/4 \log Q)(N/8)(N)(N-1)! = (N \log C/32 \log Q)N!$. However, only CQ items can move between chips in unit time, so the total time required to implement all $N!$ permutations or sorts is at least $N!(N \log C/32CQ \log Q)$ and the average time required to implement a permutation or sort is at least $N \log C/32CQ \log Q$. Therefore, $T \geq N \log C/32CQ \log Q$, and $TCQ \log Q/ \log C = \Omega(N)$.

Next, consider the case for which $y_k < C^{1/2}$. Notice that

$$\sum_{i=1}^{k-1} y_i < N/2,$$

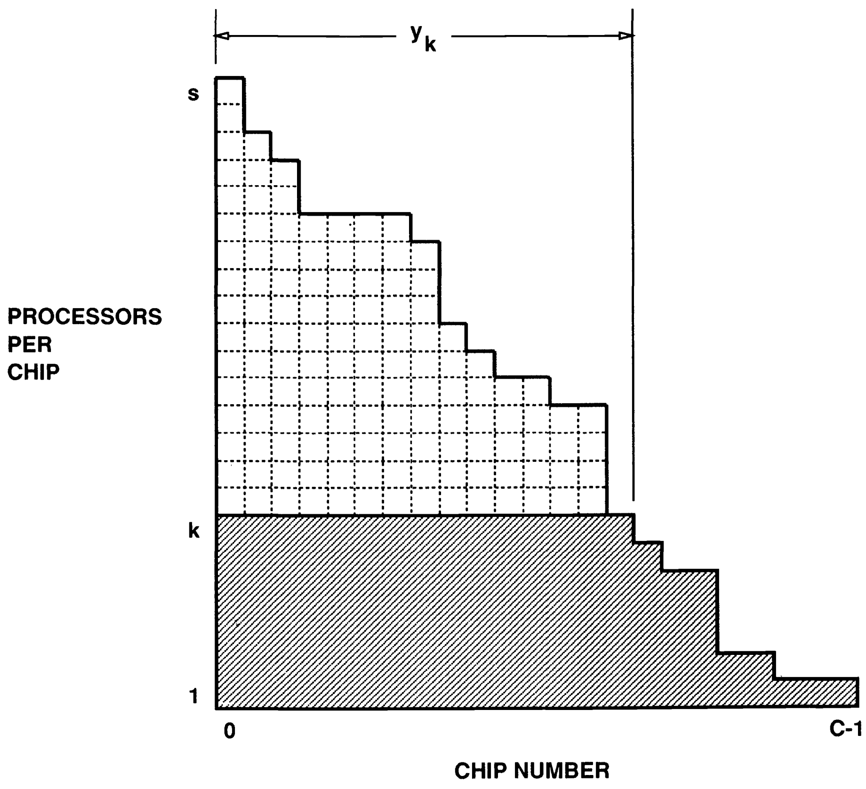


FIG. 2. *Number of processors per chip as a function of chip number. The value k is the smallest integer such that the lowest k rows contain at least half of the processors. The shaded region shows the $N/2$ or more processors considered when $Y_k \geq C^{1/2}$.*

so

$$\sum_{i=k}^{s} y_i \geq N/2,$$

as is illustrated in Fig. 3. However, for each i where $k \leq i \leq s$, $y_i < C^{1/2}$. Therefore,

$$N/2 \leq \sum_{i=k}^{s} y_i < \sum_{i=k}^{s} C^{1/2} = (s-k+1)C^{1/2},$$

so $s-k+1 > N/2C^{1/2}$. Because $k \geq 1$, $s > N/2C^{1/2}$.

Now let $S = P_0$. To implement all $N!$ of the different permutations or sorts, a total of $|S|(N-|S|)(N-1)! = s(N-s)(N-1)!$ items must be sent from processors in S to processors outside of S. Because $N/2C^{1/2} < s < N/2$, $s(N-s)(N-1)! > (N/4C^{1/2})N!$. Therefore, the total time required to implement all $N!$ different permutations or sorts is at least $N!(N/4C^{1/2})/|\text{Bound}(S,M)|$, and the average time required for a permutation or sort is at least $(N/4C^{1/2})/|\text{Bound}(S,M)|$. Because $Q \geq |\text{Bound}(S,M)|$, $T \geq N/4C^{1/2}Q$ and $TCQ/\log C \geq NC^{1/2}/4\log C \geq N/4$, so $TCQ\log Q/\log C = \Omega(N)$. □

Notice that the stronger lower bound is obtained by using Theorem 4.1 when $C = O(Q)$ and by using Theorem 4.2 when $Q = O(C)$. A natural question is whether or not

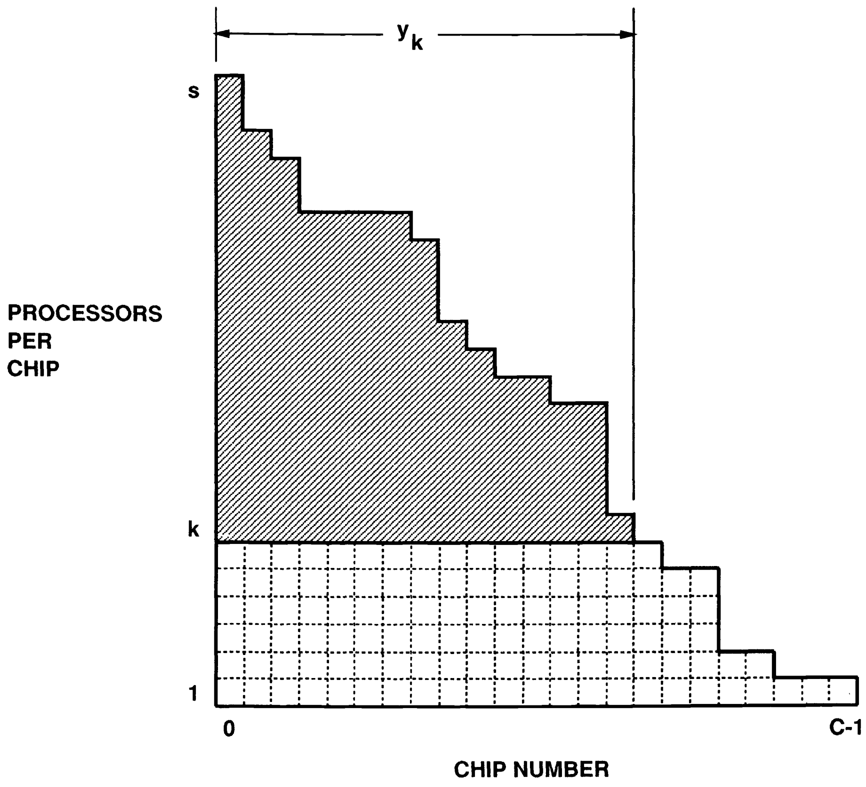


FIG. 3. *Number of processors per chip as a function of chip number. The value k is the smallest integer such that the lowest k rows contain at least half of the processors. The shaded region shows the $N/2$ or more processors considered when $Y_k < C^{1/2}$.*

these lower bounds are the strongest possible. In §6 it will be shown that for the average- and worst-case permutation problems and for the average-case sorting problem, when $C = O(Q)$ the lower bound given by Theorem 4.1 can be attained and when $Q = O(C)$ the lower bound given by Theorem 4.2 can be attained. It is unknown whether a stronger lower bound exists for the worst-case sorting problem.

5. *d*-Dimensional mesh-connected computers. The definition of a d-dimensional mesh-connected computer and an upper bound on its pin requirements were given in §3. A tight lower bound on the pin requirements of a two-dimensional mesh-connected computer was given in §3. In this section a tight lower bound on the pin requirements of a d-dimensional mesh-connected computer will be proved for any positive constant d. The lower bound is based on the following definitions and lemmas.

For the remainder of this section let $M = (V, E)$ be a d-dimensional mesh-connected computer (with or without wraparounds), where d is a constant, and let $u = N^{1/d}$. Given a set $S \subseteq V$ and an integer $k \in [d]$, Proj(S, k) is obtained by setting the kth coordinate of each vector in S to 0 and removing duplicates. More formally, Proj$(S, k) = R$, where $y = (y_{d-1}, \cdots, y_0) \in R$ if and only if $y_k = 0$ and $\exists x = (x_{d-1}, \cdots, x_0) \in S$ such that for all $i \in [d]$, where $i \neq k$, $x_i = y_i$. Given a set $S \subseteq V$ and integers i and k where $k \in [d]$, Slice(S, k, i) consists of those vectors in S for which the kth coordinate equals i. More formally, Slice$(S, k, i) = R$, where $y = (y_{d-1}, \cdots, y_0) \in R$ if and only if $y \in S$ and $y_k = i$. Given a set $S \subseteq V$ and an integer $k \in [d]$, Intersect(S, k, u) consists of those vectors $y \in \text{Proj}(S, k)$ for which all u of the vectors obtainable from y by setting the kth coordinate to a value in $[u]$ are also in S. More formally, $\text{Intersect}(S, k, u) = R$, where $y = (y_{d-1}, \cdots, y_0) \in R$ if and only if $y_k = 0$ and for all $i \in [u]$, $\exists x = (x_{d-1}, \cdots, x_0) \in S$ such that $x_k = i$ and for all $j \in [d]$, where $j \neq k$, $x_j = y_j$. Finally, for any $(x, y) \in E$, $\text{Dim}((x, y)) = k$ if and only if $x = (x_{d-1}, \cdots, x_0)$ and $y = (y_{d-1}, \cdots, y_0)$ and $x_k \neq y_k$.

LEMMA 5.1. *Given a nonempty set $S \subseteq V$,*

$$\sum_{j \in [d]} |\text{Proj}(S, j)| \geq d s^{(d-1)/d},$$

where $s = |S|$.

Proof. The proof is by induction on d. The basis is $d = 1$, in which case the claim is that $|\text{Proj}(S, 0)| \geq 1$, which follows immediately from the fact that S is nonempty. The induction hypothesis is that the lemma holds for all $d < a$, and it will be shown that the induction hypothesis implies that the lemma holds for $d = a$. Assume that $d = a$. For all $i \in [u]$ let $S_i = \text{Slice}(S, a-1, i)$ and let $s_i = |S_i|$. Note that $\sum_{i \in [u]} s_i = s$. Also, note that for each $j \in [a-1]$

$$|\text{Proj}(S, j)| = \sum_{i \in [u]} |\text{Proj}(S_i, j)|.$$

Therefore,

$$\sum_{j \in [a-1]} |\text{Proj}(S, j)| = \sum_{j \in [a-1]} \sum_{i \in [u]} |\text{Proj}(S_i, j)| = \sum_{i \in [u]} \sum_{j \in [a-1]} |\text{Proj}(S_i, j)|,$$

and, because each of the sets S_i is contained in an $(a-1)$-dimensional cube, the induction hypothesis yields

$$\sum_{i \in [u]} \sum_{j \in [a-1]} |\text{Proj}(S_i, j)| \geq \sum_{i \in [u]} (a-1) s_i^{(a-2)/(a-1)} = (a-1) \sum_{i \in [u]} s_i s_i^{-1/(a-1)}.$$

Let $h \in [u]$ be such that for each $i \in [u]$, $s_h \geq s_i$. Then

$$(a-1) \sum_{i \in [u]} s_i s_i^{-1/(a-1)} \geq (a-1) s_h^{-1/(a-1)} \sum_{i \in [u]} s_i = (a-1) s_h^{-1/(a-1)} s.$$

It has been shown that

$$\sum_{j \in [a-1]} |\text{Proj}(S, j)| \geq (a-1) s_h^{-1/(a-1)} s.$$

However, $|\text{Proj}(S, a-1)| \geq s_h$, so

$$\sum_{j \in [a]} |\text{Proj}(S, j)| \geq s_h + (a-1) s_h^{-1/(a-1)} s.$$

Let $f(x) = x + (a-1) x^{-1/(a-1)} s$. The derivative $f'(x) = 1 - x^{-a/(a-1)} s$ equals 0 when $x = s^{(a-1)/a}$, and the second derivative $f''(x) = (a/(a-1)) x^{(1-2a)/(a-1)} s$ is positive when $x = s^{(a-1)/a}$, so $f(x)$ is minimized when $x = s^{(a-1)/a}$. Therefore,

$$\sum_{j \in [a]} |\text{Proj}(S, j)| \geq s^{(a-1)/a} + (a-1)(s^{(a-1)/a})^{-1/(a-1)} s = a s^{(a-1)/a},$$

which completes the inductive step. □

LEMMA 5.2. *For any nonempty set $S \subseteq V$ and any integer $k \in [d]$, let $E_{k,S} = \{(x, y) \in \text{Bound}(S, M) \mid \text{Dim}((x, y)) = k\}$. Then for any $k \in [d]$, $|E_{k,S}| \geq |\text{Proj}(S, k)| - |\text{Intersect}(S, k, u)|$.*

Proof. Let $R = \text{Proj}(S, k) \setminus \text{Intersect}(S, k, u)$. Note that $|R| = |\text{Proj}(S, k)| - |\text{Intersect}(S, k, u)|$. For any $x \in R$ let $V_x = \{y \in V \mid \text{Proj}(\{y\}, k) = \{x\}\}$. Then there must exist a $y = (y_{d-1}, \cdots, y_0) \in V_x$ and a $y' = (y_{d-1}, \cdots, y_{k+1}, y_k + 1, y_{k-1}, \cdots, y_0) \in V_x$ where either $y \in S$ and $y' \notin S$ or $y \notin S$ and $y' \in S$. Therefore, for each $x \in R$ there exists an edge $(y, y') \in E_{k,S}$, where $y \in V_x$ and $y' \in V_x$. Note that for any $x \in R$ and $x' \in R$, if $x \neq x'$, then V_x and $V_{x'}$ are disjoint. Therefore, the edges in $E_{k,S}$ corresponding to distinct elements of R are distinct. As a result, $|E_{k,S}| \geq |R| = |\text{Proj}(S, k)| - |\text{Intersect}(S, k, u)|$. □

THEOREM 5.3. *If M is a d-dimensional MCC with or without wraparounds and if d is a constant, then $Q = \Omega((N/C)^{(d-1)/d})$.*

Proof. Let $P = P_0, \cdots, P_{C-1}$ be any legal partition of M. Let $j \in [C]$ be such that for all $i \in [C]$, $|P_j| \geq |P_i|$, and let $S = P_j$. Note that $N/C \leq |S| < N/2$. From Lemma 5.2

$$\begin{aligned}|\text{Bound}(S, M)| &\geq \sum_{k\in[d]} (|\text{Proj}(S, k)| - |\text{Intersect}(S, k, u)|) \\ &= \sum_{k\in[d]} |\text{Proj}(S, k)| - \sum_{k\in[d]} |\text{Intersect}(S, k, u)|.\end{aligned}$$

From Lemma 5.1

$$\sum_{k\in[d]} |\text{Proj}(S, k)| \geq d|S|^{(d-1)/d}.$$

Note that for any $k \in [d]$, $|\text{Intersect}(S, k, u)| \leq |S|/u$, so

$$\sum_{k\in[d]} |\text{Intersect}(S, k, u)| \leq d|S|/u.$$

Therefore,

$$\begin{aligned}|\text{Bound}(S, M)| &\geq d|S|^{(d-1)/d} - d|S|/u \\ &= d|S|^{(d-1)/d}(1 - |S|^{1/d}/N^{1/d}) \\ &> d(N/C)^{(d-1)/d}(1 - 1/2^{1/d}) \\ &= \Omega((N/C)^{(d-1)/d}).\end{aligned}$$

Because $Q \geq |\text{Bound}(S, M)|$, it follows that $Q = \Omega((N/C)^{(d-1)/d})$. □

It should be noted that Bollobás and Leader have proved isoperimetric inequalities (lower bounds on the number of neighbors that a set of nodes must have) for d-dimensional meshes both with and without wraparounds [4], [5]. However, those isoperimetric inequalities characterize a set that has a minimal number of neighbors without quantifying the number of neighbors, and as a result they do not provide a bound such as the one proved in Theorem 5.3.

6. Cube-connected-cycles computers. The definition of a CCC computer and an upper bound on its pin requirements were given in §3. By combining the lower bounds of Theorems 4.1 and 4.2 with published algorithms for permuting and sorting on a CCC computer, it is possible to prove tight lower bounds on the pin requirements of a CCC computer. The following lemma is due to Preparata and Vuillemin [22].

LEMMA 6.1. *The average- and worst-case times required to permute N items on an N-processor cube-connected-cycles computer are $O(\log N)$.*

The following lemma was proven by Reif and Valiant [23].

LEMMA 6.2. *The average-case time required to sort N items on an N-processor cube-connected-cycles computer is $O(\log N)$.*

THEOREM 6.3. *If M is a cube-connected-cycles computer with N processors, then $Q = \Omega(N/C \log(4N/C))$.*

Proof. Let T be the worst-case time required to permute N items on M. From Lemma 6.1 $T = O(\log N)$. We will consider two cases based on the relative values of C and N. First, consider the case $C \leq N^{1/2}$. In this case note that $\log(4N/C) = \Omega(\log N)$. From Theorem 4.1 $TCQ = \Omega(N)$, so $Q = \Omega(N/C \log N) = \Omega(N/C \log(4N/C))$.

Next, consider the case $C > N^{1/2}$. In this case note that $\log C = \Omega(\log N)$. From Theorem 4.2 $TCQ \log Q / \log C = \Omega(N)$, so $Q \log Q = \Omega(N \log C / C \log N) = \Omega(N/C)$. If $Q < 4N/C$ then $\log Q = O(\log(4N/C))$ and $Q = \Omega(N/C \log(4N/C))$. Otherwise, $Q \geq 4N/C$, so $Q = \Omega(N/C)$ and $Q = \Omega(N/C \log(4N/C))$. □

We will now show that the CCC computer is capable of matching the lower bounds for the average- and worst-case permutation problems and for the average-case sorting problem given by Theorems 4.1 and 4.2. Recall that the stronger lower bound is obtained by using Theorem 4.1 when $C = O(Q)$ and by using Theorem 4.2 when $Q = O(C)$.

THEOREM 6.4. *If M is a cube-connected-cycles computer with N processors and if T is the average-case time required to permute or sort N items on M or the worst-case time required to permute N items on M, then either*

$$C = O((N/\log N)^{1/2})$$

and

$$C = O(Q) \quad \textit{and} \quad TCQ = O(N)$$

or

$$C = \Omega((N/\log N)^{1/2})$$

and

$$Q = O(C) \quad \textit{and} \quad TCQ \log Q / \log C = O(N).$$

Proof. First, consider the case $C = O((N/\log N)^{1/2})$. From Theorem 6.3

$$\begin{aligned} Q &= \Omega(N/C \log(4N/C)) \\ &= \Omega(N/(N/\log N)^{1/2} \log N) \\ &= \Omega((N/\log N)^{1/2}) \\ &= \Omega(C), \end{aligned}$$

so $C = O(Q)$. From Lemmas 6.1 and 6.2 $T = O(\log N)$. From Theorem 3.7 $Q = O(N/C \log(4N/C))$. Therefore, $TCQ = O(N \log N / \log(4N/C))$. Because $C = O((N/\log N)^{1/2})$, $\log(4N/C) = \Omega(\log N)$. Therefore, $TCQ = O(N)$.

Next, consider the case $C = \Omega((N/\log N)^{1/2})$. If $f(x) = x/\log x$, then the derivative $f'(x) = (\ln^{-1} x - \ln^{-2} x)\ln 2$ and $f'(x) > 0$ when $x \geq 3$. But $4N/C \geq 4$, so $f(4N/C)$ is maximized when $4N/C$ is largest. From Theorem 3.7

$$\begin{aligned} Q &= O(N/C\log(4N/C)) \\ &= O(f(4N/C)) \\ &= O((N\log N)^{1/2}/\log(N\log N)^{1/2}) \\ &= O((N/\log N)^{1/2}) \\ &= O(C). \end{aligned}$$

Because $f(4N/C) \geq 2$ and $Q = O(f(4N/C))$, it holds that $\log Q = O(\log f(4N/C)) = O(\log(4N/C))$. Therefore, $Q\log Q = O(N/C)$. From Lemmas 6.1 and 6.2 $T = O(\log N)$. Because $C = \Omega((N/\log N)^{1/2})$, $\log C = \Omega(\log N)$. As a result, $TCQ\log Q/\log C = O(N)$. □

7. Shuffle-exchange computers. This section contains tight lower and upper bounds on the pin requirements of shuffle-exchange computers [26]. A shuffle-exchange computer has $N = 2^n$ processors, each of which has a unique label in the range 0 through $N - 1$. For any integer i, where $0 \leq i < N$, let $(i_{n-1}, \cdots, i_0)$ denote the n-bit binary representation of i. Let $a \oplus b$ denote the bitwise exclusive OR of a and b, and for any integers i and j, where $0 \leq i < N$, $0 \leq j < n$, let $\text{Comp}(i,j) = i \oplus 2^j$. For any integer i, where $0 \leq i < N$, let $\text{Shuffle}(i,n) = (i_{n-2}, \cdots, i_0, i_{n-1})$, let $\text{Unshuffle}(i,n) = (i_0, i_{n-1}, \cdots, i_1)$, and let $\text{Exch}(i) = \text{Comp}(i,0)$. In a shuffle-exchange computer each processor i is connected to processor $\text{Exch}(i)$ by an *exchange edge* and to processors $\text{Shuffle}(i,n)$ and $\text{Unshuffle}(i,n)$ by *shuffle edges*.

The lower bound on the pin requirements of a shuffle-exchange computer is based on the following lemma due to Stone [26].

LEMMA 7.1. *The worst-case time required to permute N items on an N-processor shuffle-exchange computer is* $O(\log N)$.

Combining Lemma 7.1 with Theorems 4.1 and 4.2 yields the following theorem, which generalizes a result obtained by Snir [25]. The proof is identical to the proof of Theorem 6.3 and will not be repeated here.

THEOREM 7.2. *If M is a shuffle-exchange computer with N processors, then* $Q = \Omega(N/C\log(4N/C))$.

The following theorem solves an open problem posed by Snir [25]. The proof is based on results obtained by Leighton in creating an optimal area layout of the shuffle exchange [17].

THEOREM 7.3. *If M is a shuffle-exchange computer with N processors, then* $Q = O(N/C\log(4N/C))$.

Proof. First, note that if N/C is less than some constant k, then $\log(4N/C) = O(1)$ and the theorem holds from any assignment of nearly equal numbers of processors to each chip. Therefore, in the remainder of the proof it will be assumed that N/C is larger than any given constant k. To obtain an optimal area layout of the shuffle exchange, Leighton divided the processors into two sets, one that contained $O(N/\log N)$ processors and will be called the *exceptional processors* and another that contained the remainder of the processors and will be called the *regular processors* [17]. (In Leighton's terminology, the exceptional processors include processors in degenerate necklaces, processors for which the longest block of zeros has length less than $\log n - \log\ln n - 1$ or greater than $2 \log n$, and processors for which there is no distinguished node for the piece of the necklace containing the processor.) Leighton then defined a $\log N \times O(N/\log N)$ grid of nodes, and he gave rules for placing the regular processors on this grid.

A similar grid of nodes will be used here. For any regular processor i let Col(i) be what Leighton referred to as the distinguished node of the (basic, primary, or secondary) piece of the necklace containing processor i. Also, for any regular processor i let Row(i) be the row of Leighton's grid that contains processor i. Then create a $\log N \times O(N/\log N)$ grid of nodes, where the rows are numbered $0, \cdots, \log N - 1$ and the columns are numbered with all of the possible values of the function Col(i). Next, place each regular processor i in row Row(i) and column Col(i) of this grid. The following properties of this grid follow immediately from the properties of Leighton's grid.

• At most three processors are assigned to each node of the grid.

• For all but $O(N/\log N)$ regular nodes i, Col(Shuffle(i, n)) $=$ Col(i) and Row(Shuffle(i, n)) $=$ Row(i) $- 1$.

• For all but $O(N/\log N)$ regular nodes i, Col(Unshuffle(i, n)) $=$ Col(i) and Row(Unshuffle(i, n)) $=$ Row(i) $+ 1$.

• For all but $O(N/\log N)$ regular nodes i, Col(Exch(i)) $=$ Comp(Col(i), Row(i)) and Row(Exch(i)) $=$ Row(i).

This grid will be used to partition the processors onto C chips. The idea is to create an initial partition of the processors P' such that each group in the initial partition is very small and the total number of edges that cross partition boundaries is also small. A final partition of the processors P is obtained by merging groups in the initial partition until no more than C groups remain.

The initial partition P' is based on the observation that when the regular processors of a shuffle-exchange computer are assigned to the $\log N \times O(N/\log N)$ grid defined above, a similarity between their connections and the connections of a CCC computer becomes apparent. As a result, the optimal partition of a CCC computer given in Theorem 3.7 can be used to obtain a good partition of the regular nodes in a shuffle-exchange computer. Let $x = \lfloor (\log(4N/C))/2 \rfloor$, and let $y = 2^x$. Note that $x = \Theta(\log(4N/C))$ and $y = \Theta((N/C)^{1/2})$. For each regular processor i let Band(i) $= \lfloor \text{Row}(i)/x \rfloor$, and if Col($i$) $= j = (j_{n-1}, \cdots, j_0)$ then let Vert(i) $= k = (k_{n-1}, \cdots, k_0)$, where for all h, $0 \leq h < n$, $k_h = 0$ if $\lfloor h/x \rfloor =$ Band(i) and $k_h = j_h$ otherwise. The partition P' is formed by assigning each exceptional processor to its own group and by assigning regular processors i and j to the same group if and only if Band(i) $=$ Band(j) and Vert(i) $=$ Vert(j). Then each group of this partition has at most $3xy = O((N/C)^{1/2} \log(4N/C)) = O(N/C \log(4N/C))$ processors and the size of the boundary of each group is at most $O(N/C \log(4N/C))$. Also, there are at most $O((N/\log N)(\log N/\log(4N/C))) = O(N/\log(4N/C))$ shuffle edges that are not local to a group of processors and at most $O(N/\log N) = O(N/\log(4N/C))$ exchange edges that are not local to a group of processors. Let the constant a be such that there are at most $aN/\log(4N/C)$ edges that are not local to a group of processors.

Next, the groups defined by partition P' are arbitrarily divided into three classes with the only restriction being that each class contains fewer than $N/2$ processors. The final partition P is then created by merging groups in the partition determined by P', subject to the restriction that only groups in the same class can be merged. Specifically, a greedy procedure is used to merge groups defined by partition P', all of which are in the same class, until a cluster (collection of groups) is obtained that has a boundary of $4aN/C \log(4N/C)$ or more edges, or until the class is empty. Because the groups in the initial partition are so small, each cluster will have a boundary of at most $O(N/$

$C \log(4N/C))$ edges. Also, because there are at most $aN/\log(4N/C)$ edges that are not local to a group in the initial partition and because all but three of the clusters have a boundary of at least $4aN/C \log(4N/C)$ edges, there are at most $C/4 + 3 \leq C$ clusters. Finally, each cluster has fewer than $N/2$ processors. Thus the partition P, which is defined to consist of these clusters, is legal, and $Q \leq \text{Girth}(M, P) = O(N/C \log(4N/C))$. □

Following the original appearance of Theorem 7.3 [8], Koch et al. showed an embedding of an N-processor shuffle exchange into an $O(N)$-processor butterfly with constant load and congestion [16]. That embedding, combined with a similar embedding of a butterfly into a cube-connected-cycles computer and Theorem 3.7, could be used to obtain an alternative proof of Theorem 7.3.

The following lemma proves that when $C = N^{1/2}$ the upper bound on Q given in Theorem 7.3 can be achieved without assigning more than $N^{3/4}$ processors to any one chip. Although this result will not be needed in this section, it will be used in §9. The proof is omitted because it is identical to the proof of Theorem 7.3, except that $N^{1/4}$ classes of $O(N^{3/4})$ processors each are used instead of three classes of fewer than $N/2$ processors each.

LEMMA 7.4. *If M is a shuffle-exchange computer with N processors, then it is possible to partition M onto $C = N^{1/2}$ chips with $O(N^{3/4})$ processors per chip and with $O(N^{1/2}/\log N)$ pins per chip.*

8. AKS computers.

Ajtai, Komlós, and Szemerédi created a sorting network for N items that has $O(N \log N)$ comparators and $O(\log N)$ depth [2]. The constants of proportionality hidden in the $O(N \log N)$ and $O(\log N)$ bounds are extremely large, so the network is not useful in practice. However, this network, which will be called the *AKS network*, is of great theoretical interest because all previously known sorting networks required $\Omega(\log^2 N)$ depth. As a result, the AKS network has been important in answering a number of questions about parallel sorting.

We will define a parallel computer, called the *AKS computer*, that is based on the AKS network. We begin with an AKS network for sorting r items, where $r = \Theta(N/\log N)$. This network receives r input lines from the left and produces the sorted items on r output lines on the right. The network consists of $\Theta(r \log r)$ comparators that are arranged in $d = \Theta(\log r)$ columns. The AKS computer is created by forming an $r \times d$ array of $rd = N$ processors. Each processor (i, j), where $0 \leq i < r$ and $0 \leq j < d$, is connected to processors $(i, (j+1) \bmod d)$ and $(i, (j-1) \bmod d)$ by horizontal connections. Also, each processor (i, j) is connected to processor (i', j) by a vertical connection if and only if wires i and i' are connected by a comparator in column j of the AKS network (see Fig. 4). Thus an AKS computer is basically an AKS network with processors instead of comparators and with wraparound connections between the input and output lines.

One notable application of the AKS computer was obtained by Leighton, who showed that an AKS computer with N processors can sort N items in $O(\log N)$ time [18]. Leighton's technique consists of performing sorts of r items in a pipelined manner plus performing four fixed permutations of the N items. Leighton proposed adding additional connections to implement these permutations, but they could also be performed on the AKS computer defined above without sacrificing the $O(\log N)$-time performance of the sort. This can be done by breaking each of the fixed permutations of N items into permutations of the r items located in the same column of the computer, followed by permutations of the $O(\log r)$ items located in the same row of the computer, followed by permutations of the r items located in the same column of the computer. This breaking

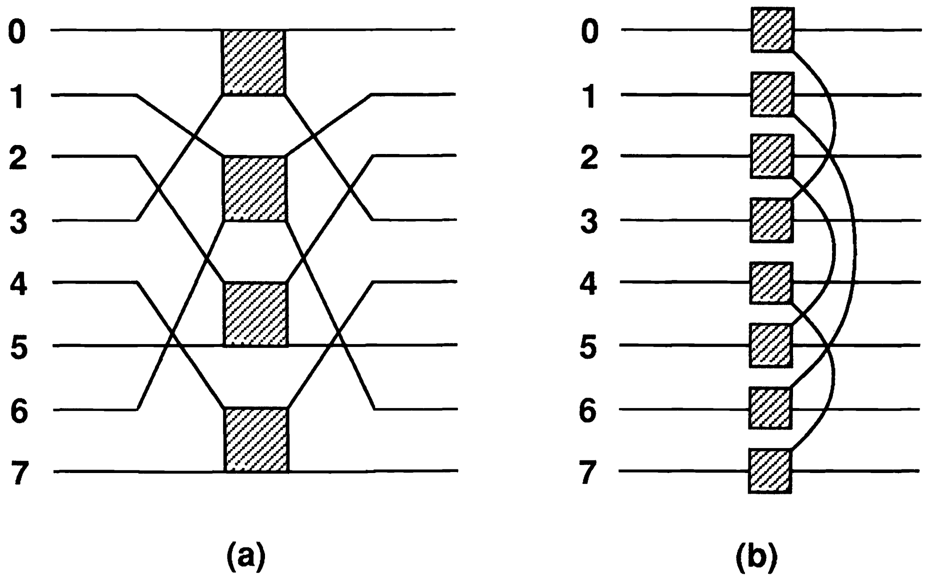


FIG. 4. *Relationship between* (a) *a column in the* AKS *network and* (b) *a column in the* AKS *computer.*

of a permutation into smaller permutations can be accomplished by using the Hall–Konig marriage theorem [33]. The permutations within rows can be implemented by simply using the horizontal connections to shift each item to its destination in $O(\log r)$ time. The permutations within columns can be implemented by sorting the items within each column according to their destination row. Of course, these sorts can also be pipelined and thus can be implemented in $O(\log r)$ time. This $O(\log N)$-time algorithm for sorting N items on an N-processor AKS computer will be called the Leighton–AKS algorithm.

The AKS network has been used to obtain minimum-area VLSI sorters. Bilardi and Preparata demonstrated that the N-input AKS sorting network has $O(N^2)$ area [3] (in the word model), and Leighton used this result and the Leighton–AKS algorithm to show that an $O(N^2/\log^2 N)$-area VLSI sorter that sorts N items in $O(\log N)$ time could be created [18]. This result matches the lower bound for sorting of $AT^2 = \Omega(N^2)$ proved by Thompson [30]. Thus despite its complex pattern of comparisons, the AKS computer requires the minimum possible amount of area for a VLSI sorter that operates in $O(\log N)$ time. In this section we will show that despite the optimality of the AKS computer from an area perspective, the AKS computer is suboptimal from a VLSI-pin-requirements perspective.

The detailed structure of an AKS computer is quite complicated and will not be repeated here. Instead, a few key characteristics of the AKS computer will be reviewed. First, recall that an AKS computer with N processors has the structure of an AKS sorting network for sorting $r = \Theta(N/\log N)$ items. The columns in an AKS computer are grouped into $\log r + 1$ stages. All of the connections from processors in stage i, where $0 \leq i \leq \log r$, are to processors in stages $i-1 \bmod (\log r+1)$, i, and $i+1 \bmod (\log r+1)$. The processors in each stage are located in r different rows, and connections between processors in different stages link processors that lie in the same row.

Each stage (except the first) is divided into three substages. During each substage the processors are partitioned into a number of groups, and comparisons are performed only within groups. Within each stage (except the first) the first and third substages use one partition, and the second substage uses a different partition. These partitions can be defined with the help of a complete binary tree having r leaves. Each node of the binary tree is assigned a natural interval in the range $0, \cdots, r-1$, where the root is given the entire interval, and the left and right children of a node are assigned the left and right halves, respectively, of that node's interval. We will use these overlapping natural intervals to define a partition of $0, \cdots, r-1$ into disjoint subintervals. More specifically, the nodes of the binary tree occupy $\log r + 1$ levels with the root being in level 0 and the leaves being in level $\log r$. During each stage i, $0 \le i \le \log r$, the nodes at level i of the tree are assigned their entire natural intervals, and nodes at level $i - j$, $1 \le j \le i$, are assigned two intervals, each of which is α^{-j} as long as the node's natural interval, located at each end of the node's natural interval ($\alpha \ge 8$ is a constant set to 100 in the original AKS paper [2]). The nodes at each level other than level 0 actually receive slightly less than their entire intervals, because the nodes at lower numbered levels have priority (see Fig. 5).

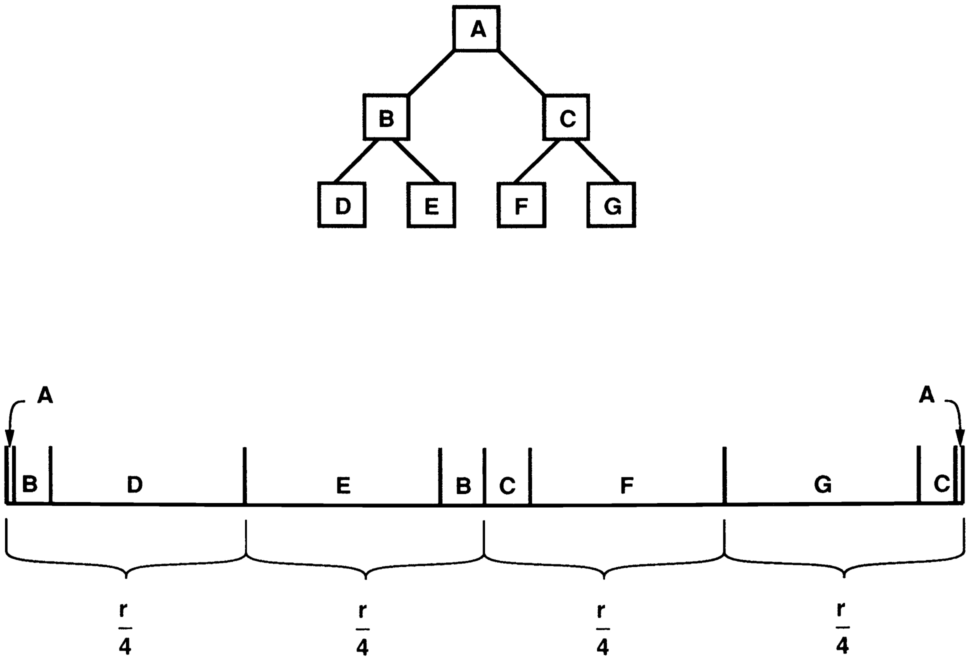


FIG. 5. *Binary tree and associated intervals for stage $i = 2$.*

This binary tree is then divided into triangles of three nodes each. During the first and third substages of a stage the binary tree is divided into triangles with apexes at even levels (see Fig. 6), but during the second substage of a stage the binary tree is divided into triangles with apexes at odd levels (see Fig. 7). During each substage the processors are partitioned by rows according to the intervals contained in the triangles in the binary tree. In the first stage all of the rows of processors lie in a single group.

Within each of the groups of a partition, an operation called a *nearsort* is performed. The details of the nearsort will be ignored, but two properties of the nearsort will be needed. First, comparisons are performed only within groups. Second, each group is divided into an upper half and a lower half, and a set of comparisons is performed of

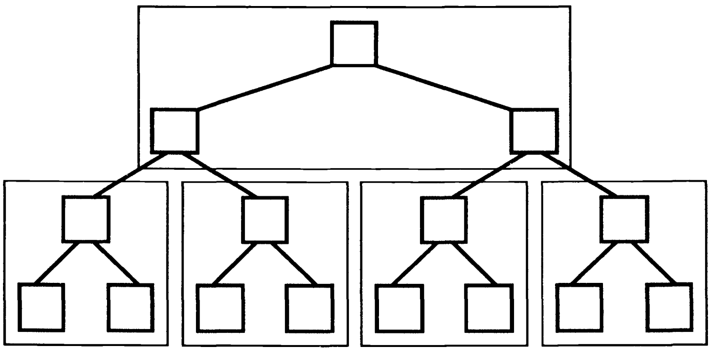


FIG. 6. *Division of binary tree for first and third substages.*

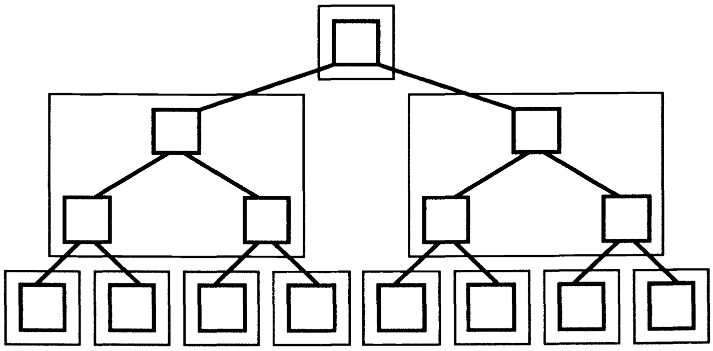


FIG. 7. *Division of binary tree for second substage.*

the two halves, which form a bipartite expander graph with parameters $((1-\epsilon)/\epsilon, \epsilon, \gamma)$, where $0 < \epsilon < \frac{1}{7}$ and $\gamma > 1$ are constants ($\epsilon = 10^{-9}$ in the original AKS paper [2]). A bipartite expander graph with $2w$ nodes and with parameters (x, y, z) has degree at most z, and yet for each set S consisting of wy or fewer nodes of one partite set, at least $x|S|$ nodes have connections to nodes in S. Such a set S will be said to be *expanded* by a factor of x. The above properties of the AKS computer will be used to prove upper and lower bounds on the pin requirements of AKS computers.

THEOREM 8.1. *If M is an AKS computer with N processors, then it holds that $Q = O(N \log C/C \log N)$.*

Proof. Recall that the processors can be viewed as forming a rectangular array with $r = \Theta(N/\log N)$ rows. If $C \geq r$, then $\log C = \Theta(\log N)$ and the claim is that $Q = O(N/C)$, which is trivially accomplished by assigning roughly equal numbers of processors to each chip. If $C < r$, then let $c = 2^{\lfloor \log C \rfloor}$ and partition the r rows of processors into c sets each containing r/c adjacent rows (it is assumed that r is a power of 2). It will be shown that this partition is legal and has only $O(N \log C/C \log N)$ pins per chip. First, note that all horizontal connections are local to chips, so only vertical connections require pins. Next, note that stages $0, \cdots, \log c$ contribute only $O(r \log c/c)$ processors to each chip, and thus they contribute only $O(r \log c/c)$ pins to each chip.

Now consider the processors in stages $\log c + 1, \cdots, \log r$. In the binary tree that determines the partitions used by the AKS network, the natural intervals of the nodes at levels $\log c + 1, \cdots, \log r$ lie entirely within the blocks of r/c rows that are assigned to single chips. As a result, only levels $0, \cdots, \log c$ (which will be called the *upper levels*) correspond to processors that will require pins for their vertical connections. There are at most kr/c (where k is a constant determined by the AKS network) processors per chip in the upper levels during stage $\log c + 1$, at most kr/ck' (where $k' > 1$ is a constant determined by the AKS network) processors per chip in the upper levels during stage $\log c + 2$, and, in general, at most $kr/c(k')^{i-1}$ processors per chip in the upper levels during stage $\log c + i$, where $1 \leq i \leq \log r - \log c$. Therefore, there are a total of at most $O(r/c)$ processors per chip in the upper levels during stages $\log c + 1, \cdots, \log r$ combined and at most $O(r/c) + O(r \log c/c) = O(N \log C/C \log N)$ pins per chip. Finally, the partition is legal because $c \geq 4$ and each chip has at most N/c processors. $\square$

We will now prove a matching lower bound on the pin requirements of an AKS computer. Our proof will require a number of definitions. First, let r be such that an AKS computer with N processors has the connections of an AKS sorting network with r inputs. Recall that $r = \Theta(N/\log N)$. Let α and ϵ be constants given by the AKS network construction as defined above. Let β be the maximum number of processors in any single row and stage of an AKS computer. Let $k_1 = \log(4/\epsilon)$, and let k_2 be such that an AKS computer with N processors requires at most $k_2 \log N$ time to sort N items by using the Leighton–AKS algorithm. Let k_3 be such that for any parallel computer, if T is the worst-case time required to sort N items, then $TCQ \geq k_3 N$ (see Theorem 4.1). Let $k_4 = (k_2 + 1)^4 + 2^{4k_1}$, let k_5 be such that $r \leq k_5 N/\log N$, and let $k_6 = \min(1/16\beta, 1/k_5, k_3/k_2k_4k_5)$. Note that α, β, ϵ, k_1, k_2, k_3, k_4, k_5, and k_6 are constants. Also, notice that $\alpha \geq 8$, $\beta \geq 1$, $0 < \epsilon < 1/7$, $(1-\epsilon)/\epsilon > 6$, $k_1 \geq 4$, $k_4 \geq 2^{16}$, and $k_6 \leq 1$.

The following lemmas will be helpful in obtaining the lower bound on Q.

LEMMA 8.2. *If M is an AKS computer with N processors and if P is a legal partition of M onto C chips, then either* $\mathrm{Girth}(M, P) \geq k_6 r \log C/C$ *or* $C > k_4$.

Proof. Assume for the sake of contradiction that the claim is false, in which case $C \leq k_4$ and $\mathrm{Girth}(M, P) < k_6 r \log C/C$. Let T be the time required to implement the Leighton–AKS algorithm on M. Because $Q \leq \mathrm{Girth}(M, P)$, $TCQ \geq k_3 N$, and $T \leq k_2 \log N$, it follows that

$$\begin{aligned} \mathrm{Girth}(M, P) &\geq k_3 N/TC \\ &\geq k_3 N/k_4 k_2 \log N \\ &\geq k_3 r/k_2 k_4 k_5 \\ &\geq k_6 r \\ &\geq k_6 r \log C/C \end{aligned}$$

which is a contradiction. $\square$

LEMMA 8.3. *If M is an AKS computer with N processors and if P is a legal partition of M onto C chips, then either* $\mathrm{Girth}(M, P) \geq k_6 r \log C/C$ *or each chip contains processors from at most $r/C^{1/2}$ different rows of the AKS computer.*

Proof. Assume for the sake of contradiction that the claim is false, in which case $\mathrm{Girth}(M, P) < k_6 r \log C/C$ and there exists some chip x that contains processors from more than $r/C^{1/2}$ different rows. We will classify each row of the AKS computer according to how many of the processors in that row are assigned to chip x. Let a row of processors be a *full row* if all of the processors in that row are assigned to chip x, let a row of processors be a *partial row* if some but not all of the processors in that row are assigned to chip x, and let a row be an *empty row* otherwise. First, we will show that there are fewer than $k_2 r \log C/C$ full rows. This is true because if there are at least $k_2 r \log C/C$ full rows, then chip x must contain at least $k_2 N \log C/C$ processors. Because the items from chip x can be moved to other chips in $k_2 \log N$ time by using the Leighton–AKS algorithm, chip x must have at least $N \log C/C \log N \geq r \log C/k_5 C \geq k_6 r \log C/C$ pins, which would imply that $\text{Girth}(M, P) \geq k_6 r \log C/C$, which would be a contradiction. Therefore, there are fewer than $k_2 r \log C/C$ full rows.

Because chip x contains processors from more than $r/C^{1/2}$ different rows, there must be more than $r/C^{1/2} - k_2 r \log C/C$ partial rows. But from Lemma 8.2 $C > k_4 \geq 2^{16}$ so $C^{1/4} - k_2 \geq 1$, $\log C \leq C^{1/4}$ and $(\log C)/C \leq 1/C^{3/4}$. Therefore, $r/C^{1/2} - k_2 r \log C/C \geq (r/C^{3/4})(C^{1/4} - k_2) \geq r/C^{3/4} \geq r \log C/C$. But chip x must have at least one pin per partial row because the processors in each row are connected in a cycle. Therefore, $\text{Girth}(M, P) \geq r \log C/C \geq k_6 r \log C/C$, which is a contradiction. $\square$

THEOREM 8.4. *If M is an AKS computer with N processors, then it follows that $Q = \Omega(N \log C/C \log N)$.*

Proof. Assume for the sake of contradiction that there exists a legal partition $P = P_0, \cdots, P_{C-1}$ of M onto C chips such that $\text{Girth}(M, P) < k_6 r \log C/C$. Note that from Lemma 8.2 $C > k_4$.

Consider an arbitrary stage i of the AKS computer, where $1 \leq i \leq \frac{1}{2} \log C - k_1$. If i is even, let $j = 1$, and if i is odd, let $j = 2$. As was discussed earlier, the processors in each substage of each stage are partitioned with the use of a binary tree. In the first substage of each stage the nodes in the binary tree are divided into triangles with apexes at even levels, and in the second substage of each stage they are divided into triangles with apexes at odd levels. Thus in substage j of stage i the nodes in the binary tree are divided so that each node in level i of the binary tree forms its own group. The natural interval for each node in level i of the binary tree has size $r/2^i$. Although each node in level i does not receive all of its natural interval (part of the interval is taken by nodes at lower numbered levels), it does receive at least

$$1 - \sum_{h=1}^{i} (2/\alpha)^h \geq 1/2$$

of its natural interval (because each level $i - h$, $1 \leq h \leq i$, takes a fraction of at most $(2/\alpha)^h$ of the natural interval of a node at level i). Let the intervals assigned to nodes at level i of the binary tree be called *primary intervals*, and let the upper and lower halves of a primary interval be called *half intervals*. Because $1 \leq i \leq \frac{1}{2} \log C - k_1$ and because each node at level i receives at least half of its natural interval, each primary interval contains at least $r/2^{i+1} \geq r(4/\epsilon)/2C^{1/2} = 2r/C^{1/2}\epsilon$ rows and each half interval contains at least $r/C^{1/2}\epsilon$ rows.

Now consider an arbitrary primary interval y and an arbitrary chip P_x, $0 \leq x < C$. Let V be the set of processors in P_x that are in substage j of stage i, and let W be the set of processors in V that are in rows associated with primary interval y. Let S be the set of rows in the AKS computer that contain one or more processors in W. Note that from Lemma 8.3 $|S| \leq r/C^{1/2}$. Each row in set S will be called a *complete row* if and only if all of the processors in that row that are in substage j of stage i are in P_x. Let U be the set of rows in S that are complete. There are two cases:

Case 1: $|U| \geq |S|/2$. In this case let U_1 and U_2 be the subsets of U contained in the upper and lower half intervals of y, respectively. Assume without loss of generality that $|U_1| \geq |U_2|$. Because $|U_1| \leq |S| \leq r/C^{1/2}$ and the half interval containing U_1 has at least $r/C^{1/2}\epsilon$ rows, U_1 consists of at most a fraction of ϵ of the rows in the half interval, and as a result the rows in U_1 are expanded by a factor of $(1 - \epsilon)/\epsilon$ (that is, they are connected to at least $|U_1|(1 - \epsilon)/\epsilon$ rows in the lower half interval). However, at most $|S|$ rows in the lower half interval contain processors in W, so there are at least $(|U_1|(1-\epsilon)/\epsilon) - |S| > (|S|/4)(6) - |S| = |S|/2$ vertical connections between processors in W and processors not in P_x. Therefore, chip P_x must have at least $|S|/2$ pins for the processors in W, which is an average of at least $1/2\beta$ pins per processor in W.

Case 2: $|U| < |S|/2$. In this case there are at least $|S| - |U| \geq |S|/2$ horizontal connections between processors in W and processors not in P_x. Therefore, chip P_x must have at least $|S|/2$ pins for the processors in W, which is an average of at least $1/2\beta$ pins per processor in W.

Thus in either case chip P_x must have at least $1/2\beta$ pins per processor in W. Because the chip P_x and primary interval y were chosen arbitrarily and because there are at least $r/2$ processors in substage j of stage i that are in some primary interval, there must be a total of at least $r/4\beta$ pins required by the processors in substage j of stage i. Note that $C > k_4 \geq 2^{4k_1}$, so $\frac{1}{2}\log C - k_1 \geq \frac{1}{4}\log C$. Because i is an arbitrary stage in the range $1 \leq i \leq \frac{1}{2}\log C - k_1$, there are a total of at least $\frac{1}{4}(\log C)(r/4\beta) = r\log C/16\beta \geq k_6 r \log C$ pins and some chip has at least $k_6 r \log C/C$ pins. Therefore, $\mathrm{Girth}(M, P) \geq k_6 r \log C/C$, which is a contradiction. As a result, for any legal partition P, $\mathrm{Girth}(M, P) \geq k_6 r \log C/C$, which implies that $Q = \Omega(N \log C/C \log N)$. □

An immediate consequence of Theorem 8.4 is that an AKS computer is suboptimal with respect to its TCQ product for the worst-case sorting problem. To show this, we will need the following lemma proved by Cypher and Plaxton [9].

LEMMA 8.5. *The worst-case time required to sort N items on an N-processor cube-connected-cycles or shuffle-exchange computer is* $O(\log N(\log\log N)^2)$.

Now let T be the worst-case time required to sort N items, and consider the case for which $C = N^{1/3}$. In this case for an AKS computer $T = \Theta(\log N)$ and $Q = \Theta(N^{2/3})$, so $TCQ = \Theta(N \log N)$. In contrast, for a cube-connected-cycles or shuffle-exchange computer $T = O(\log N(\log\log N)^2)$ and $Q = O(N^{2/3}/\log N)$, so $TCQ = O(N(\log\log N)^2)$. As a result, the TCQ product of an AKS computer is suboptimal for the worst-case sorting problem.

9. AKS algorithm lower bound.

In this section we will prove a lower bound on the running time of the AKS sorting algorithm when it is implemented on a cube-connected-cycles or shuffle-exchange computer. This lower bound is based on the bounds for the pin requirements of the AKS, cube-connected-cycles, and shuffle-exchange computers presented earlier.

THEOREM 9.1. *If M is a cube-connected-cycles or shuffle-exchange computer with N processors, then the time required to sort N items on M by using the AKS algorithm is* $\Omega(\log^2 N)$.

Proof. The proof follows from Theorems 3.7 and 8.4 and Lemma 7.4 with the parameter C set to $N^{1/2}$. From Theorem 3.7 and Lemma 7.4 the pin requirements of M are $Q = O(N/C \log(4N/C)) = O(N^{1/2}/\log N)$. The AKS network with N inputs can be viewed as being the data-flow graph for the AKS algorithm, where each comparator in the network represents a comparison operation in the graph and each link in the network represents the transmission of a data value from one comparison to another. Any time T implementation of the AKS algorithm on M will require that this data flow graph be mapped to M with at most T nodes of the graph being assigned to each processor and at most T edges of the graph lying on each communication link of M. Note that the ith largest item, $0 \leq i < N$, must be moved to processor i, which is the processor that originally had the ith input. This corresponds to the existence of wraparound edges in the data-flow graph. Thus a time T implementation of the AKS algorithm will also define a partition of an $O(N \log N)$ node AKS network (with wraparounds) onto $N^{1/2}$ chips with girth at most $QT = O(TN^{1/2}/\log N)$. From Lemma 7.4 and the proof of Theorem 3.7 each chip contains at most $N^{3/4}$ processors from M. Therefore, when $T = O(\log^2 N)$ the resulting partition of the AKS network (with wraparounds) is legal. However, from Theorem 8.4 an $O(N \log N)$ node AKS network (with wraparounds) that is legally partitioned onto $N^{1/2}$ chips must have a girth of at least $\Omega(N^{1/2} \log N)$. Thus $TN^{1/2}/\log N = \Omega(N^{1/2} \log N)$ and $T = \Omega(\log^2 N)$. $\square$

It is quite easy to attain the lower bound given in Theorem 9.1. The AKS sorting algorithm performs $O(\log N)$ stages of comparisons, where the pattern of the comparisons is fixed. As a result, each stage can be implemented by first permuting the data items so that items that need to be compared are in connected processors. Because these permutations are fixed, each of them can be implemented in $O(\log N)$ time on a cube-connected-cycles or shuffle-exchange computer [22], [26]. This implementation yields the following theorem.

THEOREM 9.2. *If M is a cube-connected-cycles or shuffle-exchange computer with N processors, then the time required to sort N items on M by using the AKS algorithm is $O(\log^2 N)$.*

Acknowledgments. The author thanks Jorge Sanz, Marc Snir, and Gianfranco Bilardi for their helpful discussions and comments.

REFERENCES

[1] A. AGGARWAL AND J. VITTER, *The I/O complexity of sorting and related problems* (extended abstract), in Proc. 14th International Colloquium on Automata, Languages and Programming, Karlsruhe, Germany, Vol. 267, Springer–Verlag Lecture Notes in Computer Science, 1987, pp. 467–478.

[2] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, *An $O(n \log n)$ sorting network*, Combinatorica, 3 (1983), pp. 1–19.

[3] G. BILARDI AND F. PREPARATA, *The VLSI optimality of the AKS sorting network*, Inform. Process. Lett., 20 (1985), pp. 55–59.

[4] B. BOLLOBÁS AND I. LEADER, *An isoperimetric inequality on the discrete torus*, SIAM J. Discrete Math., 3 (1990), pp. 32–37.

[5] ———, *Compressions and isoperimetric inequalities*, J. Combin. Theory Ser. A, 56 (1991), pp. 47–62.

[6] L. CIMINIERA AND A. SERRA, *LSI Implementation of modular interconnection networks for MIMD machines*, in Proc. International Conference on Parallel Processing, 1980, pp. 161–162.

[7] T. CORMEN, *Efficient multichip partial concentrator switches*, in Proc. International Conference on Parallel Processing, 1987, pp. 525–532.

[8] R. CYPHER, *Theoretical aspects of VLSI pin limitations*, Tech. Report 89-02-01, Department of Computer Science, University of Washington, Seattle, WA, 1989.

[9] R. CYPHER AND G. PLAXTON, *Deterministic sorting in nearly logarithmic time on the hypercube and related computers*, in Proc. 22nd ACM Symposium on Theory of Computing, 1990, pp. 193–203.

[10] M. FRANKLIN AND S. DHAR, *Interconnection networks: Physical design and performance analysis*, J. Par. Distr. Comput., 3 (1986), pp. 352–372.

[11] M. FRANKLIN, D. WANN, AND W. THOMAS, *Pin limitations and partitioning of VLSI interconnection networks*, IEEE Trans. Comput., C-31 (1982), pp. 1109–1116.

[12] S. HART, *A note on the edges of the n-cube*, Discrete Math., 14 (1976), pp. 157–163.

[13] J. HAYES, T. MUDGE, Q. STOUT, S. COLLEY, AND J. PALMER, *Architecture of a hypercube supercomputer*, in Proc. International Conference on Parallel Processing, 1986, pp. 653–660.

[14] J. HONG AND H. KUNG, *I/O complexity: The red-blue pebble game*, in Proc. 13th ACM Symposium on Theory of Computing, 1981, pp. 326–333.

[15] S. KNAUER, J. O'NEILL, AND A. HUANG, *Self-routing switching network*, in Principles of CMOS VLSI Design, N. Weste and K. Eshraghian, eds., Addison-Wesley, Reading, MA, 1985.

[16] R. KOCH, T. LEIGHTON, B. MAGGS, S. RAO, AND A. ROSENBERG, *Work-preserving emulations of fixed-connection networks* (extended abstract), in Proc. 21st ACM Symposium on Theory of Computing, 1989, pp. 227–240.

[17] T. Leighton, *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press, Cambridge, MA, 1983.
[18] ———, *Tight bounds on the complexity of parallel sorting*, IEEE Trans. Comput., C–34 (1985), pp. 344–354.
[19] H. Li and M. Maresca, *Polymorphic-torus network*, in Proc. International Conference on Parallel Processing, 1987, pp. 411–414.
[20] R. McMillen and H. Siegel, *Evaluation of cube and data manipulator networks*, J. Par. Distr. Comput., 2 (1985), pp. 79–107.
[21] D. Nassimi and S. Sahni, *Data broadcasting in SIMD computers*, IEEE Trans. Comput., C–30 (1981), pp. 101–107.
[22] F. Preparata and J. Vuillemin, *The cube-connected cycles: A versatile network for parallel computation*, Comm. ACM, 24 (1981), pp. 300–309.
[23] J. Reif and L. Valiant, *A logarithmic time sort for linear size networks*, J. Assoc. Comput. Mach., 34 (1987), pp. 60–76.
[24] C. Seitz, *The cosmic cube*, Comm. ACM, 28 (1985), pp. 22–33.
[25] M. Snir, *I/O limitations on multi-chip VLSI systems*, in Proc. 19th Allerton Conference on Communications, Control and Computing, Monticello, IL, sponsored by University of Illinois at Urbana-Champaign, 1981, pp. 224–233.
[26] H. Stone, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C–20 (1971), pp. 153–161.
[27] T. Szymanski, *A VLSI comparison of switch-recursive Banyan and crossbar interconnection networks*, in Proc. International Conference on Parallel Processing, 1986, pp. 192–199.
[28] C. Thompson, *Area-time complexity for VLSI*, in Proc. 11th ACM Symposium on Theory of Computing, 1979, pp. 81–88.
[29] ———, *A complexity theory for VLSI*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.
[30] ———, *The VLSI complexity of sorting*, IEEE Trans. Comput., C-32, 1983, pp. 1171–1184.
[31] J. Ullman, *Flux, sorting and supercomputer organization for AI applications*, J. Par. Distr. Comput., 1 (1984), pp. 133–151.
[32] J. Vuillemin, *A combinatorial limit to the computing power of V.L.S.I. circuits*, in Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 294–300.
[33] A. Waksman, *A permutation network*, J. ACM, 15 (1968), pp. 159–163.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 379–394, April 1993

GAP THEOREMS FOR DISTRIBUTED COMPUTATION*

SHLOMO MORAN† AND MANFRED K. WARMUTH‡

Abstract. Consider a bidirectional ring of n identical processors that communicate asynchronously. The processors have no identifiers, and hence the ring is called anonymous. Each processor receives an input letter, and the ring is to compute a function of the circular input string. If the function value is constant for all input strings, then the processors do not need to send any messages. On the other hand, it is proven that any deterministic algorithm that computes any nonconstant function for anonymous rings requires $\Omega(n \log n)$ bits of communication for some input string. Also exhibited are nonconstant functions that require $O(n \log n)$ bits of communication for every input string. The same gap for the bit complexity of nonconstant functions remains even if the processors have distinct identifiers, provided that the identifiers are taken from a large enough domain.

When the communication is measured in messages rather than bits, the results change. A nonconstant function that can be computed with $O(n \log^* n)$ messages on an anonymous ring is presented.

Key words. distributed algorithms, networks, asynchronous, message complexity, bit complexity, ring of processors, lower bounds, gap theorem

AMS(MOS) subject classifications. 68M10, 68Q10, 68Q15, 68Q25

1. Introduction. There has been an extensive amount of research on studying computation on a ring of n asynchronous processors. The ring topology is in a sense the simplest distributed network that produces many typical phenomena of distributed computation. In this model the processors may communicate by sending messages along the links of the ring, which are either unidirectional or bidirectional. All the messages sent reach their targets after a finite, but unpredictable and unbounded delay. Numerous algorithms [ASW88], [DKR82], [P82] have been found for this *asynchronous ring* model. All these algorithms require the transmission of $\Omega(n \log n)$ bits. This is not surprising in view of the results of this paper. We establish a gap theorem for asynchronous distributed computation on the ring which says that either the function computed is constant and no messages need to be sent, or the function is nonconstant and there is an input string requiring $\Omega(n \log n)$ bits.

In our model there is no leader among the processors. All processors receive a letter from some input alphabet as their input and run the same deterministic program, which may depend on the ring size. We first treat the case where the ring is unidirectional and the processors have no identifiers (the anonymous model of [ASW88]). Then we show that the lower bound holds also for bidirectional anonymous rings, and for rings of processors with distinct identifiers, provided the set of possible identifiers is sufficiently large. Note that in the anonymous ring without a leader it is necessary that the processors "know" some bound on the ring size. Otherwise the processors cannot determine when to terminate [ASW88].

Let us contrast the anonymous ring with the model of a ring with a leader (which is also the initiator of the algorithm). If we assume unidirectional communication, then any nonconstant function that depends on the input of any processor other than the input of leader requires $\Omega(n)$ bits. In the bidirectional case, there are simple nonconstant

*Received by the editors September 28, 1988; accepted for publication (in revised form) January 16, 1992. A preliminary version of this paper appeared in [MW86].

†Department of Computer Science, the Technion, Haifa 32000, Israel. Part of this work was done while the author was at IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

‡Department of Computer and Information Sciences, University of California, Santa Cruz, California 95064. The research of this author was supported by Faculty Research funds granted by the University of California, Santa Cruz, and by Office of Naval Research grants N00014-86-K-0454 and N00014-91-J-1162.

functions for any bit complexity $\Theta(b(n))$. If $b(n) \geq n^2$, such a function is achieved using nonconstant size input alphabet. We now show how to achieve such a function with $b(n) < n^2$. In this case the input letters are bits. The input to the ring is denoted as an n-bit cyclic string ω, the ith bit ω_i of ω being the input letter of the ith processor. Using a crossing sequence argument on pairs of links we can show that the following function is a nonconstant function of bit complexity $\Theta(b(n)) : f(\omega) = 1$ if and only if ω contains a palindrome of

$$2\left\lceil\sqrt{b(n)}\right\rceil + 1$$

bits centered at the leader (the same function was first described in [MZ87], where it was used for a similar purpose). Thus there is no gap for rings with a leader. Our gap theorem for anonymous rings clearly quantifies the price we have to pay for having no distinguished processor.

In the model where there is a leader but the size of the ring is unknown, a different sort of "gap" has been found by Mansour and Zaks [MZ87]. They showed that a language is accepted in this model in $O(n)$ bit complexity if and only if it is regular, and every nonregular language requires $\Omega(n \log n)$ bits. These results of [MZ87] are analogous to the classical results for one-tape Turing machines [T64], [H68]. Note that the bit complexity of nonregular language coincides with the bit complexity required for computing the size of the ring. As discussed above, if the size of the ring is known, then for any bit complexity, there is a language with that complexity on rings with a leader.

The lower bound of $\Omega(n \log n)$ for anonymous rings proven here does not hold if one counts *messages* (of arbitrary length) instead of bits. In fact, we show in this paper that if the input alphabet is of size at least n, then there are simple nonconstant functions that can be computed in $O(n)$ messages, for arbitrary ring size. Considering the message complexity is more interesting in the case when the input alphabet is of constant size, independent of the ring size n. In [ASW88] a nonconstant function was presented that is computable in $O(n)$ messages on an anonymous ring when the inputs are bits. However, this function is only defined for rings of odd size. It is easy to find similar functions for rings whose sizes are not divisible by some fixed integer. When the smallest nondivisor of the ring size is large, the ring contains more inherent symmetry, and it is hard to find nonconstant functions of low message complexity. We exhibit a nonconstant function with binary inputs for arbitrary ring size that can be computed[1] in $O(n \log^* n)$ messages. For some values of n, a matching lower bound of $O(n \log^* n)$ was proven for computing any nonconstant function [DG87]. The lower bound assumes that the input alphabet is finite.

Our lower bound proofs rely heavily on the asynchronous nature of the computation. We use the fact that the result of a computation must be independent of the specific delay times of the links, and impose certain delays on the links that assure that $\Omega(n \log n)$ bits are sent. In contrast, on synchronous anonymous rings, the Boolean *AND* can be computed with $O(n)$ bits [ASW88], and $\Omega(n)$ is also a trivial lower bound for an arbitrary nonconstant function.

Our research opens many challenging problems concerning gap theorems in distributed computation. Given an asynchronous network of anonymous processors, define the *distributed bit* (*message*) *complexity* of the network to be the smallest bit (message) complexity of a nonconstant function when computed on that network. Intuitively, this complexity measures the minimum effort needed to coordinate the processors of the net-

[1]The function $\log^* n$ grows very slowly (≤ 5 for $n \leq 2^{65536}$). It is the number of iterations of the function $\log_2$ required to get the value n down to 1 or below.

work in any sensible way.[2] This coordination should be more difficult if the network is highly symmetric. What parameters of the network correspond to this complexity? How does this complexity depend on the connectivity, diameter, or other properties of the network? Our results show that the distributed bit complexity of a ring of n processors is $\Theta(n \log n)$. The distributed bit complexity of the torus was recently shown to be linear in the number of processors [BB89].

In the next section we summarize the model of computation. We first prove the $\Omega(n \log n)$ lower bound on the bit complexity for unidirectional rings with no identifiers (§3), and then we generalize the result to bidirectional rings (§4) and to rings with identifers (§5). In the last section we show that for each n, there are nonconstant functions that can be computed on a ring of n anonymous processors by algorithms of $O(n \log n)$ bit complexity and $O(n \log^* n)$ message complexity. The latter function is defined by interleaving de Bruijn sequences of various lengths. We also show there that if we allow the alphabet size to be $\Omega(n)$, then there is a nonconstant function (defined uniformly for all ring sizes) of linear message complexity.

The Gap Theorem with distinct identifiers assumes that the identifiers are chosen from a set of double exponential size. Similar theorems for small identifier sets are considered in a subsequent paper [BMW91]. Also Gap Theorems for probabilistic models have been recently shown in [AAHK89].

2. The model. Our computational model is a ring R of n processors, $p_1, p_2, \ldots, p_n$. The processors are *anonymous*, i.e., they do not have identifiers, and they run the same deterministic program which may depend on the ring size. In particular, the program of p_i does not depend on its index i. Consecutive processors (as well as the last processor p_n and the first processor p_1) are connected by communication *links*, and thus each processor has two *neighbors*. A processor can distinguish between its two neighbors; one is called the *left* and the other one the *right* neighbor. An *orientation* of the ring is a particular assignment of left and right to the links of each processor. If the notions of left and right of all processors are consistent, then we say the ring is *oriented*. In the *bidirectional* ring connected processors can send and receive messages to and from both neighbors, respectively. In the *unidirectional* ring we assume that the ring is oriented and that messages can only be sent to the right and received from the left, that is, messages can only travel in one direction around the ring.

We assume that messages are encoded as nonempty bit strings. The messages sent along a fixed direction of a link arrive in the order in which they were sent. The communication is *asynchronous*, meaning that messages arrive with an unpredictable but finite delay time. We assume that any nonempty subset of the processors may wake up spontaneously and start running the algorithm. Processors that do not wake up spontaneously are awakened upon receiving a message from a neighbor.

In our model, the input of each processor is a letter of an arbitrary alphabet I, i.e., the functions have the domain I^n. We shall assume that I contains the letter 0. An *execution* of an algorithm on a unidirectional or bidirectional ring consists of (i) an input assignment to each processor, (ii) in the case of a bidirectional ring an orientation of the ring, and (iii) a combined schedule of how the algorithm is run at each processor. The schedule includes wake up times for each processor, the exact times for each step taken by the individual runs of the algorithm and times for when each message was sent and received. The schedule has to satisfy the requirements of asynchronous ring computations, as described above.

[2]This complexity might also depend on the size of the input alphabet over which the functions are defined.

An algorithm *computes* a function f (which is defined for a particular ring size[3] n) if for every input string ω in I^n and each execution on the ring labeled with ω every processor outputs $f(\omega)$. For any nonconstant function f there is a string $\omega \in I^n$ such that $f(\omega) \neq f(0^n)$. For any algorithm that computes f we say for convenience that if the input is ω (respectively, 0^n), then all processors running AL reach an *accepting* (respectively, *rejecting*) state.

Note that functions computed on a ring without a leader must be invariant under circular shifts of the input string and in case of (unoriented) bidirectional ring also under reversal of the input string [ASW88]. The *bit* (*message*) *complexity* of any algorithm is the maximal number of bits (messages) sent over all possible executions of this algorithm. The bit (message) complexity of a function f is the minimal possible bit (message) complexity of any algorithm that computes f. The bit (message) complexity of a given network is the minimal possible bit (message) complexity of any nonconstant function when computed on that network. The lower bound we prove on the bit complexity of bidirectional rings also holds when the bidirectional rings are oriented. All algorithms presented in this paper are for unidirectional rings. We discuss how they can be converted to algorithms of similar bit and message complexities that work on unoriented bidirectional rings.

Proof method for the lower bounds. We consider an arbitrary algorithm AL that computes a nonconstant function on a ring of size n, and we construct executions of AL on lines of processor of various lengths. By cutting and pasting such executions we get an execution of AL on a ring of size n in which at least $cn \log n$ bits are sent for some constant c independent of n. We first present the lower bound for the unidirectional case since it helps us understand the more complicated bidirectional case.

3. The lower bound for unidirectional rings.

THEOREM 1. *The bit complexity of a unidirectional ring of n anonymous processors is* $\Omega(n \log n)$.

The proof of Theorem 1 will follow from some lemmas given below. We will give lower bounds on the worst case complexity of any algorithm AL that accepts some string $\omega \in I^n$ and rejects 0^n. The first lemma (which holds for both the unidirectional and bidirectional case) is similar to Theorem 5.1 of [ASW88]. We repeat its proof here to make the paper more self-contained. Since the function value is independent of the delay times of the asynchronous computation, we may choose any delay times for the proofs: we assume that the ring is oriented, all processors start at time zero, internal computation at a processor takes no time, and links are either *blocked* (very large delay) or are *synchronized* (it takes exactly one time unit to traverse the link). For the lemma below we assume that all links are synchronized. Intuitively, this keeps the execution symmetric and causes the most messages to be sent.

LEMMA 1. *Let AL be an algorithm for a unidirectional or bidirectional ring of n processors. If AL rejects 0^n and accepts $0^z \cdot \tau$ for some τ, then AL requires at least $n\lceil z/2 \rceil$ messages on input 0^n.*

Proof. Consider the synchronized execution of AL with input string 0^n. Since all processors have the same input letter and run the same algorithm, at any given time all the processors are in the same state of the algorithm. Hence, at least one message is sent by each processor at each integral time until some time T at which no message is sent. From then on the processors are inactive because no new messages are received. Thus

[3]In §6 (the last section) and only in that section we present algorithms for computing nonconstant functions that are defined for more than one ring size. In that case we give the algorithm the ring size as an argument.

the processors terminate at time T after sending at least nT messages altogether. Now consider a second execution with input string $O^z \cdot \tau$. If $T < \lceil z/2 \rceil$, then the processor $p_{\lceil z/2 \rceil}$ is in the same state of the algorithm at time step T in both executions. Thus in both cases $p_{\lceil z/2 \rceil}$ terminates with the same result, which is a contradiction. We conclude that at least $n\lceil z/2 \rceil$ messages are sent in the execution with 0^n as the input string. □

Consider an execution of the algorithm AL on ring R with ω as input string and all links synchronized. The sequence of messages sent by an anonymous processor in such an execution is uniquely determined[4] by its input letter and the sequence of messages received by the processor, which we call the "history" of the processor. Suppose all processors terminate before some time t. For $0 \leq s \leq t$ and for $1 \leq i \leq n$, we define the *history* $h_i(s)$ as the string $m_i(1)L \cdots Lm_i(r_s)$, where L is a separating symbol, and $m_i(1), \ldots, m_i(r_s)$ are the messages (nonempty bit strings) received by p_i until (and including) time s, in chronological order. Note that r_s might be smaller than s. $H_i = h_i(t)$ is the *total history* of p_i in this execution. Since the messages are nonempty bit strings, the total length of H_i is less than twice the number of bits received by p_i. Thus, a lower bound on the bit complexity of AL is implied by a lower bound on the sum of the lengths of the total histories of the processors in a certain execution of AL. The lower bound on the sum of the lengths will follow from the fact that during a certain execution of AL the number of distinct histories of the processors is $\Omega(n)$, and therefore the lemma below implies the $\Omega(n \log n)$ bound.

LEMMA 2. *Let $H_1, \ldots, H_l$ be l distinct strings over an alphabet of size $r > 1$. Then $|H_1| + |H_2| + \cdots + |H_l| \geq (l/2)\log_r(l/2)$.*

Proof. Represent the H_i with an order r-ary tree, such that each H_i corresponds to a path from the root to an internal node or a leaf of the tree. In the tree each leaf is responsible for some H_i. Assume the overall length of the strings H_i is minimized and the leaves at the lowest level of the tree are as far to the left as possible. Then in the corresponding tree each leaf and each internal node is responsible for some H_i. Furthermore, all but possibly one internal node have out-degree r, and hence at least half of all nodes are leaves. The lemma is implied by the fact that the average height of the leaves in an r-ary tree with v leaves is at least $\log_r v$. □

Outline of the proof of Theorem 1. An execution of AL is constructed for which either Lemmas 1 or 2 implies the lower bound of $\Omega(n \log n)$ bits. In the first case a processor accepts an input string that contains $\log n$ consecutive zeros. Thus Lemma 1 implies that $\Omega(n \log n)$ messages are required for input 0^n. In the second case there will be an execution with more than $n - \log n$ processors with distinct histories and Lemma 2 gives an $\Omega(n \log n)$ bits lower bound.

Let k be an integer such that all the processors have terminated before time $t = kn$ in the synchronized execution of AL on ω, and let C be a line of kn processors, denoted by $p_{1,1}, p_{2,1}, \ldots, p_{n,1}, p_{1,2}, \ldots, p_{n,k}$. Informally, C consists of k copies of the ring R of n processors that were cut at the link $p_n - p_1$ and then concatenated to form one line of kn processors. Thus, processor $p_{i,j}$ in C corresponds to the processor p_i in the jth copy of R. We make C a ring by connecting $p_{n,k}$ with $p_{1,1}$ by a link which is blocked. Note that even though every processor in C acts as if it is on a ring, the block on the link $p_{n,k} - p_{1,1}$ makes the global behavior of C to be that of a line of processors.

Let ω^k be the input string to C, where $p_{i,j}$ receives the input letter ω_i, and consider the execution of AL on C in which all links are synchronized except for the block on the link $p_{n,k} - p_{1,1}$. For $0 \leq s \leq t$, the histories $h_{i,j}(s)$ and $H_{i,j}$ of the processor $p_{i,j}$

[4]This does not hold for arbitrary executions. In this paper we only need to consider histories of specific executions.

in C (with input string ω^k) are defined similarly to the histories of the processors in R (with input ω) given above. Recall that all processors of R terminate at time $t-1$ or before. Using an argument similar to the "shifting scenario" argument of [FLM85], we show that p_n and $p_{n,k}$ act alike.

LEMMA 3. *Processor $p_{n,k}$ in C accepts.*

Proof. Recall that C consists of k identical copies of R. Assume for a moment that there is no block on the link $p_{n,k}-p_{1,1}$. It is easy to see that in that case $h_{i,j}(s)=h_i(s)$ for all i,j and for $0 \leq s \leq t$. If we now restore the block on $p_{n,k}-p_{1,1}$, then by time s, the block can only effect the s leftmost processors. Thus at time $t-1$ processor $p_{n,k}$ has exactly the same history that p_n has at time $t-1$ and $p_{n,k}$ accepts because p_n does so. □

Next, we define a subsequence $\bar{C}$ of C such that all of its processors have distinct histories at the end of the execution. First we use C to construct a directed graph, G, and the construct $\bar{C}$ from G.

The vertices of G are the processors of C, and there is a directed edge from p to q if q is the rightmost processor having the same history as the processor to the right of p. It is easy to see that there is exactly one edge leaving every processor except the last processor, $p_{n,k}$, and that G contains no cycles. Thus, G is a directed tree rooted at $p_{n,k}$. $\bar{C}$ is now the sequence of processors on the unique path that starts at $p_{1,1}$ and ends at $p_{n,k}$.

LEMMA 4. *No two processors of $\bar{C}$ have the same history in the execution on C.*

Proof. The first processor in $\bar{C}, p_{1,1}$, is the only processor that receives no messages during the execution described above, due to the block on the link entering it. For the other processors the lemma follows from the fact that for each history H there is at most one rightmost processor p in C with $H_p = H$. □

Let the sequence of processors $\bar{C}$ defined above be $\bar{C} = (p_{i_1,j_1},\ldots,p_{i_m,j_m})$, and let τ be the input string $\omega_{i_1}\cdots\omega_{i_m}$ of length m. We now run AL on $\bar{C}$ with all links synchronized except for the link $p_{i_m,j_m}-p_{i_1,j_1}(=p_{n,k}-p_{1,1})$ which is blocked.

LEMMA 5. *In the execution of AL on $\bar{C}$ with input string τ, the history of processor $p_{i_l,j_l}(1 \leq l \leq m)$ of $\bar{C}$ is the same as the history of p_{i_l,j_l} in the corresponding execution of AL on C with input string ω^k. In particular, processor $p_{i_m,j_m}(=p_{n,k})$ of $\bar{C}$ accepts.*

Proof. This follows by a "cut and paste" argument, using the way $\bar{C}$ is constructed from C : $p_{1,1}(=p_{i_1,j_1})$ clearly has the same history in both executions, and the history of any other processor p in both executions is completely determined by its input letter and the history of the preceding processor. □

COROLLARY 1. *For any $1 \leq l \leq m$, the number of bits received by l distinct processors of $\bar{C}$ in the execution described above is at least $(l/4)\log_3(l/2)$.*

Proof. Let P be a set of l processors in $\bar{C}$, and let $p \in P$. By Lemma 5, the history of p in the execution of $\bar{C}$ is the same as its history in the execution of C. This implies, by Lemma 4, that no two processors in P have the same history. Thus, by Lemma 2, the sum of the lengths of the histories of the processors in P is at least $(l/2)\log_3(l/2)$. The corollary follows from the observation made earlier, that this sum is less than twice the number of bits received by these processors. □

Proof of Theorem 1. Let τ' be the first n letters of $\tau \cdot 0^n$. Let $m' = \min(\{m,n\})$ (recall that $|\tau| = m$). Consider an execution of AL on R with input string τ', in which the first m' processors have exactly the same history as the first m' processors in the execution of $\bar{C}$ on τ described above, and no message sent by the remaining processors is ever received. We distinguish two cases, depending on the length m of $\bar{C}$.

Case $m \leq n - \log n$. By Lemma 5, τ' will be accepted by at least one processor in R. Since τ' ends with $\log n$ zeros, Lemma 1 guarantees that $\Omega(n \log n)$ messages are required for the input string 0^n and the theorem holds.

Case $m > n - \log n$. By Corollary 1, the total number of bits received by the first m' processors is $(m'/4)\log_3(m'/2)$, which is $\Omega(n \log n)$. This completes the proof of Theorem 1. □

4. The lower bound for bidirectional rings. The proof of the gap theorem for bidirectional rings follows the same general outline. However, the corresponding "cut and paste" construction is more subtle, and requires a more careful analysis.

THEOREM 1′. *The bit complexity of a bidirectional ring of n anonymous processors is $\Omega(n \log n)$. This bound holds even if the ring is oriented.*

Proof. Let AL be an algorithm for an oriented bidirectional ring R of size n that accepts ω and rejects 0^n and consider a synchronized execution of AL on ω. The history of a processor p_i at time s in such an execution is a string $h_i(s) = d_i(1)m_i(1) \cdots d_i(r_s)m_i(r_s)$, where $d_i(j)$ is either R (for right) or L (for left), and the $m_i(j)$'s are the distinct messages (nonempty bit strings) received by p_i up to (and including) time s, in chronological order; $m_i(j)$ is received from direction $d_i(j)$, and when two messages arrive at the same time, we assume that the left one is received before the right one. Note that the length of H_i is at most two times larger than the number of bits received by p_i.

Assume that AL accepts ω in less than t time units, where $t = nk$. For each positive integer $b \leq k$, define a line D_b of $2nb$ processors as follows: Let C_b be a line $(p_{1,1}, p_{2,1}, \ldots, p_{n,b})$ of nb processors, and let C_b' be the line obtained by replacing each $p_{i,j}$ in C_b by $p_{i,j}'$. D_b is constructed by connecting the last processor $p_{n,b}$ of C_b with the first processor $p_{1,1}'$ of C_b'. As before make D_b a ring by connecting $p_{n,b}'$ to $p_{1,1}$ via a blocked link.

For each $b \leq k$, we construct a particular execution, E_b, of AL on D_b with input string ω^{2b}. Again internal computation of a processor takes no time, and a message requires exactly one unit to traverse a link. A processor is *blocked* at time s if it receives no messages at time s or later. In execution E_b we impose the following schedule: $p_{1,1}$ and $p_{n,b}'$ are blocked at time 1, $p_{2,1}$ and $p_{n-1,b}'$ are blocked at time 2, and in general at time s $(1 \leq s \leq bn)$, the s leftmost and the s rightmost processors of D_b are blocked. Since $p_{1,1}$ and $p_{n,b}'$ are blocked at time 1 and since the execution is synchronized, no message sent on the link $p_{n,b}' - p_{1,1}$ is ever received and thus D_b acts as a line of processors.

The following stronger version of Lemma 3 states that at the end of the execution E_b, the history of the sth leftmost [rightmost] processor in D_b is equal to that of the corresponding processor in the synchronized execution of AL on R after $s - 1$ time units. □

LEMMA 6. *Let $p_{i,j}$ $[p_{i,j}']$ be the sth leftmost [rightmost] processor in $D_b (1 \leq s \leq bn)$, and let $h_{i,j}$ $[h_{i,j}']$ denote the history of processor $p_{i,j}$ $[p_{i,j}']$ in execution E_b. Then $h_{i,j}(t) = h_i(s-1)$ $[h_{i,j}'(t) = h_i(s-1)]$ and in the execution E_k on D_k, both $p_{n,k}$ and $p_{1,1}'$ accept.*

Since every n consecutive processors in D_b correspond to the n processors of R, Lemma 6 implies the following.

COROLLARY 2. *Let R' be any set of n consecutive processors in D_b. Then the sum of the lengths of the histories in E_b is not larger than the sum of the lengths of the histories of the processors in the synchronized execution on R.*

Similar to the proof for the unidirectional case, we associate with the sequence D_b described above an undirected graph as follows: The vertices are the processors in

$D_b = C_b \cdot C'_b$. There is an edge between each processor p in C_b and the rightmost processor in C_b which has the same history as the right neighbor of p, and there is an edge between each processor p' in C'_b and the leftmost processor in C'_b, which has the same history as the left neighbor of p'. We also add an extra edge between the leftmost processor in $C'_b, p'_{1,1}$, and the rightmost processor of $C_b, p_{n,b}$. It is easily observed that this graph is an undirected tree. Let $\overline{C_b}$ be the path in that tree from $p_{1,1}$ to $p_{n,b}$, and $\overline{C'_b}$ be the path from $p_{n,b}$ to $p'_{n,b}$. $\overline{D_b}$ is the concatenation $\overline{C_b} \cdot \overline{C'_b}$.

LEMMA 7. *There is an execution $\overline{E_b}$ on $\overline{D_b}$ in which every processor has the same history as it has in the execution E_b on D_b.*

Proof. Consider two processors, p and q, that are adjacent on $\overline{D_b}$ but not on D_b. Now let $\hat{D}$ be the line obtained from D_b by connecting p and q with a link e, and deleting all the processors between p and q. We now define an execution $\hat{E}$ and $\hat{D}$ in which the histories of all the processors in $\hat{D}$ are the same as the corresponding histories in the execution E_b on D_b.

Assume without loss of generality that in E_b, p sends a message to its right neighbor before it receives a message from it. Then the execution $\hat{E}$ is initiated by simulating the execution E_b on the processor p and the processors to its left, and delaying the computations in all other processors. Such a simulation is possible since no processor to the left of p ever receives a message from a processor to the right of p (here we use the fact that the link between $p'_{n,b}$ and $p_{1,1}$ is blocked). Continue the simulation as long as p sends messages to its right neighbor without receiving a message from it, up to a point where p receives message from its right neighbor. At this point, continue the execution $\hat{E}$ by making a similar simulation on q and the processors to its right. This procedure can be repeated until the desired execution $\hat{E}$ is constructed.

Once the execution $\hat{E}$ is constructed, select in $\hat{D}$ two processors that are adjacent in $\overline{D_b}$ but not in $\hat{D}$, connect them by a link as above, and repeat the procedure above. By repeating this for every link in $\overline{D_b}$, we get the desired execution $\overline{E_b}$. □

The construction of $\overline{C_b}$ and $\overline{C'_b}$ guarantees that no two processors in $\overline{C_b}$ and no two processors in $\overline{C'_b}$ have the same history in the execution $\overline{E_b}$ on $\overline{D_b}$ described above. Hence no three processors in $\overline{D_b}$ have the same history in $\overline{E_b}$. This implies, by Lemma 2, that the number of bits received by any l distinct processors in $\overline{D_b}$ is larger than $(l/8)\log_4(l/4)$.

Let m_k be the length of $\overline{D_k}$ (recall that k is defined by the equality $t = nk$). In the case where $m_k \leq n$ we can pad $\overline{D_k}$ with $n - m_k$ processors that receive input zero. As in the proof Theorem 1, the messages sent by these $n - m_k$ never reach their target. According to Lemma 6, $p_{n,k}$ accepts, and hence by Lemma 7 the corresponding processor in $\overline{D_k}$ accepts too. By distinguishing two cases depending on whether $m_k \leq n - \log n$ or not, we can complete the proof of Theorem 1′ as in Theorem 1, provided that $m_k \leq n$.

However, if $m_k > n$, then we cannot proceed as in the unidirectional case. If we "cut" the n leftmost processors of $\overline{D_k}$, then the nth processor does not receive the proper messages from the right, and the proof requires modification. This is where we use the histories $\overline{E_b}$ for $b < k$.

LEMMA 8. *Let m_b be the length of $\overline{D_b}$, and let $m_0 = 0$. For any b such that $1 \leq b \leq k$, either the last n processors of C_b or the first n processors of C'_b contain at least $(m_b - m_{b-1})/2$ distinct histories in E_b.*

Proof. For notational convenience assume that processor lines with subscript 0 are empty. For a line Q let $|Q|$ denote the number of processors in it. Recall that D_b is obtained by inserting a line of $2n$ processors between the left half, C_{b-1}, and the right half, C'_{b-1}, of D_{b-1}. Since $m_b = |\overline{D_b}| = |\overline{C_b}\,\overline{C'_b}|$ and $m_{b-1} = |\overline{D_{b-1}}| = |\overline{C_{b-1}}\,\overline{C'_{b-1}}|$, either

$$|\overline{C_b}| - |\overline{C_{b-1}}| \geq \frac{m_b - m_{b-1}}{2}$$

or

$$|\overline{C'_b}| - |\overline{C'_{b-1}}| \geq \frac{m_b - m_{b-1}}{2}.$$

Assume without loss of generality that the former is the case. It suffices to show that the last $(m_b - m_{b-1})/2$ processors of $\overline{C_b}$ are the last n processor of C_b. Observe that $\overline{C_b}$ is constructed by taking a prefix of $\overline{C_{b-1}}$ and appending it with a sequence of distinct processors from the last n processors of C_b. Thus the processors at position $|\overline{C_{b-1}}| + 1$ through position $|\overline{C_b}|$ of the line $\overline{C_b}$ must be from the last n processors of C_b. □

We are now ready to prove the Theorem 1′ for the remaining case that $m_k > n$ (recall that $t = kn$). Let b be the smallest integer such that $m_b > n$. Clearly either $m_b - m_{b-1} \geq n/2$ or $n/2 < m_{b-1} \leq n$.

In the former case by Lemma 8 there are n consecutive processors in D_b, which have at least $n/4$ distinct histories the lower bound follows from Lemma 2 and Corollary 2. In the second case, let τ be the string composed of the concatenation of the input letters to the processors in $\overline{D_{b-1}}$. Then $n/2 < m_{b-1} = |\tau| \leq n$. Consider the execution on a line with input string $\tau \cdot 0^{n-|\tau|}$ consisting of the execution $\overline{E_{b-1}}$ on the first m_{b-1} processors, and assume that the remaining processors are never awakened. Since this execution has more than $n/2$ distinct histories, the result follows. This completes the proof of Theorem 1′. □

5. Lower bounds for rings with identifiers.

5. Lower bounds for rings with identifiers. Interestingly, the gap remains for rings with identifiers if the set of possible identifiers is large enough relative to the ring size n. Now the algorithm AL that each processor runs is given two inputs, a symbol of the input alphabet and an *identifier* of the processor. Different processors must receive distinct identifiers. An *execution* of an algorithm includes a labeling of the ring with distinct identifiers in addition to an assignment of an input symbol to each processor, an orientation of the ring, and a combined schedule of how the algorithm is run at each processor. The function value may only depend on the labeling of the processors with input symbols and must be invariant over all labelings of the ring with distinct identifiers, all orientations, and all possible schedules. The *bit* (*message*) *complexity* of an algorithm is again the maximal number of bits (messages) sent over all possible executions of this algorithm and the bit (message) complexity of a function f is the minimal possible bit (message) complexity of any algorithm that computes f. Note that this notion of complexity is worst case over all possible labelings of the ring with distinct identifiers.

To prove the lower bound of $\Omega(n \log n)$ on the bit complexity for rings with identifiers we make the following simplifying assumptions. Assume that AL rejects 0^n and accepts an input string ω, as before. Let $\sum$ denote the set of letters in the string $0 \cdot \omega$. Observe that the function computed by AL is nonconstant even for input strings over the restricted input alphabet $\sum^n$. For each $a \in \sum$ let $e(a)$ be an encoding of a by at most $\log n + 2$ bits. For a specific execution of AL, in which all the input letters are from $\sum$, define $eh_i(s)$, the *extended history* of the processor p_i at time s in this execution, to be a string $e(a)t_i(1)d_i(1)m_i(1) \cdots t_i(r_s)d_i(r_s)m_i(r_s)$, where a is the input of p_i in this execution, the $d_i(j)s$ and the $m_i(j)s$ are, as before, directions and messages, and the $t_i(j)s$ are either I if the corresponding message was received by p_i, or O if it was sent by it.

THEOREM 2. *The bit complexity of a bidirectional oriented ring of n processors with distinct identifiers is $\Omega(n \log n)$, provided that the set of identifiers that can be assigned to the processors is large enough.*

Proof. Assume that there is an algorithm AL of bit complexity[5] $o(n \log n)$ computing a nonconstant function, and let Z be the set of possible identifiers, with $|Z| \geq n2^{n^n}$. Let $\boldsymbol{H}_z$ be the set of possible extended histories that can occur at a processor with identifier $z \in Z$ when AL is executed on an input from $\sum^n$. Then the definition of extended history implies that the program of a processor with identifier z in executions on input from $\sum^n$ is uniquely determined by $\boldsymbol{H}_z$, and hence that if $\boldsymbol{H}_z = \boldsymbol{H}_{z'}$, then two processors with identifiers z and z' behave identically in all executions on inputs from $\sum^n$. Since, apart from the $\log n + 2$ bits required to encode the input, the length of the extended history of a processor is at most three times the number of bits received or sent by that processor, the assumption on the bit complexity of AL implies that the set of all possible extended histories $\boldsymbol{H} = \cup_{z \in Z} \boldsymbol{H}_z$ is of cardinality $o(2^{n \log n}) = o(n^n)$.

Denote $z \equiv_{AL} z'$ if $\boldsymbol{H}_z = \boldsymbol{H}_{z'}$. The relation $\equiv_{AL}$ partitions Z into at most $2^{|\boldsymbol{H}|} = o(2^{n^n}) = o(|Z|/n)$ equivalence classes, where all the processors in each class behave identically on inputs from $\sum^n$. Thus, there must be n distinct identifiers in Z that belong to the same equivalence class. By restricting the identifiers to this equivalence class, the proof of the anonymous class implies a lower bound of $\Omega(n \log n)$ on the bit complexity of AL, which contradicts the assumption. □

6. Functions computable with a small number of messages. In [ASW88] a nonconstant function was presented that is computable in $O(n)$ messages and $O(n \log n)$ bits. That function is the characteristic function for the subset of $\{0, 1\}$, which consists of all cyclic shifts pattern $0(01)^*$. The pattern requires that the ring size n is odd. As we will show here, for any nondivisor k of n there are similar patterns that can be recognized in $O(kn)$ messages and $O(kn + n \log n)$ bits. Since for any ring size n there is a nondivisor of size $O(\log n)$ this leads to a nonconstant function defined for all ring sizes with message and bit complexity $O(n \log n)$. This shows that the $\Omega(n \log n)$ bounds on the bit complexity given in the previous sections are tight.

As the main result of this section we exhibit a function with "almost linear," i.e., $\Omega(n \log^* n)$, message complexity, and this function is defined for arbitrary ring size. The lower bound proven in [DG87] for specific ring sizes matches this upper bound. The upper bounds proven in this section assume alphabets of constant size at least two. In fact, we show that if the alphabet size is at least n, then there are nonconstant functions that can be computed with $O(n)$ messages. (The $\Omega(n \log^* n)$ lower bound of [DG87] assumes constant alphabet size.)

All algorithms presented in this section work on the unidirectional ring and output one if and only if the input is a cyclic shift of a string Θ. It is easy to derive bidirectional versions of these algorithms working on bidirectional rings of arbitrary orientation such that the message complexity increases by at most a factor of two. Recall that algorithms working on unidirectional rings assume that the ring is oriented and they only receive messages from the left and send messages to the right. Let U be such an algorithm. In the bidirectional version each processor runs two noninteracting versions of U, one that receives messages from the left neighbor and sends messages to the right neighbor, and a second one that receives messages from the right neighbor and sends to the left neighbor. Here left and right refer to the local orientation of the processor. The local orientations between processors do not have to be consistent with each other. After both algorithms terminate each processor outputs one if and only if one of the algorithms outputs one. It is easy to see that the bidirectional version computes one if and only if the input string or the reverse of the input string is a cyclic shift of Θ.

[5]We write $f(n) = o(g(n))$ if $\lim_{n \to \infty} f(n)/g(n) = 0$.

To get an easy introduction to the unidirectional algorithms presented in this section, we will first present a simple generalization of an algorithm of [ASW88], which is also used in our algorithm. This generalization, which we call *NON-DIV*, is defined for two parameters: the ring size n, and an integer k which does not divide $n (n > k)$. It accepts the (cyclic shifts of) the pattern $\pi = 0^{n \bmod k}(0^{k-1}1)^{\lfloor n/k \rfloor}$. The message complexity of *NON-DIV* is $O(kn)$ and its bit complexity is $O(kn + n \log n)$. Algorithm *BRUIJN*, the second algorithm described, follows the same outline as *NON-DIV* but it is used to recognize more complicated patterns. Finally, *BRUIJN* is used iteratively in Algorithm *STAR* for recognizing a pattern in $O(n \log^* n)$ messages.

```
ALGORITHM NON-DIV(k, n):
%% it is assumed that $n \bmod k = r \neq 0$. %%
FOR all $n$ processors in parallel DO
N1. Set your status to passive and send your bit to the right neighbor. Forward $k+r-3$
    bits received from the left to the right and wait until you receive $k + r - 2$ bits
    from the left.
N2. Let $\Psi$ be the concatenation of the $k + r - 2$ bits received from the left with your
    own bit ($\Psi$ contains $k + r - 1$ bits).
    IF $\Psi$ is not a cyclic substring of $\pi = 0^r(0^{k-1}1)^{\lfloor n/k \rfloor}$ THEN send a zero-message
    and terminate with output zero.
    IF $\Psi = o^{k+r-1}$ THEN send a size-counter message with count one to the right
    and change your status to active.
    Wait for a message from the left.
    %% This must be either a zero-message or a size-counter. %%
N3. IF the message receives is a zero-message THEN forward it and terminate with
    output zero.
    IF the message received is a size-counter THEN
        IF your status is passive THEN increment the size-counter by one and forward
        it to the right.
        ELSE (i.e., your status is active):
        IF the value of the size-counter is $n$ THEN send a one-message to the right and
        terminate with output one.
        ELSE send a zero-message to the right and terminate with output zero.
    ELSE (The message received must be either one-message or a zero-message)
N4. Forward the message you received and terminate with output zero if it was a zero-
    message and with output one otherwise.
```

We sketch below a proof that *NON-DIV* indeed recognizes the string $\pi = 0^r(0^{k-1}1)^{\lfloor n/k \rfloor}$, where $r = n \bmod k$.

Case 1. The input string contains a substring of length $k + r - 1$ which is not a substring of π. Then the processor with the rightmost bit of this substring will initiate a *zero-message* in Step N2, and will never forward a *size-counter*. This latter fact means that no *size-counter* will make a complete traversal of the ring, and hence that no *one-message* can be initialized. The *zero-messages* (possibly more than one) produced in Step N2 will eventually cover the whole ring and all processors will output zero.

Case 2. This case is the complement of Case 1. Thus no *zero-message* will be initiated in Step N2. A simple case analysis shows that in this case the input string must contain a substring of $k + r - 1$ successive zeros, and that it contains exactly one such substring if and only if the input string is a cyclic shift of π. This means that at least one *size-counter* will be initiated in Step N2, and exactly one *size-counter* will be initiated if and only if the input string should be accepted. The proof is completed by observing that output one is obtained only if some *size-counter* traverses the whole ring which is only possible if exactly one such counter was initiated. If more than one *size-counter* is initiated, then eventually some processor will generate a *zero-message* and all processor will output zero.

Each processor sends at most $2k$ messages in an execution of *NON-DIV*, so that the total number of messages is $O(kn)$. Counters cost at most $\lfloor \log n \rfloor + 1$ bits, so the total bit complexity of *NON-DIV* is $O(kn + n \log n)$.

Now it is easy to derive an algorithm computing a nonconstant function of $O(n \log n)$ bit complexity uniformly for all ring sizes: First each processor determines the smallest nondivisor of k of the ring size n and then runs *NON-DIV*(k, n). Since k is $O(\log n)$ we get an algorithm for a nonconstant function whose bit complexity matches the lower bounds of the previous sections.

LEMMA 9. *There is a nonconstant function for binary input alphabet defined for all ring sizes n with bit complexity $O(n \log n)$ on the unidirectional ring.*

Before proceeding we note that if the input alphabet has size at least n, then there are nonconstant functions of $O(n)$ message complexity: let $\sigma_0, \ldots, \sigma_{n-1}$ be n letters of the alphabet. The cyclic shifts of $\sigma = \sigma_0\sigma_1 \cdots \sigma_{n-1}$ can be accepted as follows. Send your input letter to your right neighbor as in Step N1 of *NON-DIV*, and concatenate the letter received with your own letter as in N2 to form Ψ; if Ψ is not of the form $\sigma_i\sigma_{i+1 \bmod n}(i = 0, \ldots, n-1)$, then initiate a *zero-message* and terminate with output 0. The pattern $\Psi = \sigma_{n-1}\sigma_0$ initiates a counter message. Thus we have:

LEMMA 10 (Hans Bodlaender). *If the input alphabet is of size at least n, then the distributed message complexity of ring n processors is $O(n)$.*

The above lemma can be generalized to alphabet size ϵn for arbitrary positive constant ϵ. Moreover, a similar technique can be used to show a linear distributed message complexity of other networks, provided the alphabet size is linear in the network size.

The final goal of this section is to present an algorithm called *STAR*(n) that computes a nontrivial function in $O(n \log^* n)$ messages for arbitrary ring size n. So far we have shown that if n is not divisible by some integer $k = O(\log^* n)$, then *NON-DIV*(k, n) is such an algorithm. However, it is much harder to find algorithms of low message complexity if n has no small nondivisors.

STAR(n) recognizes patterns based on the classical de Bruijn Sequences [B46]. A *de Bruijn Sequence* β_k is a sequence of 2^k bits with the property that each binary string of length k occurs in β_k exactly once as a cyclic substring [B46]. For each $k \geq 1$ there are such sequences (see, e.g., [E79]). From now on assume that β_k denotes a fixed such sequence with the property that its first k bits are zeros, and that the first zero is barred. That means that our input alphabet has three letters $0, 1$, and $\bar{0}$, which we call bits for simplicity. We discuss later how to get the corresponding results for the two-letter case.

One way to construct a de Bruijn sequence β_k is the following: start with $\bar{0}0^{k-1}$; bit i for $(k + 1 \leq i \leq 2^k)$ is one if the string of length k formed by the bits $i - k + 1, i - k + 2, \ldots, i - 1$ appended by a one does not appear yet in the sequence; otherwise bit i is zero. The sequences for $k = 1, 2, 3, 4$ are $\bar{0}1, \bar{0}011, \bar{0}0011101, \bar{0}000111101100101$, respectively.

We will use prefixes of strings in $(\beta_k)^*$ to construct patterns recognizable with a small number of messages. Note that in these prefixes, each new copy of β_k starts with $\bar{0}$. Let $\pi_{k,n}$ (for $k \leq n$) be the first n bits of $(\beta_k)^n$. For example, $\pi_{3,21} = \bar{0}0011101\bar{0}0011101\bar{0}0011$. The Algorithm *STAR* (n) will essentially recognize cyclical shifts of a word formed by interleaving a number of patterns of the form $\pi_{k',n'}$, for various choices of k' and n'.

At times the algorithm checks whether the parts of the input string fulfill a local condition of legality with respect to some $\pi_{k',n'}$, which we define now. Let $\Theta = \Theta_0 \cdots \Theta_{n-1}$ be a cyclic string of length n representing the input of the n processors. Then Θ_i is *legal* with respect to $\pi_{k,n}$ if the k bits to the left of Θ_i appended with Θ_i produces a string that occurs as a cyclic substring in $\pi_{k,n}$. The following lemma shows that if all bits of Θ are legal with respect to $\pi_{k,n}$, then this string must be equal to $\pi_{k,n}$ or a close relative thereof.

Let τ be any bit string and σ be a cyclical substring in τ. A bit b is called a *successor* of σ in τ if σb is a cyclical substring of τ. Let the last k bits of a string $\alpha_0, \alpha_1, \ldots, \alpha_{l-1}$ of length $l \geq k$ be the string $\alpha_{l-k}, \alpha_{l-k+1}, \ldots, \alpha_{l-1}$. Let ρ be the last k bits of $\pi_{k,n}$. Every length k cyclic substring $\sigma \neq \rho$ of $\pi_{k,n}$ has exactly one successor in $\pi_{k,n}$. However there might be two successors of ρ in $\pi_{k,n}$. Clearly $\bar{0}$ is always a successor. If $n > 2^k$ then $\pi_{k,n}$ contains β_k as substring and ρ has b as a successor in $\pi_{k,n}$, where b is the unique successor of ρ in β_k. Now $b \neq 0$ if and only if and $n \neq 0 \bmod 2^k$.

LEMMA 11. *Assume all n bits of an input string Θ are legal with respect to $\pi_{k,n}$ and let ρ consist of the last k bits of $\pi_{k,n}$. If $n = 0$ mod 2^k then Θ is a cyclic shift of $(\beta_k)^{n/2^k}$. If $n \neq 0$ mod 2^k then Θ contains $\rho\bar{0}$ at least once as a cyclical substring and $\rho\bar{0}$ occurs exactly once if and only if Θ is a cyclical shift of $\pi_{k,n}$.*

Proof. If $n = 0 \bmod 2^k$, then $\pi_{k,n} = (\beta_k)^{n/2^k}$. Thus, $\pi_{k,n}$ and β_k have the same set of cyclical substrings of length k and each such substring has the same unique successor in $\pi_{k,n}$ and in β_k. Since all bits of Θ are legal with respect to $\pi_{k,n}$ we conclude that if $n = 0 \bmod 2^k$, then Θ must be a cyclic shift of $(\beta_k)^{n/2^k}$.

If $n \neq 0 \bmod 2^k$, then by the legality assumption the successors of all cyclical substrings of length k of Θ are uniquely determined, except for ρ, which has two successors: $\bar{0}$ and the successors of ρ in β_k. Thus after each occurrence of ρ the current copy of β_k is completed or it is cut off at ρ and a new copy of β_k is begun with the $\bar{0}$. The definition of ρ implies that the subword of β_k beginning with $\bar{0}$ and ending with ρ has length $n \bmod 2^k$ and equals $\pi_{k,n\mathrm{mod}2^k}$. Thus Θ must be a cyclical shift of a string of length n of the form $(\beta_k + \pi_{k,n\mathrm{mod}2^k})^*$. Now the second half for the claim follows easily. □

If the ring size n is not divisible by $k = 1 + \log^* n$ then *STAR*(n) simply calls *NON-DIV*(k, n). However if $n = 0 \bmod (1 + \log^* n)$ then the algorithm recognizes a complicated pattern $\Theta^{(n)}$ which we will now describe. Let $n' = n/(1 + \log^* n)$. $\Theta^{(n)}$ is a string in the language $F = (\#\{\bar{0}, 0, 1\}^{\log^* n})^{n'}$ over the four letter alphabet $\{0, 1, \bar{0}, \#\}$ containing $l(n)$ "interleaved" de Bruijn Sequences, where $l(n)$ is defined as follows: let $k_0 = 1$ and $k_{i+1} = 2^{k_i}$; $l(n)$ is the minimum i such that k_i does not divide n'. Note that $\log^* n$ is the minimum i such that $k_i \geq n$, and hence $l(n) \leq \log^* n$.

For all strings Θ in F, let $\Theta[i]$ (for $1 \leq i \leq \log^* n$) be the concatenation of the n' bits which are the ith letters to the right of the $\#$ letters. For example, $\Theta[1]$ consists of the bits to the right of the $\#$ letters, and $\Theta[\log^* n]$ consists of the bits to the left of the $\#$ letters. $\Theta^{(n)}$ is the string in F with the following properties:

(1) $\Theta^{(n)}[i] = \pi_{k_{i-1},n'}$, for $1 \leq i \leq l(n)$; and

(2) $\Theta^{(n)}[i]$ contains only zeros for $l(n) + 1 \leq i \leq \log^* n$.

As a rough outline, Algorithm *STAR* (n) first checks in Step S0 whether the input Θ is in F. Note that if this is true, then there are exactly n' processor with input $\#$. These processors check in loop i of Step S1 whether $\Theta[i]$ is a cyclic shift of $\pi_{k_{i-1},n'}$. To do this for $\Theta[l(n)]$ Step S2 is needed as well.

```
ALGORITHM $STAR(n)$.
S0. IF $n \neq 0 \bmod (\log^* n + 1)$ THEN call $NON\text{-}DIV(k,n)$, for $k = \log^* n + 1$,
    to recognize cyclic shifts of $0^{n \bmod k}(0^{k-1}1)^{n/k}$.
    ELSE
      BEGIN
      Send your input to your right neighbor, forward $log^*n$ inputs, and wait for $1+$
    $\log^* n$ inputs.
      IF the number of # signs you have received is not exactly one THEN send a
    zero-message to the right and terminate with output zero.
      %% If no processor initiates a zero-message in the previous statement then
    there are $n'$# signs in the input, all of which are exactly $1 + \log^* n$ apart. Each of
    processor with input # denotes the $\log^* n$ bit string between itself and the previ-
    ous processor with input # as $b_1 \cdots b_{\log^* n}$.%%
      IF your input is # and $b_{l(n)+1}b_{l(n)+2} \cdots b_{\log^* n}$ contains a letter other than 0 or
    if $b_i = \bar{0}$, for $1 < i \leq l(n)$, and $b_{i-1} \neq \bar{0}$ THEN send a zero-message to the right
    and terminate with output zero.
      END
    %% For the remaining part of the algorithm whenever any processor receives
    a zero-message (one-message) it forwards it and terminates with output zero (re-
    spectively, one). Furthermore we only address processors with input #. All other
    processors always only forward the messages they receive and terminate as de-
    scribed above. %%

S1. For $i := 1$ to $l(n)$ DO
    BEGIN
    IF $i = 1$ THEN all processors are initiators.
    ELSE you are an initiator if your bit $b_{i-1}$ equals $\bar{0}$.
    %% The initiators are $k_{i-1}(1 + \log^* n)$ apart.%%
    IF you are an initiator
    THEN start an input collection message by sending your bit $b_i$ to the right.
    IF you are an initiator THEN do the following once and ELSE to it twice:
        Append your bit $b_i$ to the input collection message received and forward it.
    %% The second input collection message received by each initiator processor has
    length $2k_{i-1}$. The concatenation of the second halves of all these messages con-
    stitute $\Theta^{(n)}[i]$ and the same holds for the first halves.%%
    IF you are an initiator and any of the bits in the second half of the last message is
    not legal with respect to $\pi_{k_{i-1},n'}$ THEN initiate a zero-message.
      %% Note that $2^{k_{i-1}} = k_i$ divides $n'$ for all $i$ except $i = l(n)$. %%
      END

S2.   IF your last message contains $\rho\bar{0}$ as a substring, where $\rho$ consists of the last
      $k_{l(n)-1}$ bits of $\pi_{k_{l(n)-1},n'}$
      THEN send a size-counter with count one to the right.
      IF you receive a size-counter and did not start a size-counter
      THEN increment it and forward it.
      ELSE IF the received size-counter equals $n'$
      THEN send a one-message to the right.
      ELSE Send a zero-message.
END
```

LEMMA 12. *For all ring sizes $n = 0 \bmod (\log^* n + 1)$, Algorithm STAR$(n)$ recognizes cyclic shifts of $\Theta^{(n)}$ in $O(n \log^* n)$ messages.*

Proof.

Case 1. A *zero-message* was generated in Step S0. That is either (i) the input string Θ is not in $F = (\#\{\bar{0}, 0, 1\}^{\log^* n})^{n'}$, or (ii) for some processor with input $\#$ one of the bits $b_{l(n)+1}, \ldots, b_{\log^* n}$ is not 0, or (iii) for some such processor $b_i = \bar{0}$ and $b_{i-1} \neq \bar{0}$ for some $1 \leq i \leq l(n)$. In any of the above three cases the input string is not a cyclical shift of $\Theta^{(n)}$ and the processor which generated a *zero-message* will not forward any *size-counter*. Hence no *size-counter* will traverse the whole ring, and no processor will initiate a *one-message*. We conclude that in this case all the processors will eventually output zero.

Case 2. Case 1 does not hold and for some $1 \leq i \leq l(n)$, $\Theta[i]$ contains a bit which is not legal with respect to $\pi_{k_{i-1},n'}$. Assume i is minimum. We first claim that all initiators of loop i are $k_{i-1}(1+\log^* n)$ apart. This clearly holds for $i = 1$. If $i > 1$ then in loop $i-1$ all bits of $\Theta[i-1]$ were legal with respect to $\pi_{k_{i-2},n'}$. Recall that $n' = 0 \bmod k_{i-1}$ and that $2^{k_{i-2}} = k_{i-1}$. Thus Lemma 11 implies that $\Theta[i-1]$ is a cyclic shift of $(\beta_{k_{i-2}})^{n/k_{i-1}}$ and all occurrences of $\bar{0}$ in $\Theta[i-1]$ are k_{i-1} apart. This guarantees that the claim holds for $i > 1$ as well.

Since all initiators are properly spaced the second input collection message the initiators receive are exactly $2k_{i-1}$ long. Thus the second halfs of these messages form $\Theta[i]$ and the same holds for the first halves. Each initiator can check whether the bits in the second half are legal with respect to $\pi_{k_{i-1},n'}$ since it knows the preceding substrings of length k_{i-1}. By the above choice of i one of the bits must be illegal and thus some initiator will start a *zero-message*. Thus in Case 2 all processors correctly output zero.

Case 3. Case 1 does not hold and for $1 \leq i \leq l(n)$, all bits of $\Theta[i]$ are legal with respect to $\pi_{k_{i-1},n'}$. Since all bits of $\Theta[l(n)]$ are legal with respect to $\pi_{k_{i-1},n'}$ and $n' \neq 0 \bmod k_{l(n)}$, Lemma 11 guarantees that there is at least one occurrence of $\rho\bar{0}$ in $\Theta[l(n)]$, where ρ consists of the last $k_{l(n)-1}$ bits ρ of $\pi_{k_{l(n)-1},n'}$. Thus there will be at least one initiator that has $\rho\bar{0}$ as a substring in its *second input collector* message and these initiators will start *size-counters*. As for *NON-DIV* if more than one counter is initiated then this eventually causes a *zero-message* and all processors terminate with output zero. Only if exactly one counter travels around the whole ring, a *one-message* is initiated and all processors output one. The correctness of *STAR*(n) in Case 3 follows again from Lemma 11 which assures us that exactly one counter is initiated in Step S3 if and only if $\Theta[l(n)]$ is a cyclic shift of $\pi_{k_{l(n)-1},n'}$.

It is easy to see that Algorithm *STAR* sends $O(n \log^* n)$ messages in Step S0. In Step S1 each loop costs $O(n)$ messages and there are $l(n) = O(\log^* n)$ iterations. In Step S2 only $O(n)$ messages are sent. $\square$

Note that $\Theta^{(n)}$ might use up to four letters. To recognize similar strings $\Theta'^{(n)}$ over a binary input alphabet we encode the ith letter $(1 \leq i \leq 4)$ by $1^i 0^{5-i}$. If $n \neq 0 \bmod 5$ then $\Theta'^{(n)} = 0^{n \bmod 5}(0^4 1)^{n/5}$ and otherwise $\Theta'^{(n)}$ equals $\Theta^{n/5}$, using the five bit encoding for all letters. It is easy to see that for all ring sizes $\Theta'^{(n)}$ can also be recognized in $O(n \log^* n)$ messages.

THEOREM 3. *There is a nonconstant function for binary input alphabet defined for all ring sizes n with message complexity $O(n \log^* n)$ on the unidirectional ring.*

Acknowledgment. The authors thank Hans Bodlaender for pointing out Lemma 10, Gerhard Buntrock for many useful discussions, and an anonymous referee for numerous and insightful comments which helped to clarify and improve the presentation of this paper.

REFERENCES

[AAHK89] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick, *Randomized function evaluation on a ring*, Distributed Computing, 3 (1989), pp. 107–117.

[ASW88] C. Attiya, M. Snir, and M. K. Warmuth, *Computing on an anonymous ring*, J. Assoc. Comput. Mach., 35 (1988), pp. 845–875.

[B46] N. G. de Bruijn, *A Combinatorial Problem*, *Proceedings of Koninklijke Nederlands Akademie van Wetenschappen*, Vol. 49, Part 2, 1946, pp. 758–764.

[BB89] P. W. Beame and H. L. Bodlaender, *Distributed Computing on Transitive Networks*: *The Torus*, Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science, 1989, pp. 294–303.

[BMW91] H. L. Bodlaender, S. Moran, and M. K. Warmuth, *The distributed bit complexity of the ring*: *From the anonymous to the non-anonymous case*, Inform. Comput., to appear.

[DKR82] D. Dolev, M. Klawe, and M. Rodeh, *An $O(n \log n)$ unidirectional algorithm for extrema finding in a circle*, J. Algorithms, 3 (1982), pp. 245–260.

[DG87] P. Duris and Z. Galil, *Two Lower Bounds in Asynchronous Distributed Computation*, Proceedings of IEEE Conference on Foundations in Computer Science, IEEE Press, New York, 1987, pp. 326–330.

[E79] S. Even, *Graph Algorithms*, Computer Science Press, New York, 1979.

[FLM85] M. J. Fischer, N. A. Lunch, and M. Merritt, *Easy impossibility proofs for distributed consensus problems*, Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computation, Minaki, Ontario, Canada, August 1985, pp. 59–70.

[H68] J. Hartmanis, *Computational complexity of one-tape Turing machine computations*, J. Assoc. Comput. Mach., 15 (1968), pp. 325–339.

[MW86] S. Moran and M. K. Warmuth, *Gap theorems for distributed computation*, Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, 1986, pp. 131–140.

[MZ87] Y. Mansour and S. Zaks, *On the bit complexity of distributed computations in a ring with a leader*, Inform. Comput., 75 (1987), pp. 162–177.

[P82] G. L. Peterson, *An $O(n \log n)$ unidirectional algorithm for the circular extrema problem*, ACM Trans. Programming Languages and Systems, 4 (1982), pp. 758–762.

[T64] B. A. Trachtenbrot, *Turing companions with logarithmic delay*, Algebra i Logika, 3 (1964), pp. 33–48 (in Russian): English translation in University of California Computing Center, Tech. Report 5, Berkeley, CA, 1966.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 395–402, April 1993

ON LANGUAGES WITH VERY HIGH SPACE-BOUNDED KOLMOGOROV COMPLEXITY*

RONALD V. BOOK† AND JACK H. LUTZ‡

Abstract. It is shown that if a language recognizable in exponential space is bounded truth-table reducible in polynomial time to a language with very high space-bounded Kolmogorov complexity, then it is bounded truth-table reducible in polynomial time to a sparse language.

There are a number of corollaries, including the following:

(a) no language with very high space-bounded Kolmogorov complexity is $\leq_{btt}^{P}$-hard for NP, unless P = NP;

(b) no language with very high space-bounded Kolmogorov complexity is $\leq_{btt}^{P}$-hard for the class of languages accepted in exponential time.

Key words. Kolmogorov complexity, bounded truth-table reducibility, exponential space, sparse sets

AMS(MOS) subject classifications. 68Q15, 03D15

1. Introduction.

In complexity theory a great deal of attention has been given to sparse languages and to properties of languages efficiently reducible to sparse languages (see, for example, [KL82], [Ma82], [OW91], [Wa87]). Such languages are considered to have very low information content. A number of results in this area have improved our understanding of the structure of complexity classes and resource-bounded reducibilities.

The subject of this paper is the class of languages that are bounded truth-table (or Turing) reducible in polynomial time to languages with essentially maximal information content. Specifically, it is shown (Theorem 3.1) that any language recognizable in exponential space that is bounded reducible in polynomial time to a set with very high space-bounded Kolmogorov complexity must be bounded reducible in polynomial time to a sparse set. Using a recent result of Ogiwara and Watanabe [OW91], this implies that no language with very high space-bounded Kolmogorov complexity can be $\leq_{btt}^{P}$-hard for NP, unless P = NP; similar results are given for other classes.

From the proof of the main result, other conclusions can be made. For example, Watanabe [Wa87] has shown that the class of languages recognizable deterministically in exponential time (denoted E) is not included in the class of languages that are bounded truth-table reducible in polynomial time to sparse languages; we can conclude that no language with very high space-bounded Kolmogorov complexity can be $\leq_{btt}^{P}$-hard for E.

Languages with very high space-bounded Kolmogorov complexity are very complex in the sense that each such language has essentially maximal information content. It is known that almost every language has space-bounded Kolmogorov complexity of this magnitude. The results presented here provide evidence that the information in languages with very high space-bounded Kolmogorov complexity is encoded in such a way that very little of it can be retrieved in a computationally useful way.

*Received by the editors July 22, 1991; accepted for publication (in revised form) January 31, 1992. The results of this paper were announced at the Seventh IEEE Conference on Structure in Complexity Theory, June 1992, and an extended abstract appears in the proceedings of that conference.

†Department of Mathematics, University of California, Santa Barbara, California 93106. The work of this author was supported in part by National Science Foundation grant CCR-8913584 and by the Alexander von Humboldt Stiftung while he visited the Institut für Informatik, Technische Universität München, Germany.

‡Department of Computer Science, Iowa State University, Ames, Iowa 50011. The work of this author was supported in part by National Science Foundation grant CCR-8809238 and in part by the Center for Discrete Mathematics and Theoretical Computer Science, where he was a visitor while a portion of this work was done.

2. Preliminaries. For the most part our notation is standard, following that used by Balcázar, Díaz, and Gabarró [BDG88], [BDG90].

All *languages* here are sets $A \subseteq \{0,1\}^*$. A language S is *sparse*, written $S \in \mathrm{SPARSE}$, if there is a polynomial q such that $\|S_{\leq n}\| \leq q(n)$ for all n, where $S_{\leq n}$ denotes $S \cap \{0,1\}^{\leq n}$. We write REC for the set of all recursive languages.

A $\leq^{\mathrm{P}}_{k-tt}$*-reduction* of language A to language B is a pair (f,g) of polynomial time computable functions such that for each $x \in \{0,1\}^*$, the following hold:

(i) $f(x) = (f_1(x), f_2(x), \ldots, f_k(x))$ is an ordered k-tuple of strings;

(ii) $g(x) : \{0,1\}^k \to \{0,1\}$ is a Boolean function;

(iii) $[\![x \in A]\!] = g(x)([\![f_1(x) \in B]\!][\![f_2(x) \in B]\!] \cdots [\![f_k(x) \in B]\!])$, where $[\![\phi]\!]$ is the truth value of ϕ (i.e., $[\![\phi]\!] =$ **if** ϕ **then** 1 **else** 0).

We write $A \leq^{\mathrm{P}}_{btt} B$ if for some k, $A \leq^{\mathrm{P}}_{k-tt} B$.

Given a polynomial time-bounded reducibility $\leq^{\mathrm{P}}_{\mathrm{r}}$ and a class C of languages, we write $\mathrm{P_r(C)}$ for

$$\{A | (\exists C \in \mathrm{C}) \text{ such that } A \leq^{\mathrm{P}}_{\mathrm{r}} C\}.$$

For example, $\mathrm{P}_{btt}(\mathrm{C}) = \cup_{k>0} \mathrm{P}_{k-tt}(\mathrm{C})$.

We use the following notation for exponential complexity classes:

$$\mathrm{E} = \mathrm{DTIME}(2^{\mathrm{linear}}); \mathrm{ESPACE} = \mathrm{DSPACE}(2^{\mathrm{linear}}).$$

A property of natural numbers holds *infinitely often* (i.o.) if it holds for infinitely many $n \in \mathbb{N}$, and holds *almost everywhere* (a.e.) if it holds for all but finitely many $n \in \mathbb{N}$.

3. Main result. The main technical result of this paper is the following: any language in ESPACE that is bounded truth-table reducible in polynomial time to a set with high space-bounded Kolmogorov complexity must be bounded truth-table reducible in polynomial time to some sparse set. In the next section we show that this result yields a variety of corollaries.

Given a machine M, a function $t : \mathbb{N} \to \mathbb{N}$, a language $L \subseteq \{0,1\}^*$, and a natural number n, the t-*space-bounded Kolmogorov complexity* of $L_{\leq n}$ relative to M is defined as

$$KS^t_M(L_{\leq n}) = \min\{|\pi| \; |M(\pi, n) = \chi_{L_{\leq n}} \text{in } t(2^n) \text{ space}\}.$$

Thus, $KS^t_M(L_{\leq n})$ is the length of the shortest string π such that M on input (π, n) outputs the $(2^{n+1}-1)$-bit characteristic string of $L_{\leq n}$ and halts while using no more than $t(2^n)$ workspace. This quantity is often interpreted as the "amount of information" that is contained in $L_{\leq n}$ and is "accessible" by means of computations that use at most $t(2^n)$ workspace.

Well-known simulation techniques show that there exists a universal machine U that is optimal in the sense that for each machine M there is a constant c such that for all t, L, and n, we have

$$KS^{ct+c}_U(L_{\leq n}) \leq KS^t_M(L_{\leq n}) + c.$$

Hence, we fix an optimal machine U and omit it from the notation. (See [BDG90] or [LV90] for additional discussions of Kolmogorov complexity.)

The main result involves languages with essentially maximal information content, that is, languages B such that for every polynomial q, $KS^q(B_{\leq n}) > 2^{n+1} - 2n$ almost everywhere. (Notice that for every language A, $KS^n(A_{\leq n}) < 2^{n+1}+c$ for some absolute constant c; this justifies the phrase "essentially maximal.")

THEOREM 3.1. *Let $A \in$ ESPACE, and let $k > 0$ be an integer. Suppose that there is a language B such that $A \leq^{\mathrm{P}}_{k-tt} B$ and, for every polynomial q, $KS^q(B_{\leq n}) > 2^{n+1} - 2n$ almost everywhere. Then there exists a sparse set S such that $A \leq^{\mathrm{P}}_{k-tt} S$.*

Before proceeding with the proof of Theorem 3.1, we provide some notation and terminology.

First, suppose that for some languages C and D and some functions f and g, $C \leq^{\mathrm{P}}_{k-tt} D$ via (f, g). Define $\tilde{f}$ as follows: for each string $x \in \{0,1\}^*$ and each $i, 1 \leq i \leq k$, $\tilde{f}_i(x) = f_i(x)10^{|x|}$. Note that for each $x \in \{0,1\}^*$ and $i, 1 \leq i \leq k$,

$$(3.1) \qquad |\tilde{f}_i(x)| > |x|.$$

For each $L \subseteq \{0,1\}^*$, define $\tilde{L}$ by

$$\tilde{L} = \{\tilde{f}_i(x) | x \in \{0,1\}^*, 1 \leq i \leq k, f_i(x) \in L\}.$$

Notice that $C \leq^{\mathrm{P}}_{k-tt} D$ via (f, g) implies that $C \leq^{\mathrm{P}}_{k-tt} \tilde{D}$ via $(\tilde{f}, g)$.

Second, some terminology will be convenient. For $n \in \mathbb{N}$, an *n-assignment* is a partial function α from $\{0,1\}^{\leq n}$ into $\{0,1\}$. (Thus, an n-assignment is also an m-assignment for all $m \geq n$.) A language $C \subseteq \{0,1\}^*$ *satisfies* an n-assignment α if $\alpha(x) = [\![x \in C]\!]$ for all $x \in$ dom α. Two n assignments α and β are *consistent* with one another if there exists a language $C \subseteq \{0,1\}^*$ such that $\tilde{C}$ satisfies both α and β. For an n-assignment α and $x \in \{0,1\}^*$, let $\alpha\tilde{f}(x)$ denote the string $\alpha(\tilde{f}_1(x)) \dots \alpha(\tilde{f}_k(x)) \in \{0,1\}^k$. (Note that $\alpha\tilde{f}(x)$ is defined if and only if $\tilde{f}_1(x), \dots, \tilde{f}_k(x) \in$ dom α.) A string $x \in \{0,1\}^*$ *forbids* an n-assignment α if $\alpha\tilde{f}(x)$ is defined and $g(x)\alpha\tilde{f}(x) \neq [\![x \in A]\!]$. (This implies that if $A \leq^{\mathrm{P}}_{k-tt} \tilde{C}$ via $(\tilde{f}, g)$ and x forbids α, then $\tilde{C}$ does not satisfy α.) An *n-assignment to* $\tilde{f}(x)$ is an n-assignment α such that dom $\alpha = \{\tilde{f}_1(x), \dots, \tilde{f}_k(x)\} \subseteq \{0,1\}^{\leq n}$. Finally, fix an "end-marker" \$ and, for each $n \in \mathbb{N}$ and n-assignment α, let

$$x(\alpha, n) = \begin{cases} \text{the lexicographically first string } x \in \{0,1\}^{\leq n} \text{ that forbids} \\ \text{some } n-\text{assignments to } \tilde{f}(x) \text{ that is consistent} \\ \text{with } \alpha, \text{ if such an } x \text{ exists,} \\ \$, \quad \text{if no such } x \text{ exists.} \end{cases}$$

Now we turn to the proof of Theorem 3.1.

Proof. Assume that the result is false, that is, assume that $A \in \mathrm{ESPACE} - {}^{\mathrm{P}}_{k-tt}(\mathrm{SPARSE})$. Let f and g be functions such that $A \leq^{\mathrm{P}}_{k-tt} B$ via (f, g). We will show that there is a polynomial r such that

$$(3.2) \qquad KS^r(B_{\leq n}) < 2^{n+1} - 2n \quad \text{i.o.,}$$

thus contradicting the hypothesis.

Let us briefly sketch the argument. For each $n \in \mathbb{N}$, we define a finite tree T_n whose vertices at each level include partial specifications of all languages $\tilde{C}$ such that $A \leq^{\mathrm{P}}_{k-tt} \tilde{C}$ via $(\tilde{f}, g)$. In particular, $\tilde{B}$ will satisfy some leaf of T_n. For a randomly selected language $C \subseteq \{0,1\}^*$, let $\mathcal{E}_l$ be the event that $\tilde{C}$ satisfies some vertex at depth $\geq l$ in T_n. The trees T_n will be constructed so that the probability $\Pr(\mathcal{E}_l)$ decays exponentially as l increases. It will follow by the algorithmic construction of T_n that $KS(C_{\leq n})$ is small for all C such that $\widetilde{C}$ satisfies some vertex that is deep in T_n, thus guaranteeing that $KS(C_{\leq n})$ is small for infinitely many n. The quantitative details will be such that (3.2) holds.

Construction of the trees. For each $n \in \mathbb{N}$, define a tree T_n as follows. Each vertex of each T_n is an n-assignment. The tree T_0 consists of a single vertex, the empty 0-assignment α_ϕ. (That is, dom $\alpha_\phi = \phi$.) The tree T_{n+1} is constructed from T_n by attaching subtrees to zero or more of the leaves of T_n. The vertices of T_n are the *old vertices* of T_{n+1}. The *new vertices* of T_{n+1} are introduced recursively as follows. Let α be a leaf of T_n or a new vertex of T_{n+1}. If $x(\alpha, n+1) = \$$, then α is a leaf of T_{n+1}. Otherwise, the immediate successors of α in T_{n+1}, when they exist, are those $(n+1)$-assignments β such that β is consistent with α, dom $\beta = (\text{dom}\ \alpha) \cup \{\widetilde{f}_1(x(\alpha, n+1)), \ldots, \widetilde{f}_k(x(\alpha, n+1))\}$, and $x(\alpha, n+1)$ does not forbid β.

General properties of T_n. (a) It is clear that, for each vertex α of T_n, dom $\alpha \subseteq W_n \cap \widetilde{W}_n$, where $W_n = \{0,1\}^{\leq n}$.

(b) Note that each vertex of each T_n has at most $2^k - 1$ immediate successors. Also, along any path from the root of T_n to a leaf of T_n, the domain of each vertex is a proper subset of the domain of its immediate successors, so the depth of T_n is at most $\|\{0,1\}^{\leq n}\| = 2^{n+1} - 1$. Thus, each T_n is a finite tree.

(c) By hypothesis $A \in$ ESPACE, and f and g were assumed to be computable in polynomial time. Thus, the trees T_n can be constructed and traversed in space polynomial in their depth.

The key property of the trees T_n is best understood in probabilistic terms. Fix $n \in \mathbb{N}$ and consider the random experiment in which a set $S \subseteq \{0,1\}^{\leq n}$ is chosen probabilistically according to the uniform distribution on all such sets. For each n-assignment α, let F_α be the event that $\widetilde{S}$ satisfies α, and let $F_{\alpha+}$ be the event that $\widetilde{S}$ satisfies some immediate successor of α in T_n. (If α is not an interior vertex of T_n, then $F_{\alpha+} = \phi$. We emphasize that it is S, not $\widetilde{S}$, which is chosen according to the uniform distribution.) The key property of our construction is the following.

Local property. For each interior vertex α of T_n,

$$\Pr(F_{\alpha+}|F_\alpha) \leq 1 - 2^{-k}. \tag{3.3}$$

Proof of the local property. Assume that α is an interior vertex of T_n, and let $x = x(\alpha, m)$, where m is the least integer such that $x(\alpha, m) \neq \$$. (Note that $m \leq n$.) By the construction of T_n, there is an n-assignment β such that β is consistent with α, dom $\beta = (\text{dom}\ \alpha) \cup \{\widetilde{f}_1(x), \ldots, \widetilde{f}_k(x)\}$, and β is not an immediate successor of α in T_n (because x forbids β). Since β is consistent with α, there exists a set $S_0 \subseteq \{0,1\}^{\leq n}$ such that $\widetilde{S_0}$ satisfies both α and β.

Note that dom α and dom β are subsets of $W_n \cap \widetilde{W}_n$, where $W_n = \{0,1\}^{\leq n}$. Define $h : W_n \cap \widetilde{W}_n \to W_{n-1}$ by $h(u10^i) = u$ for all $u10^i \in W_n \cap \widetilde{W}_n$. Note that $F_\alpha = \{S | S \cap h(\text{dom}\ \alpha) = S_0 \cap h(\text{dom}\ \alpha)\}$ and $F_\beta = \{S | S \cap h(\text{dom}\ \beta) = S_0 \cap h(\text{dom}\ \beta)\}$. Let

$$G = \{S | S \cap (h(\text{dom}\ \beta) - h(\text{dom}\ \alpha)) = S_0 \cap (h(\text{dom}\ \beta) - h(\text{dom}\ \alpha))\}.$$

Then G and F_α are independent events whose intersection is F_β, so that $\Pr(F_\beta|F_\alpha) = \Pr(G)$. Since $\|h(\text{dom}\ \alpha)\| \leq \|h(\text{dom}\ \beta - \text{dom}\ \alpha)\| \leq \|\text{dom}\ \beta - \text{dom}\ \alpha\| \leq k$, it follows that $\Pr(f_{\alpha+}|F_\alpha) \leq 1 - \Pr(F_\beta|F_\alpha) = 1 - \Pr(G) = 1 - 2^{-\|h(\text{dom}\beta) - h(\text{dom}\alpha)\|} \leq 1 - 2^{-k}$. □

For each $l \in \mathbb{N}$, let $\mathcal{E}_l$ be the event that $\widetilde{S}$ satisfies some vertex at depth l in T_n. (The set S is still chosen uniformly, with n fixed.) Write $T_n(l)$ for the set of all interior nodes of T_n at depth l.

Claim. For all

$$l \in \mathbb{N}, Pr(\mathcal{E}_l) \leq (1-2^{-k})^l. \tag{3.4}$$

Proof. The local property (3.3) yields

$$\begin{aligned} Pr(\mathcal{E}_{l+1}) &= \sum_{\alpha \in T_n(l)} Pr(F_{\alpha+}) \\ &= \sum_{\alpha \in T_n(l)} Pr(F_{\alpha+} \cap F_\alpha) \\ &= \sum_{\alpha \in T_n(l)} Pr(F_{\alpha+}|F_\alpha) Pr(F_\alpha) \\ &\leq (1-2^{-k}) \sum_{\alpha \in T_n(l)} Pr(F_\alpha) \\ &\leq (1-2^{-k}) Pr(\mathcal{E}_l). \end{aligned}$$

It follows inductively that for all $l \in \mathbb{N}$,

$$Pr(\mathcal{E}_l) \leq (1-2^{-k})^l. \qquad \square$$

For each $n \in \mathbb{N}$, define a path $\beta_{n,0}, \ldots, \beta_{n,J(n)}$ from the root of T_n to a leaf of T_n by the following recursion. First, $B_{n,0} = \alpha_\phi$. For the recursion step, for each interior vertex α of T_n, let $x(\alpha) = x(\alpha, m)$, where m is the least integer such that $x(\alpha, m) \neq \$$. Assume that a path $\beta_{n,0}, \ldots, \beta_{n,j}$ in T_n has been defined so that $\beta_{n,j}$ is an interior node of T_n and $\beta_{n,j}\tilde{f}(x) = [\![f_1(x) \in B]\!] \ldots [\![f_k(x) \in B]\!]$ for each $x = x(\beta_{n,i}), 0 \leq i < j$. (Note that this hypothesis is satisfied vacuously when $j = 0$.) Then $\beta_{n,j+1}$ is the unique n-assignment such that $\beta_{n,j+1}$ is consistent with $\beta_{n,j}$, dom $\beta_{n,j+1} = (\text{dom } \beta_{n,j}) \cup \{\widetilde{f_1}(x), \ldots, \widetilde{f_k}(x)\}$, and $\beta_{n,j+1}\tilde{f}(x) = [\![f_1(x) \in B]\!] \cdots [\![f_k(x) \in B]\!]$, where $x = x(\beta_{n,j})$. (Recall that (f, g) is a $\leq^{\mathrm{P}}_{k-tt}$-reduction of A to B, so this implies that $g(x)(\beta_{n,j}\tilde{f}(x)) = [\![x \in A]\!]$. It follows that x does not forbid $\beta_{n,j+1}$, so that $\beta_{n,j+1}$ is indeed an immediate successor of $\beta_{n,j}$ in T_n.)

For each $n \in \mathbb{N}$, let $\beta_n = \beta_{n,J(n)}$. A routine induction shows that for every $n \in \mathbb{N}$, $\widetilde{B}$ satisfies β_n.

For each $n \in \mathbb{N}$, let $S_{n,1}, \ldots, S_{n,I(n)}$ be the lexicographic enumeration of all sets $S \subseteq \{0,1\}^{\leq n}$ such that $\widetilde{S}$ satisfies some vertex α of T_n whose depth in T_n is at least dn, where

$$d = \frac{3}{k - \log(2^k - 1)}. \tag{3.5}$$

Note that this is precisely an enumeration of $\mathcal{E}_{dn}$, so that from (3.4) we have

$$\begin{aligned} \log I(n) &= 2^{n+1} - 1 + \log Pr(\mathcal{E}_{dn}) \\ &\leq 2^{n+1} - 1 + d \cdot n \cdot \log(1 - 2^{-k}) \\ &= 2^{n+1} - 3n - 1. \end{aligned} \tag{3.6}$$

By hypothesis $A \in$ ESPACE, and f and g were assumed to be computable in polynomial time. Thus, the trees T_n can be constructed and traversed in space polynomial in their depth. Thus, there is a machine M such that, if $1 \leq i \leq l(n)$ and i is written in binary, then $M(i, n)$ outputs the $(2^{n+1} - 1)$-bit characteristic string of $S_{n,i}$ using workspace polynomial in 2^n. Since $\widetilde{B}$ satisfies each β_n, it follows by (3.6) that there is a polynomial r' such that $KS_M^{r'}(B_{\leq n}) \leq 1 + \log I(n) \leq 2^{n+1} - 3n$ for all $n \in D$, where $D = \{n \in \mathbb{N} | J(n) \geq dn\}$. Then, by the optimality of the universal machine, there exists a polynomial r and a constant c such that

$$(3.7) \qquad KS^r(B_{\leq n}) \leq 2^{n+1} - 3n + c$$

for all $n \in D$.

To prove that (3.2) holds, it will be sufficient to show that the set D is infinite.

Claim. D is infinite.

Proof. Let $B' = \cup_{n \geq 0} \beta_n^{-1}(\{1\})$. Since $\widetilde{B}$ satisfies each β_n, it is clear that $B' \subseteq \widetilde{B}$. Fix a strictly increasing polynomial s such that $|\widetilde{f}_i(x)| \leq s(|x|)$ for all $x \in \{0,1\}^*$ and $1 \leq i \leq k$. Then

$$(3.8) \qquad B'_{\leq n} \subseteq \beta_{s(n)}^{-1}(\{1\})$$

for all $n \in \mathbb{N}$. To see that this is true, let $y \in B'_{\leq n}$. Then $y = \widetilde{f}_i(x)$ for some $x \in \{0,1\}^*$ and $i, 1 \leq i \leq k$. By (3.1), $|x| < n$, so that $|\widetilde{f}_j(x)| \leq s(n)$ for all $j, 1 \leq j \leq k$. Since $x(\beta_{s(n)}, s(n)) = \$$, x does not forbid any $s(n)$-assignment to $\tilde{f}(x)$ that is consistent with $\beta_{s(n)}$. It follows that $y = \widetilde{f}_i(x)$ is not in (dom β_{m+1}) $-$ (dom β_m) for any $m \geq s(n)$. Since $y \in B' = \cup_{m \geq 0} B_m^{-1}(\{1\})$. It follows that $y \in \beta_{s(n)}^{-1}(\{1\})$, confirming (3.8).

We have already noted that $(\tilde{f}, g)$ is a $\leq_{k-tt}^{\mathrm{P}}$-reduction of A to $\widetilde{B}$. In fact, $(\tilde{f}, g)$ is a $\leq_{k-tt}^{\mathrm{P}}$-reduction of A to B'. To see this, fix $x \in \{0,1\}^*$, let $n = |x|$, and let $m = s(s(n))$. Let β'_m be the unique m-assignment such that β'_m is consistent with β_m, dom β'_m = (dom β_m) $\cup \{\widetilde{f}_1(x), \ldots, \widetilde{f}_k(x)\}$, and $\beta'_m(\widetilde{f}_i(x)) = 0$ for each $i, 1 \leq i \leq k$, such that $\widetilde{f}_i(x) \notin$ dom β_m. Since $x(\beta_m, m) = \$$, x does not forbid β'_m. Since $\beta'_m \tilde{f}(x)$ is defined, it follows by (3.8) that

$$\begin{aligned}
[\![x \in A]\!] &= g(x)(\beta'_m \tilde{f}(x)) \\
&= g(x)(\beta'_m(\widetilde{f}_1(x)) \cdots \beta'_m(\widetilde{f}_k(x))) \\
&= g(x)([\![\widetilde{f}_1(x) \in B'_{\leq s(n)}]\!] \cdots [\![\widetilde{f}_k(x) \in B'_{\leq s(n)}]\!]) \\
&= g(x)([\![\widetilde{f}_1(x) \in B']\!] \cdots [\![\widetilde{f}_k(x) \in B']\!]),
\end{aligned}$$

confirming that $A \leq_{k-tt}^{\mathrm{P}} B'$ via $(\tilde{f}, g)$. By assumption, $A \notin \mathrm{P}_{k-tt}(\mathrm{SPARSE})$ so that we have

$$(3.9) \qquad \|B'_{\leq n}\| > k \cdot d \cdot s(n) \quad \text{i.o.}$$

On the other hand,

$$(3.10) \qquad \|\mathrm{dom}\, \beta_{s(n)}\| \leq k \cdot \mathrm{depth}(\beta_{s(n)}) = k \cdot J(s(n)).$$

By (3.8), (3.9), and (3.10), $J(s(n)) > d \cdot s(n)$ for infinitely many n. Since s is strictly increasing, it follows that the set D is infinite.

Since D is infinite, (3.2) follows from (3.7). This completes the proof of Theorem 3.1. □

Consider the reducibility specified by the pair (f, g) in the proof of Theorem 3.1. What properties were used? Time as such played no role. Another measure of computation such as space or time-space could have been used; for example, $\leq_{k-tt}^{\mathrm{PSPACE}}$ could have been used instead of $\leq^{\mathrm{P}}_{k-tt}$. The fact that the reducibility was specified by deterministic machines plays no role except for the fact that both f and g are functions; another mode of computation could have been used as long as functions are used to specify the reducibility. The hypothesis that $A \in$ ESPACE, combined with the fact that f and g could be computed in polynomial time, allowed the trees T_n to be constructed and traversed in space polynomial in their depth. In each case the corresponding result would hold and the proof would be essentially the same as that of Theorem 3.1.

4. Applications. We develop some applications of Theorem 3.1. In each case we consider languages satisfying the lower bounds on the space-bounded Kolmogorov complexity of the language. In order to eliminate the repetition of awkward phrases, we introduce a new definition.

The class HIGH is the collection of all languages B that for every polynomial q, satisfy $KS^q(B_{\leq n}) > 2^{n+1} - 2n$ almost everywhere.

As noted in §3, if a language is in HIGH, then it has essentially maximal information content.

Almost every language is in HIGH. This follows from the fact [Ma71] that RAND $\subset$ HIGH, where RAND is the set of *algorithmically random languages* as defined by Martin-Löf [Ma66]. Martin-Löf showed that almost every language is in RAND. In fact, the inclusion of RAND in HIGH is proper since almost every recursive language is in HIGH [Lu92], while no recursively enumerable (hence, no recursive) language is in RAND. On the other hand, HIGH $\cap$ ESPACE $= \phi$, that is, no language recognized by a machine that uses workspace $O(2^{cn})$ for any $c > 0$ is in HIGH.

Recall that Theorem 3.1 shows that for every integer $k > 0$, P_{k-tt}(HIGH) $\cap$ ESPACE $\subseteq \mathrm{P}_{k-tt}$(SPARSE).

The following result shows that it is highly unlikely that any language in HIGH is $\leq^{\mathrm{P}}_{btt}$-hard for many of the classes studied in structural complexity theory. Recall that the reducibility $\leq^{\mathrm{P}}_{btt}$ is transitive.

THEOREM 4.1. *Let K be any class chosen from* {PSPACE, NP, PP, $\mathrm{C_{=}P}$, $\mathrm{MOD_2P}$, $\mathrm{MOD_3P}, \ldots$}. *If there is a language in* HIGH *that is $\leq^{\mathrm{P}}_{btt}$-hard for K, then $K =$* P.

Proof. Ogiwara and Watanabe [OW91] have shown that if there is a sparse set that is $\leq^{\mathrm{P}}_{btt}$-hard for NP, then P = NP. Later, Ogiwara and Lozano [OL91] extended that result so that it holds for the other choices of K. Theorem 3.1 shows that if there is a language $A \in$ HIGH such that A is $\leq^{\mathrm{P}}_{btt}$-hard for any such class K, then there is a sparse set S such that S is $\leq^{\mathrm{P}}_{btt}$-hard for K. Hence, $K =$ P. □

Thus, Theorem 4.1 shows that for any class K chosen from {PSPACE, NP, PP, $\mathrm{C_{=}P}$, $\mathrm{MOD_2P}$, $\mathrm{MOD_3P}, \ldots$}, if P $\neq K$, then no language in HIGH can be $\leq^{\mathrm{P}}_{btt}$-hard for K. A similar consequence holds for E = DTIME(2^{linear}) with no unproven hypothesis.

THEOREM 4.2. *No language in* HIGH *is $\leq^{\mathrm{P}}_{btt}$-hard for* E.

Proof. Watanabe [Wa87] has shown that no sparse set is $\leq^{\mathrm{P}}_{btt}$-hard for E, so by Theorem 4.1 no language in HIGH is $\leq^{\mathrm{P}}_{btt}$-hard for E. □

The main theorem was stated in terms of a fixed integer k that bounds the number of queries. Instead, we could consider a function $k(n) = O(\log n)$ that is computable in polynomial time and replace the fixed integer k with the function $k(n)$ to obtain a bound on the number of queries. Thus, $\mathrm{P}_{k(n)-tt}$(HIGH) $\cap$ ESPACE $\subseteq \mathrm{P}_{k(n)-tt}$(SPARSE). In addition, the proof of the main theorem allows us to conclude that P_m(HIGH) $\cap$ ESPACE $\subseteq \mathrm{P}_m$(SPARSE) and

$$\mathrm{P}_{(\log n)-dtt}(\mathrm{HIGH}) \cap \mathrm{ESPACE} \subseteq \mathrm{P}_{(\log n)-dtt}(\mathrm{SPARSE}).$$

The results have been stated in terms of languages reducible to languages with extremely high space-bounded Kolmogorov complexity. No such language is in ESPACE. However, the proof of the main theorem yields a bound on the Kologogorov complexity of languages to which languages in ESPACE can be reduced, and this bound allows us to make additional conclusions.

COROLLARY 4.3. *For every* $A \in \mathrm{ESPACE} - \mathrm{P}_{btt}(\mathrm{SPARSE})$, *there exists a polynomial* r *such that for all languages* B, *if* $A \leq^{\mathrm{P}}_{btt} B$, *then* $KS^r(B_{\leq n}) < 2^{n+1} - 2n$ *i.o.*

Corollary 4.3 implies that hard languages have unusually low Kolmogorov complexity. Moreover, the fact that the polynomial r is fixed enables us to strengthen Theorem 4.1 as follows.

THEOREM 4.4. (a) *If* $\mathrm{P} \neq \mathrm{NP}$, *then there is a* fixed *polynomial* q *such that every* $\leq^{\mathrm{P}}_{btt}$-*hard language* H *for* NP *has space-bounded Kolmogorov complexity* $KS^q(H_{\leq n}) < 2^{n+1} - 2n$ *i.o.*

(b) *There is a* fixed *polynomial* q *such that every* $\leq^{\mathrm{P}}_{btt}$-*hard language* H *for* E *has space-bounded Kolmogorov complexity* $KS^q(H_{\leq n}) < 2^{n+1} - 2n$ *i.o.*

Proof. (a) If $\mathrm{P} \neq \mathrm{NP}$, then there exists a language $A \in \mathrm{NP} - \mathrm{P}_{btt}(\mathrm{SPARSE})$ [OW91]. Fix q for A as in Corollary 4.3. Then every $\leq^{\mathrm{P}}_{btt}$-hard language H for NP satisfies $A \leq^{\mathrm{P}}_{btt} H$, while $KS^q(H_{\leq n}) < 2^{n+1} - 2n$ i.o.

(b) The proof is the same as for (a) except that we fix $A \in \mathrm{E} - \mathrm{P}_{btt}(\mathrm{SPARSE})$ [Wa87]. □

REFERENCES

[BDG88] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, Springer-Verlag, New York, 1988.

[BDG90] ———, *Structural Complexity* II, Springer-Verlag, New York, 1990.

[KL82] R. KARP AND R. LIPTON, *Turing machines that take advice*, Enseign. Math., 28 (1982), pp. 191–209.

[LV90] M. LI AND P. VITANYI, *Kolmogorov complexity and its applications*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Vol. A, Elsevier, Amsterdam, 1990, pp. 187–254.

[Lu92] J. LUTZ, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[Ma82] S. MAHANEY, *Sparse complete set for* NP: *Solution to a conjecture by Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.

[Ma66] P. MARTIN-LÖF, *On the definition of random sequences*, Inform. and Control, 9 (1966), pp. 602–619.

[Ma71] ———, *Complexity oscillations in infinite binary sequences*, Z. Wahrscheinlichkeitstheorie und Verwandte Gebiete, 19 (1971), pp. 225–230.

[OL91] M. OGIWARA AND A. LOZANO, *On one query self-reducible sets*, Proc. 6th IEEE Conference on Structure in Complexity Theory, Chicago, IL 1991, pp. 139–151.

[OW91] M. OGIWARA AND O. WATANABE, *On polynomial bounded truth-table reducibility of* NP *sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.

[Wa87] O. WATANABE, *Polynomial time reducibility to a set of small density*, Proc. 2nd IEEE Conference on Structure in Complexity Theory, 1987, pp. 138–146.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 403–417, April 1993

COIN-FLIPPING GAMES IMMUNE AGAINST LINEAR-SIZED COALITIONS*

NOGA ALON† AND MONI NAOR‡

Abstract. Perfect information coin-flipping and leader-election games arise naturally in the study of fault tolerant distributed computing and have been considered in many different scenarios. This paper answers a question of Ben-Or and Linial by proving that for every $c < 1$ there are such games on n players in which no coalition of cn players can influence the outcome with probability greater than some universal constant times c. (Note that this paper actually proves this statement only for all $c < \frac{1}{3}$, but since its universal constant is bigger than 3 the above is trivial for $c \geq \frac{1}{3}$.) This paper shows that a random protocol of a certain length has this property and gives an explicit construction as well.

Key words. fault tolerance, distributed computing, perfect information games, coin-flipping, leader election

AMS(MOS) subject classifications. 68C05, 05B25, 90D99

1. Introduction.

A fundamental problem of fault tolerant distributed computing is that of n processors wishing to agree on a random value. The problem becomes nontrivial when some of the processors are faulty. The problem has been considered in many different scenarios, depending on the assumptions made on the type of communication between the processors, the kind and number of faults, and the power of the adversary. See the surveys of Chor and Dwork [9] and Ben-Or, Linial, and Saks [6]. In the present paper we consider it in the natural model formulated by Ben-Or and Linial [5] (whose formal description is given in the next subsection): the processors have complete information, (i.e., the communication type is a public broadcast channel), the processors take turns broadcasting some random values and the outcome is a function of all the bits that were sent. The adversary is assumed to be computationally unlimited. The problem is to design protocols where the influence on the outcome of any set of faulty processors not exceeding a certain size is bounded.

A closely related problem is that of leader election: the processors take turns broadcasting messages. At the end of the protocol, as a function of the bits transmitted, one processor is considered the leader. The problem is to design protocols that have the property that for any coalition of faulty processors whose size does not exceed a certain threshold, the probability that a member of the coalition be elected is bounded.

Unlike the Byzantine case, where the exact thresholds for achieving an agreement were known [9], [10], for the perfect information scenario a gap existed: it was known that $n/2$ cheaters (out of n players) can completely control the protocol, yet the best known protocol [12], [1] (improving the one in [5]) has the property that only sets of cheaters of size less than $n/3 \log n$ have influence bounded away from 1.

In this paper we resolve this problem and show that there are protocols where even a linear number of cheaters have only bounded influence. We first show in §2, via a *probabilistic construction*, the existence of an election protocol that can tolerate up to $\frac{1}{3}n$ cheaters (i.e., the elected leader will not be faulty with some nonzero probability). Then, in §3, we show an *explicit construction* that works for a smaller (yet linear in n) threshold. This easily gives coin-flipping protocols with a similar behavior.

*Received by the editors August 12, 1991; accepted for publication (in revised form) September 5, 1991.

†Department of Mathematics, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv, Israel (noga@taurus.bitnet). Most of this work was done while the author was visiting IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120.

‡Department K53/802, IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 (naor@ibm.com).

Our proofs combine probabilistic arguments with an iterative procedure based on the pseudorandom properties of projective and affine planes. The analysis requires a study of a game that we call faulty baton passing, which is performed in each iteration of the procedure.

1.1. Preliminaries, background, and results. A *perfect-information coin-flipping game* of n players is a rooted tree T whose leaves are labeled by zero or 1 and whose internal vertices are labeled by the names of the players. In addition, each internal vertex v is associated with a probability distribution D_v on its children. Starting from the root, the player whose name labels the current vertex v chooses one of its children according to the distribution D_v, and the game proceeds to the chosen child. When a leaf is reached the game ends and its value is the label of the leaf. Note that the same player may have to make more than one choice in a game. It is sometimes assumed that the tree T is binary and that the probability distribution associated with each internal vertex is the uniform distribution on its two children. This makes no essential difference, and hence we use here the more general definition.

Let N denote the set of players. We say that a player $I \in N$ plays *fairly* if he makes his choice randomly (according to the corresponding probability distribution) whenever it is his turn to make a choice in the course of the game.

Let p_1^T be the probability of reaching a leaf labeled 1 if all players play fairly. T is a *fair* game if $p_1(T) = \frac{1}{2}$. For a subset S of the set of players N, let $p_1^T(S)$ denote the probability of reaching a 1-leaf when the coalition S plays the optimal strategy trying to maximize the probability of reaching a 1-value. Here we assume that all the other players play fairly and that each player in S knows exactly which other players are in S. The *influence* $I_1^T(S)$ *of* S *towards* 1 *in the game* T is defined by $I_1^T(S) = p_1^T(S) - p_1^T$. The influence $I_0^T(S)$ of S towards zero in T is defined similarly and the (*total*) *influence* $I(S) = I^T(S)$ is defined by $I(s) = I_1^T(S) + I_0^T(S)$. Therefore, $I(S)$ measures the capability of S to control the game, and a game is robust if $I(S)$ is small for every relatively small S. In [5] it is proved that for every perfect-information coin-flipping game T of a set N of n players, where p_1^T is bounded away from zero and 1, and for every $k \leq n$ there is a subset $S \subset N$ of cardinality k whose influence $I(S)$ is at least $\Omega(k/n)$. In the same paper the authors construct a fair game of n players in which the influence of each set of k players, where $k \leq O(n^{\log_3 2})$ is at most $O(k/n)$. Improving the estimates of Saks [12], together with the main result of [11], Ajtai and Linial [1] showed that there is a fair game on n players in which the influence of each set of $k \leq n/3 \log n$ is at most $O(k/n)$. These results lead to the following problem, raised in [5], and referred to as the most outstanding problem in this area in some of the more recent papers on the subject.

Problem (see [5] and [1]). Are there fair perfect-information coin-flipping games of n players in which for every $k \leq n$ the influence of every set of k players is at most $O(k/n)$? In particular, are there such games in which there is no set of size $o(n)$ whose influence is $1 - o(1)$?

In the present paper we show that the answer to both questions is "yes." We first present a probabilistic proof of existence of such games. Afterwards we describe, for every positive $c < 1$, an explicit construction of perfect-information coin-flipping fair games of n players in which there is no set of cn players whose influence exceeds $O(c)$. (Note that the construction works only for all positive real $c < C$, where C is some absolute positive constant; but since the multiplicative constant in the expression $O(c)$ can be adjusted, this implies the result for all $c < 1$.)

Our results are better formulated in terms of leader-election games. A *leader-election game* of n players is a rooted tree T whose vertices are labeled by the names of the players, and each internal vertex v is associated with a probability distribution D_v on its children. Starting from the root, the player whose name labels the current vertex v chooses one of its children according to the distribution D_v, and the game proceeds to the chosen child. When a leaf is reached the game ends and the chosen leader is the label of this leaf.

For a subset S of players, let $p^T(S)$ denote the probability that a leader from S is chosen, when the coalition S plays the optimal strategy trying to maximize the probability of such a choice, (and when all other players play fairly). For a constant $\delta > 0$ and for $t \leq n$ let us call, following [12], a leader-election game T *δ-robust against t-cheaters* if $p^T(S) \leq \delta$ for any subset S of at most t players. Let $t(n, \delta)$ denote the maximum t such that there exists a leader-election game of n players that is δ-robust against t cheaters. From any leader-election game T we can construct its associated coin-flipping fair game $C(T)$ obtained by letting the chosen leader flip a coin and decide the value of the game. Thus, formally, $C(T)$ is the game obtained from the tree T by adding to each leaf v of T two children labeled zero and 1, and by associating v with the uniform distribution on these two children. It is obvious that for a subset S of players, the influence of S towards zero or towards 1 in $C(T)$ is precisely $p^T(S)/2$. Therefore, the existence of robust leader-election games implies the existence of robust coin-flipping games. This leads naturally to the problem of estimating $t(n, \delta)$.

In [12] Saks constructed a leader-election game that is δ-robust against $c(\delta)n/\log n$ cheaters, for any $0 < \delta < 1$, where $c(\delta)$ is a positive constant depending only on δ. This shows that $t(n, \delta) > c(\delta)n/\log n$. Answering a question raised in [12] we show that in fact $t(n, \delta) > \Omega(\delta n)$ for all $0 < \delta < 1$. Moreover, for any $\epsilon < \frac{1}{3}$ there is a $\delta < 1$ such that $t(n, \delta) > \epsilon n$. As noted in [12] there is a simple argument that shows that $t(n, \delta) < n/2$ for all $\delta < 1$. It would be interesting to close the gap between the two constants $\frac{1}{2}$ and $\frac{1}{3}$ here.

Our best lower bounds (in terms of the constants) for $t(n, \delta)$ are obtained by probabilistic arguments, described in the next section. However, we also give an explicit construction of leader-election games of n players that are δ-robust against $\Omega(\delta n)$-cheaters.

2. The existence of robust games. Let T be a full binary rooted tree of depth d. Put $N = \{1, \ldots, n\}$, and let us label each internal vertex v of T, randomly and independently, by a number in N chosen according to a uniform distribution on N. Observe that T is a leader-election game of n players. Our first result in this section is the following theorem, which demonstrates the existence of very robust leader-election games.

THEOREM 2.1. *Let T be the leader-election game chosen randomly as above. Then, the probability that there is a set $S \subset N$ of cardinality ϵn, where $\epsilon < \frac{1}{3}$, such that $p^T(S)$ is at least*

$$\epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} + \gamma$$

does not exceed

$$\binom{n}{\epsilon n} \frac{\epsilon((1/2) + (3\epsilon/2))^d}{\gamma^2}.$$

In particular, there is an n-player leader-election game T of depth $O(n)$ in which for any set of $\epsilon n \leq \frac{1}{4} n$ players $p^T(S) \leq \epsilon + 12\epsilon^{3/2}$.

Note that even if all players in an n-players leader-election game play fairly, then for every ϵ (which is an integral multiple of $1/n$) there is a set S of ϵn players such that the leader is chosen among the members of S with probability of at least ϵ. This shows that the last estimate for $p^T(S)$ in the theorem above is sharp, up to the additive lower order term $12\epsilon^{3/2}$.

The theorem is proved by deriving estimates on the expectation and variance of the value of $p^T(S)$ in a randomly chosen leader-election game. Then, using Chebyshev's inequality it is shown that with the required probability there is no set S where $p^T(S)$ is larger than the bound in the theorem. A similar strategy is used in the proof of the main result of [4].

First we establish the following two lemmata.

LEMMA 2.2. *Let Y, Z be two independent random variables with equal expectations $E(Y) = E(Z) = E$ and equal variances* $\mathrm{Var}(Y) = \mathrm{Var}(Z) = \sigma^2$. *Let $\epsilon \leq 1$ be a positive constant and let X be the random variable defined as $(Y+Z)/2$ with probability $1-\epsilon$ and* $\mathrm{Max}\{Y, Z\}$ *with probability ϵ. Then*

$$E(X) = E + \frac{\epsilon}{2}E(|Y-Z|) \leq E + \frac{\epsilon\sigma}{\sqrt{2}} \tag{1}$$

$$\mathrm{Var}(X) \leq \sigma^2\left(\frac{1}{2} + \frac{3\epsilon}{2}\right). \tag{2}$$

Proof. Observe that $\mathrm{Max}(Y, Z) = (Y+Z)/2 + |Y-Z|/2$. By Jensen's inequality

$$E(|Y-Z|) \leq \sqrt{E((Y-Z)^2)} = \sqrt{\mathrm{Var}(Y-Z)} = \sqrt{\mathrm{Var}(Y) + \mathrm{Var}(Z)} = \sqrt{2}\sigma.$$

Hence,

$$E(X) = (1-\epsilon)E + \epsilon\left(E + \frac{1}{2}E(|Y-Z|)\right) = E + \frac{\epsilon}{2}E(|Y-Z|) \leq E + \frac{\epsilon}{2}\sqrt{2}\sigma.$$

Thus (1) holds.

To prove (2) observe, first, that by the Cauchy–Schwarz inequality, for every two random variables T, F,

$$(E(TF) - E(T)E(F))^2 = (E((T-E(T))(F-E(F))))^2 \leq \mathrm{Var}(T)\mathrm{Var}(F).$$

(This is simply the well-known fact that the absolute value of the correlation factor of two random variables is at most 1.) Taking $T = Y+Z$, $F = |Y-Z|$ we get

$$|E((Y+Z)|Y-Z|) - E(Y+Z)E(|Y-Z|)| \leq \sqrt{\mathrm{Var}(Y+Z)\mathrm{Var}(|Y-Z|)} \leq 2\sigma^2. \tag{3}$$

By the definition of X:

(4)
$$E(X^2) = (1-\epsilon)\frac{E(Y^2) + E(Z^2) + 2E(YZ)}{4}$$
$$+\epsilon\frac{E(Y^2) + E(Z^2) + 2E(YZ)}{4} + 2\frac{\epsilon}{4}E((Y+Z)|Y-Z|) + \frac{\epsilon}{4}E(|Y-Z|^2)$$
$$= \frac{E(Y^2) + E(Z^2) + 2E^2}{4} + \frac{\epsilon}{2}E((Y+Z)|Y-Z|) + \frac{\epsilon}{4}E(|Y-Z|^2).$$

Also, by (1),

$$(5)\qquad \begin{aligned} E(X)^2 &= E^2 + \epsilon E\, E(|Y-Z|) + \frac{\epsilon^2}{4}(E(|Y-Z|))^2 \\ &= E^2 + \frac{\epsilon}{2}E(Y+Z)E(|Y-Z|) + \frac{\epsilon^2}{4}(E(|Y-Z|))^2. \end{aligned}$$

Subtracting (5) from (4) we obtain

$$\begin{aligned} \operatorname{Var}(X) &= E(X^2) - (E(X))^2 \\ &= \frac{E(Y^2)+E(Z^2)-2E^2}{4} + \frac{\epsilon}{2}(E((Y+Z)|Y-Z|) - E(Y+Z)E(|Y-Z|)) \\ &\quad + \frac{\epsilon}{4}E(|Y-Z|^2) - \frac{\epsilon^2}{4}(E(|Y-Z|))^2. \end{aligned}$$

In view of (3), the last quantity is at most

$$\begin{aligned} &\frac{\operatorname{Var}(Y)+\operatorname{Var}(Z)}{4} + \frac{\epsilon}{2}2\sigma^2 + \frac{\epsilon}{4}E(|Y-Z|^2) \\ &= \frac{\sigma^2}{2} + \epsilon\sigma^2 + \frac{\epsilon}{4}2\sigma^2 = \sigma^2\left(\frac{1}{2}+\frac{3}{2}\epsilon\right), \end{aligned}$$

completing the proof of (2). □

In order to state the next lemma we need some more notation. Let T be, as before, a full binary rooted tree of depth d, whose vertices are labeled randomly and independently by the elements of $N = \{1,\dots,n\}$. Let $a < 1$ be a positive number, and let us choose, for every leaf v of T randomly and independently, a weight $w(v)$, where $w(v) = 1$ with probability a and $w(v) = 0$ with probability $1-a$. Let $S \subset N$ be a fixed subset of cardinality $|S| = \epsilon n$. Define a weight function $w_{a,S}$ on the vertices of T as follows. If v is a leaf of T, then $w_{a,S}(v) = w(v)$. If u is an internal vertex of T and v_1, v_2 are its two children, then

$$\begin{aligned} w_{a,S}(u) &= \operatorname{Max}\{w_{a,S}(v_1), w_{a,S}(v_2)\} \quad \text{if } u \text{ is labeled by an element of } S, \quad \text{and} \\ w_{a,S}(u) &= \frac{w_{a,S}(v_1)+w_{a,S}(v_2)}{2} \quad \text{otherwise.} \end{aligned}$$

Obviously, for fixed S, a, and for every fixed vertex v of T, $w_{a,S}(v)$ is a random variable whose value depends on the random choices of the labels of the internal vertices of T and on the random choices of the weights $w(v)$ of the leaves of T.

LEMMA 2.3. *Let v be a vertex of T whose distance from the leaves is h. Then the expectation and the variance of the random variable $w_{a,S}(v)$ satisfy*

$$\begin{aligned} E(w_{a,S}(v)) &\le a + \epsilon\frac{\sqrt{a}}{\sqrt{2}}\sum_{i=0}^{h-1}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{i/2} \le a + \epsilon\frac{\sqrt{a}}{\sqrt{2}(1-\sqrt{(1/2)+(3\epsilon/2)})}, \\ \operatorname{Var}(w_{a,S}(v)) &\le a\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^h. \end{aligned}$$

Proof. We apply induction on h. For $h = 0$ the expectation and the variance of $w_{a,S}(v)$ are a and $a(1-a)$, respectively, and hence both inequalities hold. Assuming the result holds for all $h' < h$, let u be a vertex of distance h from the leaves, $h > 1$. Let v_1 and v_2 be the two children of u. Since the label of u is in S with probability ϵ, and since $w_{a,S}(v_1)$ and $w_{a,S}(v_2)$ are two identically distributed independent random variables, we can apply Lemma 2.2 with $X = w_{a,S}(u)$, $Y = w_{a,S}(v_1)$, $Z = w_{a,S}(v_2)$. By the induction hypothesis this gives

$$\begin{aligned} E(w_{a,S}(u)) &\le E(w_{a,S}(v_1)) + \frac{\epsilon}{\sqrt{2}}\sqrt{\operatorname{Var}(w_{a,S}(v_1))} \\ &\le a + \epsilon\frac{\sqrt{a}}{\sqrt{2}}\sum_{i=0}^{h-2}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{i/2} + \frac{\epsilon}{\sqrt{2}}\sqrt{a}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{(h-1)/2} \\ &= a + \epsilon\frac{\sqrt{a}}{\sqrt{2}}\sum_{i=0}^{h-1}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{i/2}. \end{aligned}$$

Similarly, by Lemma 2.2 and by the induction hypothesis,

$$\begin{aligned} \operatorname{Var}(w_{a,S}(u)) &\le \operatorname{Var}(w_{a,S}(v_1))\left(\frac{1}{2}+\frac{3\epsilon}{2}\right) \\ &\le a\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{h-1}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right) = a\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{h}. \end{aligned}$$

This completes the proof. □

Returning, now, to our randomly chosen leader-election game given by the randomly labeled tree T of depth d, let us fix a set S with $S = \epsilon n$, where $\epsilon < \frac{1}{3}$, and let us estimate the probability that for this specific set S, the inequality

$$p^T(S) > \epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} + \gamma$$

holds. For every vertex v, let T_v denote the subtree of T rooted at v. If the leader-election game is played on T_v, then the probability that a leader from S is chosen, when the coalition S plays the optimal strategy trying to maximize the probability of such a choice, is $p^{T_v}(S)$. Obviously, if v is a leaf of T, then $p^{T_v}(S)$ is 1 if the label of v is in S and is zero otherwise. More interesting is the case that v is an internal vertex of T and u and w are its two children. It is not too difficult to check that in this case:

$$\begin{aligned} p^{T_v}(S) &= \operatorname{Max}\{p^{T_u}(S), p^{T_w}(S)\} \quad \text{if } v \text{ is labeled by an element of } S, \quad \text{and} \\ p^{T_v}(S) &= \frac{p^{T_u}(S) + p^{T_w}(S)}{2} \quad \text{otherwise.} \end{aligned}$$

Therefore, the random variables $p^{T_v}(S)$ are defined exactly as the random variables $w_{a,S}(v)$ discussed in Lemma 2.3, where here $a = \epsilon$. It follows that the expectations and variances of these random variables satisfy the bounds appearing in this lemma (with $a = \epsilon$). In particular, when we let v be the root of T we conclude that the expectation and the variance of $p^{T_v}(S) = p^T(S)$ satisfy

$$\begin{aligned} E(p^T(S)) &\le \epsilon + \epsilon\frac{\sqrt{\epsilon}}{\sqrt{2}}\sum_{i=0}^{d-1}\left(\frac{1}{2}+\frac{3\epsilon}{2}\right)^{i/2} \\ &\le \epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})}, \end{aligned}$$

and

$$\mathrm{Var}(p^T(S)) \le \epsilon \left(\frac{1}{2} + \frac{3\epsilon}{2}\right)^d .$$

Combining this with Chebyshev's inequality we obtain the following.

LEMMA 2.4. *Let S be a fixed set of $\epsilon n < \frac{1}{3}n$ players, and let T be the leader-election game chosen randomly as above. Then, for every positive γ, the probability that*

$$p^T(S) \ge \epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} + \gamma$$

does not exceed

$$\frac{\epsilon((1/2) + (3\epsilon/2))^d}{\gamma^2}.$$

Proof of Theorem 2.1. By Lemma 2.4, for every fixed subset of players S of cardinality $|S| = \epsilon n < \frac{1}{3}n$, the probability that $p^T(S)$ exceeds

$$\epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} + \gamma$$

does not exceed

$$\frac{\epsilon((1/2) + (3\epsilon/2))^d}{\gamma^2}.$$

Since the number of choices for S is $\binom{n}{\epsilon n}$, the desired result follows. □

Theorem 2.1 shows that for every ϵ that satisfies

$$\epsilon + \frac{\epsilon^{3/2}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} < 1,$$

there is a $\delta < 1$ such that $t(n, \delta) \ge \epsilon n$; i.e., there are leader-election games on n players which are δ-robust against ϵn-cheaters. This does not suffice to prove the existence of such a δ for ϵ, which is, e.g., at least $\frac{1}{4}$. Still, we can modify the proof to show that such a δ exists for every $\epsilon < \frac{1}{3}$. To do so, we first need one of the simple properties of the *baton-passing game.* The baton-passing game, first analyzed in [12], is a leader-election game where a leader is chosen by passing a baton among the players. Initially the baton is held by an arbitrary player, and each player that receives the baton picks (randomly) a player that has not been selected so far and passes the baton to him. The leader is the last person to hold the baton. As was already mentioned, in any leader-election game T of n players there is a coalition S of at most $\lceil n/2 \rceil$ players such that $p^T(S) = 1$, i.e., S can guarantee a leader from the coalition. The baton-passing game shows that this is sharp. This is because if there are n players, and S is a set of less than half of them, then there is a positive (though exponentially small) probability that whenever a player not in S has to pass the baton and there are still yet unselected players from S he chooses a player from S. If this happens, the chosen leader will not be in S. We have thus proved the following simple lemma.

LEMMA 2.5. *For every integer m there is a leader-election game G of m players such that $p^G(S) < 1$ for each set S of less than $m/2$ players.*

COROLLARY 2.6. *For every $\epsilon < \frac{1}{3}$ there is a $\delta < 1$ such that for every n there is an n-players leader-election game that is δ-robust against ϵn-cheaters.*

Proof. Given $e < \frac{1}{3}$ let us choose small positive constants $a = a(\epsilon)$ and $\gamma = \gamma(\epsilon)$ such that

$$a + \epsilon \frac{\sqrt{a}}{\sqrt{2}(1 - \sqrt{(1/2) + (3\epsilon/2)})} + \gamma < \frac{1}{2}.$$

Next choose a sufficiently large integer h so that

$$\binom{n}{\epsilon n} \frac{a((1/2) + (3\epsilon/2))^h}{\gamma^2} < 1.$$

Finally choose a sufficiently large $m = m(\epsilon, a)$ so that

$$\sum_{i \geq m/2} \binom{m}{i} \epsilon^i (1 - \epsilon)^{m-i} \leq a.$$

Let T be a full binary rooted tree of depth h, and let us choose, for each *internal* vertex of T, randomly and independently, a label in the set of players $N = \{1, \ldots, n\}$. For each leaf v of T we choose, randomly an independently (with repetitions) a (multi) set M_v of m members of N. Consider the following leader-election game represented by T. Starting at the root, the player whose name labels the current vertex chooses randomly one of its two children, and the game proceeds to the chosen child. When a leaf v is reached, a leader is chosen by playing the baton-passing game among the players in M_v. Observe that by Lemma 2.5, for every coalition S of ϵn players, if, during the game, the set M_v corresponding to the chosen leaf contains less than $m/2$ members of S, then the probability that the leader will be chosen from S is bounded away from 1 (i.e., is less than 1 by a constant depending on m, which depends only on ϵ (and $a = a(\epsilon)$) and is independent of the total number of players n.) Thus, to complete the proof, it suffices to bound away from 1 the probability that the coalition S will succeed in reaching a leaf whose set contains more than $m/2$ members of S. However, by the inequalities used in defining the quantities m, a, γ, and h, and by applying Lemma 2.3 and repeating the arguments used in the proofs of Lemma 2.4 and Theorem 2.1 we conclude that there is a tree T such that for every set S of cardinality ϵn the above probability is at most $\frac{1}{2}$. This completes the proof. □

Remark. The constant $\frac{1}{3}$ can be probably increased a bit by a more careful analysis in the proof of Lemma 2.2. This would not suffice, however, to close the gap between this increased constant and $\frac{1}{2}$. We suspect that for every $\epsilon < \frac{1}{2}$ there is a $\delta < 1$ such that for every n there are leader-election games of n players that are δ-robust against ϵn players. This remains open.

As mentioned in the introduction, robust leader-election games supply robust coin-flipping games by allowing the leader choose the random bit. Therefore, as a simple consequence of Theorem 2.1 and Corollary 2.6, we obtain the following.

THEOREM 2.7. (i) *There are fair n-players coin flipping games of depth $O(n)$ such that the influence of every set of $\epsilon n \leq \frac{1}{4} n$ players towards zero or towards* 1 *is at most $\frac{1}{2}\epsilon + 6\epsilon^{3/2}$.*

(ii) *For every $\epsilon < \frac{1}{3}$ there are fair n-players coin-flipping games such that the influence of every set of ϵn players towards zero or towards* 1 *is bounded away from $\frac{1}{2}$.*

This theorem solves the problem mentioned in the introduction. Its proof, as well as that of Theorem 2.1, is probabilistic, and hence supplies no explicit construction of the corresponding robust games. In the next section we describe explicit constructions of leader-election games and coin-flipping games that are immune against linear-sized coalitions.

3. Explicit construction. In this section we show how to explicitly construct coin-flipping games where the influence of any set which is smaller than some linear threshold is bounded away from 1. We use the idea put forth by Bracha [7] in the Byzantine context of forming virtual players from committees of actual players. Say that a committee is good if it has a certain ratio of good players to bad players. The advantage of an assignment to committees is that the ratio of good committees to bad committees can be much better than the ratio of good players to bad players.

Recall that baton passing is the game analyzed by Saks [12] and Ajtai and Linial [1], where a leader is chosen by passing a baton among the players. Initially the baton is held by some arbitrary player and each player that receives the baton picks a player that has not been selected so far and gives him the baton.

In our scheme, the committees formed play baton passing. When a committee gets the baton it elects a leader (recursively) and decides on the next committee to get the baton. The leader of the last committee to hold the baton is the global leader.

The advantage our game has over baton passing is that the bad players do not know in advance which committees will elect good leaders and which bad leaders. Thus, though the percentage of bad leaders is high, a bad leader does not necessarily choose a committee that elects a good leader (which is the optimal strategy in baton passing).

The committees are assigned using an affine plane: each player corresponds to a point in the plane and a committee is a line.

In the next subsection we analyze the variant of the baton-passing game that is relevant to our scheme: a good player might turn bad when he receives the baton. (This corresponds to the case that a good committee elects a bad leader.) In §3.2 we discuss the properties of the assignment to committees by affine planes. Finally in §3.3 we analyze the resulting construction.

3.1. Faulty baton passing. In this subsection we analyze the baton-passing game when even the good players have a certain probability ϵ of becoming faulty. We call this variant the faulty baton-passing game. The game itself is identical to the usual baton passing, and only the behavior of the good players differs.

In the regular baton-passing game the best strategy for the bad participants is to select a good player to receive the baton. By the moment reflection argument [1] (or by induction as in [12]) this is the best strategy in the faulty baton-passing game as well. Thus we can assume that whenever a bad participant has the baton he selects a good participant to receive it, and a good participant that becomes faulty also selects a good participant to receive the baton.

We would like to find bounds on the function $f(s,t) = f_\epsilon(s,t)$ defined as follows. $f(s,t)$ is the probability of the baton to end at a bad player (and *not* at a good one that becomes faulty), starting from a good player when there are s (as yet unselected) good players, t (as yet unselected) bad players, and a good player has probability ϵ of becoming faulty when he gets the baton. (It is important that it not be known in advance whether a good player would become faulty when he will receive the baton.)

We assume $0 \leq \epsilon < \frac{1}{4}$. Clearly, $f(0,t) = 1\,\forall t \geq 1$ and $f(s,0) = 0\,\forall s \geq 0$. From the bad players strategy,

$$(6) \qquad \forall s,t, \geq 1 \quad f(s,t) = \frac{s+\epsilon t}{s+t} \cdot f(s-1,t) + \frac{t-\epsilon t}{s+t} \cdot f(s-1,t-1).$$

LEMMA 3.1. *For all* $s,t \geq 0$, $0 \leq \epsilon < \frac{1}{4}$,

$$f(s,t) \leq 8 \cdot \frac{(t \log_2(t+1))^{\frac{1}{1-4\epsilon}}}{(s+1)^{1-\epsilon}}. \tag{7}$$

Proof. We apply induction on $s+t$. The assertion holds for ($s=0, t\geq 1$) and for ($t=0$, $s \geq 0$). For $t=1$ and $s \geq 1$ from (6) we have

$$f(s,1) = \frac{s+\epsilon}{s+1} \cdot f(s-1,1) = \frac{s+\epsilon}{s+1} \cdot \frac{s-1+\epsilon}{s} \cdots \frac{1+\epsilon}{2} \cdot f(0,1)$$
$$\leq e^{-(1-\epsilon)(\frac{1}{s+1}+\frac{1}{s}+\ldots+\frac{1}{2})} \leq e^{-(1-\epsilon)\int_2^{s+2} \frac{1}{x}dx} \leq \frac{2^{1-\epsilon}}{(s+1)^{1-\epsilon}}.$$

Assume now that $t \geq 2$, $s \geq 1$. Then by (6) and the induction hypothesis it suffices to show that

$$\frac{s+\epsilon t}{s+t} \cdot \frac{(t\ln(t+1))^{\frac{1}{1-4\epsilon}}}{s^{1-\epsilon}} + \frac{t-\epsilon t}{s+t} \cdot \frac{((t-1)\ln t)^{\frac{1}{1-4\epsilon}}}{s^{1-\epsilon}} \leq \frac{(t\ln(t+1))^{\frac{1}{1-4\epsilon}}}{(s+1)^{1-\epsilon}},$$

i.e., that

$$\frac{t-\epsilon t}{(s+t)s^{1-\epsilon}} \cdot ((t-1)\ln t)^{\frac{1}{1-4\epsilon}} \leq (t\ln(t+1))^{\frac{1}{1-4\epsilon}} \left(\frac{1}{(s+1)^{1-\epsilon}} - \frac{s+\epsilon t}{(s+t)s^{1-\epsilon}}\right).$$

Multiplying by $(s+1)^{1-\epsilon}s^{1-\epsilon}(s+t)$ we conclude that it suffices to prove that

$$\begin{aligned} &(t-\epsilon t)(s+1)^{1-\epsilon} \cdot ((t-1)\ln t)^{\frac{1}{1-4\epsilon}} \\ &\leq (t\ln(t+1))^{\frac{1}{1-4\epsilon}}(s^{1-\epsilon}(s+t) - (s+\epsilon t)(s+1)^{1-\epsilon}). \end{aligned} \tag{8}$$

Notice that

$$s^{1-\epsilon}(s+t) - (s+\epsilon t)(s+1)^{1-\epsilon} = s^{1-\epsilon}\left(s+t-\left(1+\frac{1}{s}\right)^{1-\epsilon}(s+\epsilon t)\right)$$
$$\geq s^{1-\epsilon}\left(s+t-\left(1+\frac{1}{s}\right)(s+\epsilon t)\right) = s^{-\epsilon}(s(t-1)-\epsilon st-\epsilon t).$$

Since $t \geq 2$, the right-hand side is at least $s^{-\epsilon}(s(t-1)-4\epsilon s(t-1)) > 0$, and thus by (8) it suffices to show that

$$\frac{(t-\epsilon t)(s+1)^{1-\epsilon}s^{\epsilon}}{s(t-1)-\epsilon st-\epsilon t} \leq \left(\frac{t\ln(t+1)}{(t-1)\ln t}\right)^{\frac{1}{1-4\epsilon}}. \tag{9}$$

But the left-hand side of (9) is at most

$$\frac{(t-\epsilon t)(s+1)}{s(t-1)-\epsilon st-\epsilon t} = \frac{st-\epsilon st+t-\epsilon t}{st-\epsilon st-s-\epsilon t} = 1 + \frac{s+t}{st-\epsilon st-s-\epsilon t} \tag{10}$$
$$= 1 + \frac{s+t}{s(t-1)-\epsilon st-\epsilon t} \leq 1 + \frac{s+t}{s(t-1)-4\epsilon s(t-1)}$$
$$= 1 + \frac{s+t}{s(t-1)(1-4\epsilon)} \leq 1 + \frac{1}{(t-1)(1-4\epsilon)} + \frac{2}{s(1-4\epsilon)}.$$

Since $\ln\left(1+\frac{1}{t}\right) \geq \frac{1}{2t}$ for all $t \geq 2$ we have

$$\begin{aligned}(11)\qquad \frac{t}{t-1}\cdot\frac{\ln(t+1)}{\ln t} &= \left(1+\frac{1}{t-1}\right)\left(1+\frac{\ln\left(1+\frac{1}{t}\right)}{\ln t}\right)\\ &\geq \left(1+\frac{1}{t-1}\right)\left(1+\frac{1}{2t\ln t}\right) \geq 1+\frac{1}{t-1}+\frac{1}{2t\ln t}.\end{aligned}$$

By (9), (10), and (11) it suffices to show that

$$\left(1+\frac{1}{t-1}+\frac{1}{2t\ln t}\right)^{\frac{1}{1-4\epsilon}} \geq 1+\frac{1}{(t-1)(1-4\epsilon)}+\frac{2}{s(1-4\epsilon)}.$$

By Bernoulli's inequality,

$$\left(1+\frac{1}{t-1}+\frac{1}{2t\ln t}\right)^{\frac{1}{1-4\epsilon}} \geq 1+\frac{1}{(t-1)(1-4\epsilon)}+\frac{1}{2t\ln t(1-4\epsilon)}.$$

Therefore, it suffices to check that

$$1+\frac{1}{(t-1)(1-4\epsilon)}+\frac{1}{2t\ln t(1-4\epsilon)} \geq 1+\frac{1}{(t-1)(1-4\epsilon)}+\frac{2}{s(1-4\epsilon)},$$

i.e., that

$$(12)\qquad s \geq 4t\ln t.$$

But for s, t satisfying $s \geq 1$ that violate (12) we have

$$(s+1)^{1-\epsilon} \leq s+1 \leq 2s \leq 8t\ln t \leq 8\,(t\log_2 t)^{\frac{1}{1-4\epsilon}},$$

and thus for this case (7) is trivial (as its right-hand side is greater than 1), completing the proof. □

3.2. Amplification via affine planes. In this subsection we describe a pseudorandom property of projective planes which is applied in [2], [3] and show how affine planes have a similar property. Affine planes better fit our purposes here. The property we need can be proved by an eigenvalue argument; here we present a more direct proof, as in [3].

LEMMA 3.2 [2], [3]. *Let $\mathcal{P} = (P, L)$ be a projective plane of order p with a set P of $n = p^2+p+1$ points and a set L of n lines. If $A \subset P$, $|A| = \epsilon n$, then*

$$\sum_{\ell\in L}\left(|\ell\bigcap A| - \epsilon(p+1)\right)^2 = \epsilon(1-\epsilon)p\cdot n.$$

Proof. Clearly

$$(13)\qquad \sum_{\ell\in L}|\ell\bigcap A| = (p+1)|A| = (p+1)\epsilon n.$$

Also by double counting the number of ordered triples (ℓ, a, b) where $\ell \in L$, $a, b \in A$, $a \neq b$ we get

$$|A|(|A|-1) = \epsilon n(\epsilon n-1) = \sum_{\ell\in L}|\ell\bigcap A|(|\ell\bigcap A|-1) = \sum_{\ell\in L}|\ell\bigcap A|^2 - \sum_{\ell\in L}|\ell\bigcap A|.$$

Thus

$$\sum_{\ell\in L}|\ell\bigcap A|^2 = \epsilon n(\epsilon n-1)+(p+1)\epsilon n = \epsilon n(\epsilon n+p). \tag{14}$$

The two equations (13), (14) enable us to compute for each polynomial $q(x)$ of degree at most two the sum $\sum_{\ell\in L} q(|\ell\bigcap A|)$. In particular,

$$\begin{aligned}\sum_{\ell\in L}(|\ell\bigcap A|-\epsilon(p+1))^2 &= \sum_{\ell\in L}(|\ell\bigcap A|^2-2\epsilon(p+1)|\ell\bigcap A|+\epsilon^2(p+1)^2)\\ &= \epsilon n(\epsilon n+p)-2\epsilon(p+1)(p+1)\epsilon n+n\epsilon^2(p+1)^2\\ &= \epsilon n(\epsilon n+p-\epsilon(p+1)^2)=\epsilon n(p-\epsilon p)=\epsilon(1-\epsilon)pn,\end{aligned}$$

completing the proof. □

COROLLARY 3.3. *Let $\mathcal{A}=(\bar P,\bar L)$ be an affine plane of order p, obtained by deleting a line from the projective plane of that order. Put $m=p^2=|\bar P|=|\bar L|$ and recall that each $\ell\in\bar L$ has p points. Suppose that $A\subset\bar P$, $|A|=\epsilon m$, and suppose that $\delta>0$. Then*

$$\left|\left\{\ell\in\bar L:\left|\ell\bigcap A\right|\ge(\epsilon+\delta)\cdot p\right\}\right|\le\frac{\epsilon(p^2+p+1)}{\delta^2 p}\le\frac{\epsilon}{\delta^2}(\sqrt{m}+2).$$

Proof. Let $x=|\{\ell\in\bar L:|\ell\bigcap A|\ge(\epsilon+\delta)\cdot p\}|$. Embed $\mathcal{A}$ in the projective plane or order p, $\mathcal{P}=(P,L)$. Observe that $|A|/(p^2+p+1)=\epsilon p^2/(p^2+p+1)$, and hence $(|A|/(p^2+p+1))(p+1)=\epsilon p^2(p+1)/(p^2+p+1)\le\epsilon p$. By Lemma 3.2,

$$\sum_{\ell\in\bar L}\left(|\ell\bigcap A|-\frac{|A|}{p^2+p+1}(p+1)\right)^2\le\epsilon p(p^2+p+1).$$

Each line among the x defined above contributes to the left-hand side at least δ^2p^2. Thus $x\delta^2p^2\le\epsilon p(p^2+p+1)$, implying the desired result. □

Remark. For every $p=2^k$ there is an affine plane of order p. Our algorithm uses the planes of order 2^{2^k}. Note that the number of points in a plane of order 2^{2^k} is equal to the number of points in one line of a plane of order $2^{2^{k+1}}$. This is used for recursive application of the algorithm.

Remark. The first author has suggested previously using projective planes as a construction meeting some of the requirements of [7]. (See [9] for details.) As we shall see, unlike the Byzantine case, for our purposes it is not essential that the size of the committees be small (logarithmic in n).

Remark. The construction is an instance of graphs called *dispersers* that have many other applications. (See [8] for an extensive survey of constructions and applications.) There are several other constructions of dispersers that can be used for our purposes.

3.3. The construction. We are now ready to present the construction in detail. We can assume without loss of generality that n is of the form 2^{2^j}: otherwise, let $n'=2^{2^{\lceil\log\log n\rceil}}$ and make each of the n participants play the role of $\lfloor\frac{n'}{n}\rfloor$ or $\lceil\frac{n'}{n}\rceil$ in a game of n' participants. The ratio of bad players has not increased by more than $\frac{1}{n}$.

The scheme is as follows: form committees by treating each player a as a point $a\in\bar P$ in the affine plane $\mathcal{A}=(\bar P,\bar L)$ of order $2^{2^{j-1}}$. Each committee ℓ corresponds to a line $\ell\in\bar L$, i.e., a player a is in committee ℓ if and only if $a\in\ell$.

```
• Set $m \leftarrow$ threshold.
• If $n \le m$, then choose leader by baton passing. Otherwise,
    (1) Construct committees via affine planes.
    (2) $\ell \leftarrow$ first committee.
    (3) Repeat
        (a) Let $\ell$ choose a leader recursively;
        (b) Let the leader of $\ell$ choose a committee $\ell'$ as yet unselected;
        (c) $\ell \leftarrow \ell'$;
until there are no unselected committees.
    (4) The leader of the last chosen committee $\ell$ is crowned as the leader of all players.
```

The value of threshold is a function of ϵ. We let it be the smallest number of the form 2^{2^r}, where r is an integer, such that $2^{2^r} \geq \left(\frac{1}{\epsilon}\right)^{100}$. (Note that for this choice $\left(\frac{1}{\epsilon}\right)^{100} \leq m \leq \left(\frac{1}{\epsilon}\right)^{200}$.)

Observe that the total amount of work done by each player is polynomial in n, since at each stage of the recursion every two players are together in exactly one committee.

The next theorem implies that no set smaller than some linear threshold can control the leadership.

THEOREM 3.4. *There are $\mu, c > 0$ such that $\forall 0 < \epsilon < \mu$ the protocol specified above with an appropriately chosen threshold (as a function of ϵ) is $c\epsilon$ robust against ϵn cheaters.*

To prove this theorem we need the following two lemmas.

LEMMA 3.5. *Suppose that our procedure has the property that with $\sqrt{n} = 2^{2^k}$ participants, for every set of $(\epsilon + \delta)\sqrt{n}$ bad participants, the probability that a bad leader is elected is at most $f = f(\epsilon+\delta)$. Then for $n = 2^{2^{k+1}}$ participants, for every set of ϵn bad participants the probability that one of the bad participants is elected is at most*

$$\frac{8(t\log_2(t+1))^{\frac{1}{1-4f}}}{(s+1)^{1-f}} + f, \tag{15}$$

where $s+t+1 = n$ and $t = \lfloor \frac{\epsilon}{\delta^2}(\sqrt{n}+2) \rfloor$.

Proof. Call a committee good if it has at most $(\epsilon + \delta)\sqrt{n}$ bad participants and bad otherwise. By Corollary 3.3 there are at most $t = \frac{\epsilon}{\delta^2}(\sqrt{n}+2)$ bad committees.

To compute a lower bound on the probability that the last committee is good we can use the analysis of Lemma 3.1 of the faulty baton-passing game: assume that a bad committee is completely bad and that a good committee has probability f to elect a bad leader (i.e., to turn bad). These assumptions clearly only increase the probability of ending in a bad committee. Thus by Lemma 3.1 the probability of ending in a bad committee is at most

$$\frac{8(t\log_2(t+1))^{\frac{1}{1-4f}}}{(s+1)^{1-f}}.$$

If we end in a good committee, then by the hypothesis the conditional probability to end in a bad leader is at most f. Hence the result. □

LEMMA 3.6. *There are $\mu, c > 0$ such that $\forall 0 \leq \epsilon \leq \mu$, $\forall 0 \leq k$ when the protocol is executed with $M_k = m^{2^k}$ players and threshold m chosen as the smallest number of the form 2^{2^r} which is bigger than $\left(\frac{1}{\epsilon}\right)^{100}$, for any set of at most $\epsilon_k \cdot M_k$ bad players, p_k; the probability that a bad leader is elected is at most f_k. Here $\epsilon_k = \epsilon + \sum_{i=k}^{\infty}\left(1/m^{0.01}\right)^{2^i}$ and $f_k = c\epsilon + \sum_{i=1}^{k}\left(1/m^{0.03}\right)^{2^i}$.*

Proof. Apply induction on k. In case $k = 0$ the protocol degenerates to (nonfaulty) baton passing with ϵ_0 bad players. $\epsilon_0 = \epsilon + \sum_{i=0}^{j} \left(1/m^{0.01}\right)^{2^i} \leq 3\epsilon$. Choose μ to be small enough so that $\mu \leq 1/(1800 \log(1/\mu))$. Thus, from the analysis of baton passing in [1], there exists a positive c such that if $3\epsilon m \leq m/3 \log m$, then the probability of electing a faulty player is at most $c \cdot \epsilon = f_0$. Since μ satisfies $\mu \leq 1/(1800 \log(1/\mu))$, the condition above holds, by the choice of m.

Assume that the statement of the lemma holds up to k; we will show that it holds for $k+1$. Given M_{k+1} players with at most ϵ_{k+1} fraction of bad ones, define $\delta = \epsilon_k - \epsilon_{k+1} = (1/M_k)^{0.01}$. By the induction hypothesis and the previous lemma, the probability of electing a bad leader when executing the protocol with M_{k+1} players of whom at most ϵ_{k+1} fraction are bad is at most

$$p_k \leq f_k + \frac{8(t \log_2(t+1))^{\frac{1}{1-4f_k}}}{(s+1)^{1-f_k}},$$

where $s + t + 1 = M_{k+1}$ and $t = \lfloor \frac{\epsilon_{k+1}}{\delta^2} (\sqrt{M_{k+1}} + 2) \rfloor$, where δ is as above. Therefore,

$$t = \left\lfloor \frac{\epsilon_{k+1}}{\delta^2} \left(\sqrt{M_{k+1}} + 2\right) \right\rfloor \leq \epsilon_{k+1} \left(M_{k+1}\right)^{0.01} \left(\sqrt{M_{k+1}} + 2\right) \leq \left(M_{k+1}\right)^{\frac{2}{3}}.$$

Note that $f_k = c\epsilon + \sum_{i=0}^{k} (1/m^{0.03})^{2^i} \leq 3\epsilon c$. We choose μ small enough so that $3\epsilon c \leq 3\mu c < \frac{1}{100}$. Thus we get that

$$p_k \leq f_k + \frac{8(t \log_2(t+1))^{\frac{1}{1-4f_k}}}{(s+1)^{1-f_k}} \leq f_k + \frac{16((M_{k+1})^{\frac{2}{3}} \log M_{k+1})^{\frac{1}{1-4f_{k+1}}}}{(M_{k+1})^{1-f_{k+1}}}$$

$$\leq f_k + \frac{16((M_{k+1})^{\frac{2}{3}} \log M_{k+1})^{\frac{100}{96}}}{(M_{k+1})^{0.99}} \leq f_k + \left(\frac{1}{M_{k+1}}\right)^{0.03} = f_{k+1}. \qquad \square$$

Theorem 3.4 now follows, since by the last lemma, p_j, the probability that a bad leader is elected when the protocol is executed with $n = m^{2^j}$ players,

$$\epsilon_j = \epsilon + \sum_{i=j}^{\infty} \left(\frac{1}{m^{0.01}}\right)^{2^i}$$

fraction of whom are bad, is at most $f_j \leq 3c\epsilon$. Thus this probability is certainly at most $3c\epsilon$ if the fraction of bad players is only ϵ. $\square$

Observe that by adjusting the constant c in this theorem so that it would be bigger than $1/\mu$ we can allow ϵ to be any positive real smaller than 1. We thus have the following.

COROLLARY 3.7. *There is a positive constant* c *such that for every* $0 < \epsilon < 1$ *our protocol with an appropriately chosen threshold* (*as a function of* ϵ) *is* $c\epsilon$ *robust against* ϵn *cheaters.*

Remark. We chose to end the recursion by the baton-passing game. Alternatively, we could have ended when the size of the committee is small enough, say $\log \log \log n$: for each possible protocol for that size of the type considered in §2 and of depth, say, $100 \log \log \log n$ an equal number of committees can be allocated that will elect its leader accordingly. By the analysis of §2 most protocols are good, and thus most good committees will elect a good leader.

Remark. As already mentioned we suspect that for every $\epsilon < \frac{1}{2}$ there is a $\delta < 1$ such that for every n there are coin-flipping and leader-election games that are δ-robust against ϵn cheaters. (The results of §2 show that for every $\epsilon < \frac{1}{3}$ there is such a δ.) It is obvious that the validity of the above statement for leader-election games implies its validity for coin-flipping games. The converse is not obvious, but is true. Combining our technique with some of the ideas of [13] concerning slightly random sources, we can prove the equivalence of these two statements. The details will appear somewhere else.

Note added in proof. R. B. Boppana and B. O. Narayanan (*The biased coin problem*, to appear) have recently extended the main result of [4]. Combining their technique with our method here it is possible to settle the question mentioned in the last remark and show that indeed for every $\epsilon < \frac{1}{2}$ there is a $\delta < 1$ such that for every n there are coin-flipping and leader-election games that are δ-robust against ϵn cheaters.

REFERENCES

[1] M. Ajtai and N. Linial, *The influence of large coalitions*, IBM Res. Report 7133 (67380), Almaden, CA, November 1989.
[2] N. Alon, *Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory*, Combinatorica, 6 (1986), pp. 207–219.
[3] N. Alon and Z. Füredi, *Legitimate colorings of projective planes*, Graphs Combin., 5 (1989), pp. 95–106.
[4] N. Alon and M. O. Rabin, *Biased coins and randomized algorithms*, in Advances in Computing Research, Silvio Micali, ed., Vol. 5, JAI Press, Greenwich, CT, 1989, pp. 499–507.
[5] M. Ben-Or and N. Linial, *Collective coin flipping, robust voting schemes and minima of Banzhaf values*, Proc. 26th IEEE Symposium on Foundations of Computer Science, Washington, D.C., 1985, pp. 408–416.
[6] M. Ben-Or, N. Linial, and M. Saks, *Collective coin-flipping and other models of imperfect information*, Coll. Math. Soc. János Bolyai, 52 (1987), pp. 75–112.
[7] G. Bracha, *An $O(\log n)$ expected rounds randomized Byzantine generals protocol*, J. Assoc. Comput. Mach., 34 (1987), pp. 910–920.
[8] A. Cohen and A. Wigderson, *Multigraph amplification*, to appear.
[9] B. Chor and C. Dwork, *Randomization in Byzantine Agreement*, in Advances in Computing Research, Silvio Micali, ed., Vol. 5, JAI Press, Greenwich, CT, 1989.
[10] R. L. Graham and A. C. Yao, *On the improbability of reaching Byzantine agreements*, Proc. of the 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 467–478.
[11] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on boolean functions*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, Washington, D.C., pp. 68–80.
[12] M. Saks, *A robust non-cryptographic protocol for collective coin-flipping*, SIAM J. Discrete Math., 2 (1989), pp. 240–244.
[13] U. V. Vazirani and V. V. Vazirani, *Random polynomial time is equal to slightly random polynomial time*, Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, Washington, D.C., pp. 417–428.

SIAM J. COMPUT.
Vol. 22, No. 2, pp. 418–429, April 1993

ON THE ZONE THEOREM FOR HYPERPLANE ARRANGEMENTS*

HERBERT EDELSBRUNNER†, RAIMUND SEIDEL‡, AND MICHA SHARIR§

Abstract. The zone theorem for an arrangement of n hyperplanes in d-dimensional real space says that the total number of faces bounding the cells intersected by another hyperplane is $O(n^{d-1})$. This result is the basis of a time-optimal incremental algorithm that constructs a hyperplane arrangement and has a host of other algorithmic and combinatorial applications. Unfortunately, the original proof of the zone theorem, for $d \geq 3$, turned out to contain a serious and irreparable error. This paper presents a new proof of the theorem. The proof is based on an inductive argument, which also applies in the case of pseudohyperplane arrangements. The fallacies of the old proof along with some ways of partially saving that approach are briefly discussed.

Key words. discrete and computational geometry, arrangements, hyperplanes, zones, counting faces, induction, sweep

AMS(MOS) subject classification. 52B30

1. Introduction. A set H of n hyperplanes in d-dimensional space $\mathbf{R}^d$ decomposes $\mathbf{R}^d$ into open cells of dimension d (also called d-faces) and into relatively open faces of dimension k between 0 and $d-1$. These cells and faces define a cell complex which is commonly known as the *arrangement* $\mathcal{A}(H)$ of H. We define the *complexity of a cell in* $\mathcal{A}(H)$ to be the number of faces that are contained in the closure of the cell.

For a hyperplane b (not necessarily in H) the *zone of* b is defined to be the set of all cells in $\mathcal{A}(H)$ that intersect b. Define the *complexity of a zone* to be the sum of the complexities of the cells in the zone. A fundamental result on hyperplane arrangements is presented in the following theorem.

ZONE THEOREM. *Any zone in any arrangement of n hyperplanes in $\mathbf{R}^d$ has complexity $O(n^{d-1})$.*

Various algorithmic and combinatorial applications of this theorem appear throughout the computational and combinatorial geometry literature [6]. For the case $d = 2$ a number of different and fairly straightforward proofs are known, following paradigms such as induction [5]; sweep [3], [11]; tree construction [8]; and Davenport–Schinzel sequences [9]. Only the sweep proof was extended to three and higher dimensions. However, this generalization turned out to be too sweeping. The authors of this paper discovered an irreparable error in that proof, which left the zone theorem unproven for dimensions $d > 2$.

This paper presents a new proof of the general zone theorem. It is based on a relatively straightforward inductive argument whose simplicity fosters confidence that this time the proof is actually correct. The new proof does not exploit the "straightness" of hyperplanes and thus it applies also to arrangements of pseudohyperplanes [4]. The validity of the zone theorem for such arrangements had been considered an open question.

*Received by the editors January 28, 1991; accepted for publication (in revised form) January 10, 1992.

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The research of this author was supported by the National Science Foundation under grant CCR-89-21421.

‡Department of Electrical Engineering and Computer Science, University of California, Berkeley, California 94720. The research of this author was supported by National Science Foundation Presidential Young Investigator grant CCR-90-58440.

§School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and the Courant Institute of Mathematical Sciences, New York University, New York 10012. The research of this author has been supported by Office of Naval Research grant N00014-90-J-1284, by National Science Foundation grant CCR-89-01484, and by grants from the United States–Israeli Binational Science Foundation, the German–Israel Foundation for Scientific Research and Development, and the Fund for Basic Research of the Israeli Academy of Sciences.

Section 2 contains the new proof. Explanations of why the old sweep-based proof fails and of how it can be partially saved are presented in the appendix.

A few bibliographical remarks: After the incorrectness of the old sweep proof of the zone theorem was announced, we soon learned of two partial proofs of the theorem, one by Houle [13] dating back to 1987, and the other by Matoušek [15]. They showed that the zone theorem is correct if the complexity of a cell is defined to be just the number of its facets, i.e., the number of $(d-1)$-dimensional faces contained in the cell's boundary. In the terminology of the following section, they showed that $z_1(n,d) = O(n^{d-1})$. Interestingly, Houle and Matoušek's proofs methods are virtually identical, and in essence are also the same as the saved version of the sweep proof presented in the appendix. Attempts to adapt this proof method to the general zone theorem have failed so far.

2. The new proof. For a d-polyhedron P let $g_k(P)$ denote the number of faces of P of *codimension* k (i.e., dimension $d-k$). For a hyperplane b and a set of hyperplanes H in $\mathbf{R}^d$, let $Zone(b;H)$ denote the set of cells in the arrangement $\mathcal{A}(H)$ that intersect b, and for $0 \le k \le d$ let $z_k(b;H)$ denote $\sum_{C\in Zone(b;H)} g_k(\bar{C})$, where $\bar{C}$ denotes the topological closure of C. Finally, for $n > 0$, $d > 0$, and $0 \le k \le d$, let $z_k(n,d)$ denote the maximum of $z_k(b;H)$ over all hyperplanes b and all sets H of n hyperplanes in $\mathbf{R}^d$. Our goal is to prove the following.

THEOREM 2.1. *$z_k(n,d) = O(n^{d-1})$ for each $d > 0$ and $0 \le k \le d$. In particular, for all $n > 0$, $d > 0$, and $0 \le k < d$, we have*

$$z_k(n,d) \le c_k \binom{d-1}{k}\binom{n}{d-1} + 2^k \sum_{k\le j<d-1} \binom{j}{k}\binom{n}{j},$$

where $c_0 = 1$, and $c_k = \frac{3}{4}(6^k + 2^k)$ for $k > 0$.

As will be explained in §3, the constant can be slightly improved to $c_k = \frac{2}{3}6^k + \frac{3}{4}2^k$.

First note that, for any fixed k, $z_k(b;H)$ achieves its maximum when b and H are in generic position, i.e., every k hyperplanes in $H \cup \{b\}$ intersect in a $(d-k)$-flat, for $1 < k \le d+1$. This can be proved using a standard perturbation argument: translating b slightly can only enlarge $Zone(b;H)$ and displacing a hyperplane of H by a small amount can only increase the complexities of the cells in $Zone(b;H)$ through vertex truncation or the actions dual to vertex pulling or pushing (see [12, pp. 78–83]).

Next note that Theorem 2.1 does not state an explicit upper bound for the number of vertices in a zone. However, it is easily seen that in the generic case $z_d(b;H) \le \frac{2}{d} z_{d-1}(b;H)$ holds, since in the generic (i.e., simple) case every vertex of a d-polyhedron P is incident to d edges, and each edge of P is incident to at most two vertices. Therefore we obtain from Theorem 2.1 the bound

$$z_d(n,d) \le \frac{2}{d} z_{d-1}(n,d) \le \frac{2}{d} c_{d-1} \binom{n}{d-1}.$$

The most important ingredient for our proof of Theorem 2.1 is the following lemma.

LEMMA 2.2. *For all $d > 1$, $0 \le k < d$, and $n > k$ we have*

$$z_k(n,d) \le \frac{n}{n-k}\left(z_k(n-1,d) + z_k(n-1,d-1)\right). \qquad (1)$$

Proof. Let H be a set of n hyperplanes in $\mathbf{R}^d$, and let b be some other hyperplane. Because of the remarks above we assume that the hyperplanes in $H \cup \{b\}$ are in generic position. A face f in $\mathcal{A}(H)$ of codimension k now lies in exactly k hyperplanes of H

and is part of the boundary of 2^k cells of $\mathcal{A}(H)$. More than one of these cells can lie in *Zone*$(b; H)$, and thus the contribution of the face f to $z_k(b; H)$ can be larger than one. In order to have entities that contribute at most one to the count $z_k(b; H)$, we define a *border* of codimension k to be a pair (f, C), where f is a face of codimension k in $\mathcal{A}(H)$ and C is a cell that has f on its boundary. Thus $z_k(b; H)$ counts all borders of codimension k in *Zone*$(b; H)$, i.e., borders (f, C) for which C is in *Zone*$(b; H)$.

Now let h be some hyperplane in H, and let H/h be $\{j \cap h | j \in H\backslash\{h\}\}$. Note that H/h forms a $(d-1)$-dimensional arrangement of $n-1$ "hyperplanes" within h. Consider the expression

$$z_k(b; H\backslash\{h\}) + z_k(b \cap h; H/h).$$

We claim that it is at least as large as the number of borders (f, C) of codimension k in *Zone*$(b; H)$, for which f is not contained in h. Note that every such border is equal to or contained in a border in *Zone*$(b; H\backslash\{h\})$. Our strategy is thus to consider borders in this latter zone, and analyze what happens to them when h is added back to H. So let (f, C) be a border of codimension k in *Zone*$(b; H\backslash\{h\})$.

Case 1. $h \cap C = \emptyset$. The border (f, C) gives rise to exactly one border of codimension k in *Zone*$(b; H)$, namely, itself.

Case 2. $h \cap C \neq \emptyset$ but $h \cap f = \emptyset$. Let h_f be the (open) halfspace bounded by h that contains f, and let $C' = C \cap h_f$. If C' intersects the base hyperplane b, then (f, C) gives rise to one border of codimension k in *Zone*$(b; H)$, namely, (f, C'); otherwise it gives rise to no border in *Zone*$(b; H)$.

Case 3. $h \cap C \neq \emptyset$ and $h \cap f \neq \emptyset$. Let h' and h'' be the two open halfspaces bounded by h and let $C' = C \cap h'$ and $C'' = C \cap h''$. If only one of C' and C'' intersect b (say, C'), then (f, C) gives rise to one border of codimension k in *Zone*$(b; H)$, namely, $(f \cap h', C')$. However, if both C' and C'' intersect b, then (f, C) gives rise to two borders in *Zone*$(b; H)$, namely, $(f \cap h', C')$ and $(f \cap h'', C'')$. But in that case $C \cap h$ is part of *Zone*$(b \cap h; H/h)$ and $(f \cap h, C \cap h)$ is a border of codimension k in *Zone*$(b \cap H; H/h)$. (Note that, in generic position, the border $(f \cap h, C \cap h)$ uniquely determines the border (f, C).)

Since all borders (f, C) of codimension k in *Zone*$(b; H)$ for which f is not contained in h must arise as described in the three cases, it follows that the number of such borders is at most $z_k(b; H\backslash\{h\}) + z_k(b \cap h; H/h)$, as claimed. But from this we can conclude that

$$(n-k) z_k(b; H) \leq \sum_{h \in H} \left(z_k(b; H\backslash\{h\}) + z_k(b \cap h; H/h) \right),$$

since every border (f, C) of codimension k in *Zone*$(b; H)$ is counted in the sum $n-k$ times, once for each hyperplane h that does not contain f. From this last inequality the statement of the lemma follows immediately. $\square$

The recurrence of Lemma 2.2 is a bit unwieldy. However, it becomes more manageable by putting $z_k(n, d) = \binom{n}{k} w_k(n, d)$, for $n \geq k$, which transforms the recurrence (1) into

$$w_k(n, d) \leq w_k(n-1, d) + w_k(n-1, d-1),$$

for all $d > 1$, $0 \leq k < d$, and $n > k$. By iterating this new recurrence on the first summand one obtains

$$w_k(n, d) \leq w_k(k, d) + \sum_{k \leq m < n} w_k(m, d-1), \tag{2}$$

valid again for all $d > 1$, $0 \le k < d$, and $n > k$.

Proving the asymptotic version of Theorem 2.1 is now an easy induction on d. The base case $z_k(n, 2) = O(n)$ for $k = 0, 1, 2$ is proved separately using any one of the proofs offered in [3], [5], [8], [9] (or also in Lemma 2.3).

Now let $d > 2$ and assume inductively that $z_k(m, d-1) = O(m^{d-2})$, for all $k \le d-1$ (where the constant of proportionality depends on k and d). Then $w_k(m, d-1) = O(m^{d-2-k})$, and thus by (2)

$$w_k(n,d) = w_k(k,d) + \sum_{k \le m < n} O(m^{d-2-k}),$$

which implies $w_k(n,d) = O(n^{d-1-k})$, thereby showing that $z_k(n,d) = O(n^{d-1})$, provided $k \le d-2$. When $k = d-1$ this approach yields only an $O(n^{d-1}\log n)$ bound. However, one can now establish the desired $z_k(n,d) = O(n^{d-1})$ for $k = d-1$ and $k = d$ as follows: Euler's relation states that for any cell C in an arrangement the sum $\sum_{0 \le k \le d}(-1)^{d-k} g_k(\overline{C})$ evaluates to 0 or 1, depending on whether C is bounded (see [12, pp. 130–140]). Thus it follows that in $\mathbf{R}^d$ for any set H of $n \ge d$ hyperplanes and any hyperplane b in generic position

$$\sum_{0 \le k \le d} (-1)^{d-k} z_k(b; H) \ge 0.$$

Recalling that $z_d(b;H) \le \frac{2}{d} z_{d-1}(b;H)$, we obtain the relation

$$(3) \qquad \left(1 - \frac{2}{d}\right) z_{d-1}(b;H) \le z_{d-1}(b;H) - z_d(b;H) \le \sum_{0 \le k \le d-2} (-1)^{d-k} z_k(b;H).$$

But as $d > 2$ and we have proven already that $z_k(b;H) \le z_k(n,d) = O(n^{d-1})$ for $0 \le k \le d-2$, and since $z_d(b;H) \le \frac{2}{d} z_{d-1}(b;H)$, relation (3) yields that $z_k(b;H) = O(n^{d-1})$ holds for all k. As this is true for any H and b in generic position and since it suffices to consider only generic position we can conclude that $z_k(n,d) = O(n^{d-1})$ for $0 \le k \le d$. This completes the proof of the asymptotic version of Theorem 2.1.

The proof for the more exact, nonasymptotic version follows the same inductive scheme, except that we will use a slightly different method for dealing with the case $k > d-2$. First we need a few small lemmas. Recall that when arguing about $z_k(n,d)$ we need only to consider simple arrangements with the zone-producing hyperplane b in generic position.

LEMMA 2.3. $z_1(n, 2) \le 6n$.

Proof. Of course this is just the Zone Theorem for arrangements of lines in the plane, and we could refer to a number of different proofs ([3], [5], [8], [9]; in fact, [3] gives a slightly better bound—see §3). For the sake of completeness, however, we include here yet a different proof.

Let H be a set of n lines in the plane, and let b be some other line. Without loss of generality we assume that $H \cup \{b\}$ is in generic position (no three lines intersect, but every two do), and we assume that b is "horizontal." We need to show that the sum of the edges of the cells in $\mathit{Zone}(b;H)$ is at most $6n$.

Since no line in H is parallel to the horizontal line b, it makes sense to talk about the left bounding edges and the right bounding edges of a cell. It suffices to show that the total number of all left bounding edges of the cells in $\mathit{Zone}(b;H)$ is at most $3n$. This is clearly true when H is empty. So let h be the line in H that intersects b furthest to the right. By induction the total number of left bounding edges of the cells in

Zone$(b; H\backslash\{h\})$ is at most $3n-3$. The addition of h to the arrangement formed by $H\backslash\{h\}$ can increase this number at most by 3. $\square$

LEMMA 2.4. *For any set H of n hyperplanes and any hyperplane b in generic position in $\mathbf{R}^d$ with $d > 2$ we have*

$$z_{d-3}(b; H) \geq 2^{d-3}\binom{d-1}{2}\binom{n}{d-1} + 2^{d-3}\binom{d-2}{1}\binom{n}{d-2}.$$

Proof. Assuming generic position of a set H of n hyperplanes and an additional hyperplane b, every 2-face in the $(d-1)$-dimensional arrangement induced by H in b derives from a 3-face in $\mathcal{A}(H)$ that is in the boundary of 2^{d-3} cells of $\mathcal{A}(H)$, all of which are in *Zone*$(b; H)$. It now suffices to observe that in a simple $(d-1)$-dimensional arrangement of n hyperplanes the number of 2-faces is

$$\sum_{0\leq i\leq 2}\binom{d-1-i}{2-i}\binom{n}{d-1-i} \geq \binom{d-1}{2}\binom{n}{d-1} + \binom{d-2}{1}\binom{n}{d-2}$$

(see [6, p. 7]). $\square$

LEMMA 2.5. *For all $d > 2$ and for all $n > d-1$ we have*

$$\binom{d-1}{2} z_{d-1}(n,d) \leq 3(d-2)z_{d-2}(n,d) - 6\cdot 2^{d-3}\left[\binom{d-1}{2}\binom{n}{d-1} + \binom{d-2}{1}\binom{n}{d-2}\right].$$

Proof. Lemma 2.4 implies that it suffices to show the validity of the inequality

$$\binom{d-1}{2} z_{d-1}(b; H) \leq 3(d-2)z_{d-2}(b; H) - 6z_{d-3}(b; H),$$

for any set H of n hyperplanes in $\mathbf{R}^d$ and any hyperplane b in generic position (recall that it suffices to consider only generic position).

Since in a simple arrangement of $n > d-1$ hyperplanes in $\mathbf{R}^d$ every face is pointed (i.e., has a vertex) it suffices to show that for any pointed simple d-polyhedron P with $d > 2$, the inequality $\binom{d-1}{2} f_1 \leq 3(d-2)f_2 - 6f_3$ holds, where $f_i = g_{d-i}$ is the number of i-dimensional faces of P.

Let F_3 be the set of three-dimensional faces of P, and for a 3-face $c \in F_3$ let $e(c)$ denote the number of edges of c and let $s(c)$ denote the number of 2-faces of c. For $j = 1, 2$ let $I_{j,3}$ denote the number of pairs (X, Y) so that Y is a 3-face of P, and X is a j-face of Y.

Because of the simplicity of P every 1-face is contained in $\binom{d-1}{2}$ faces of dimension 3, and thus $I_{1,3} = \binom{d-1}{2} f_1$. Similarly $I_{2,3} = (d-2)f_2$. On the other hand $I_{1,3}$ can also be expressed as $\sum_{c\in F_3} e(c)$, and $I_{2,3}$ as $\sum_{c\in F_3} s(c)$. Euler's relation in any three-dimensional pointed polyhedron with e edges and s facets implies the inequality $e \leq 3s-6$. Hence,

$$\binom{d-1}{2} f_1 = \sum_{c\in F_3} e(c) \leq \sum_{c\in F_3}(3s(c)-6) = 3I_{2,3} - 6f_3 = 3(d-2)f_2 - 6f_3,$$

as asserted. $\square$

We now have everything ready to give a complete inductive proof of our main Theorem 2.1, which claims that for each $d > 0$ the following holds for all $n > 0$ and for each $0 \leq k < d$:

$$z_k(n,d) \le Z_k(n,d) \equiv c_k \binom{d-1}{k}\binom{n}{d-1} + 2^k \sum_{k\le j<d-1} \binom{j}{k}\binom{n}{j}, \tag{4}$$

where $c_0 = 1$, and $c_k = \frac{3}{4}(6^k + 2^k)$ for $k > 0$. For $k > 1$ it is easy to check that $c_k = 6(c_{k-1} - 2^{k-2})$ with $c_1 = 6$.

We first dispose of a few easy cases. The bound (4) is trivially correct when $d = 1$. So assume $d > 1$ and consider $n < k < d$. In this case the bound $Z_k(n,d)$ in (4) evaluates to 0, which is correct since there cannot be any face with codimension k when there are fewer than k hyperplanes.

Finally we can dispose of the case $n = k < d$ since in an arrangement of $k < d$ hyperplanes in generic position there is exactly one face of codimension k and it is in the boundary of 2^k cells, all of which are intersected by the zone hyperplane. Thus $z_k(k,d) = 2^k \le Z_k(k,d)$, as desired. From now on we will thus assume that $d > 1$ and $n > k$.

Applying the binomial product identity $\binom{A}{B}\binom{C}{A} = \binom{C}{B}\binom{C-B}{A-B}$ to each term on the right side of (4), and using the substitution $z_k(n,d) = \binom{n}{k} w_k(n,d)$ as before, we can rewrite (4) equivalently as

$$w_k(n,d) \le W_k(n,d) \equiv c_k \binom{n-k}{d-1-k} + 2^k \sum_{k\le j<d-1} \binom{n-k}{j-k}.$$

We can now prove the bounds (4) by induction on d. For the base case $d = 2$ Lemma 2.3 implies that indeed $z_1(n,2) \le 6n = Z_1(n,d)$; the fact that $z_0(n,d) \le n+1 = Z_0(n,d)$ is trivial.

Now let $d > 2$ and assume inductively that for $0 \le k < d-1$ and for all $m \ge k$ the bounds $z_k(m,d-1) \le Z_k(m,d-1)$ and therefore also the equivalent bounds $w_k(m,d-1) \le W_k(m,d-1)$ hold.

Recall that for $0 \le k < d$, and $n > k$ Lemma 2.2 implies the inequality (2)

$$w_k(n,d) \le w_k(k,d) + \sum_{k\le m<n} w_k(m,d-1).$$

Clearly $w_k(k,d) = z_k(k,d)$ and hence $w_k(k,d) = 2^k$. Thus employing our inductive assumption and exploiting the binomial identities $\binom{A}{0} = 1$ and $\sum_{0\le i<A}\binom{i}{B} = \binom{A}{B+1}$ if $B \ge 0$, we obtain

$$\begin{aligned}
w_k(n,d) &\le 2^k + \sum_{k\le m<n} W_k(m,d-1)\\
&= 2^k + \sum_{k\le m<n}\left[c_k\binom{m-k}{d-2-k} + 2^k \sum_{k\le j<d-2}\binom{m-k}{j-k}\right]\\
&= 2^k + \left[c_k\binom{n-k}{d-1-k} + 2^k \sum_{k\le j<d-2}\binom{n-k}{j+1-k}\right]\\
&= c_k\binom{n-k}{d-1-k} + 2^k \sum_{k\le j<d-1}\binom{n-k}{j-k} = W_k(n,d),
\end{aligned}$$

as desired.

Thus we have established for all $n > k$ the desired $w_k(n,d) \leq W_k(n,d)$ and equivalently $z_k(n,d) \leq Z_k(n,d)$—however, only for $0 \leq k < d-1$. For $k = d-1$ our inductive assumption does not hold.

To prove the desired bound for $k = d-1$ we proceed as follows. We need only consider the case $n > k = d-1$. We just proved that

$$z_{d-2}(n,d) \leq Z_{d-2}(n,d) = c_{d-2}(d-1)\binom{n}{d-1} + 2^{d-2}\binom{n}{d-2}.$$

Plugging this in the inequality

$$z_{d-1}(n,d) \leq \frac{1}{\binom{d-1}{2}}\left[3(d-2)z_{d-2}(n,d) \; -6\cdot 2^{d-3}\binom{d-1}{2}\binom{n}{d-1} \right. \\ \left. -6\cdot 2^{d-3}\binom{d-2}{1}\binom{n}{d-2}\right]$$

of Lemma 2.5 yields the desired

$$z_{d-1}(n,d) \leq \tfrac{1}{\binom{d-1}{2}}\left[3(d-2)c_{d-2}(d-1)\binom{n}{d-1} + 3(d-2)2^{d-2}\binom{n}{d-2} \right. \\ \left. -6\cdot 2^{d-3}\binom{d-1}{2}\binom{n}{d-1} - 6\cdot 2^{d-3}(d-2)\binom{n}{d-2}\right] \\ = 6(c_{d-2} - 2^{d-3})\binom{n}{d-1} = c_{d-1}\binom{n}{d-1} = Z_{d-1}(n,d).$$

This completes the induction on d and the proof of Theorem 2.1.

3. Remarks. First let us point out that our proof of the zone theorem does not exploit the "straightness" of hyperplanes per se, but only the restricted kinds of intersection patterns that are possible amongst hyperplanes. Thus the proof applies equally well to arrangements of pseudohyperplanes which can be modeled combinatorially by oriented matroids (see [4]).

For the two-dimensional case Bern et al. [3] prove the slightly stronger bound of $z_1(n,2) \leq \frac{11}{2}n$, which is tight up to an additive constant in the worst case. Using this bound as an induction basis in our proof yields a slightly better value for the constants c_k, namely, $c_k = \frac{2}{3}6^k + \frac{3}{4}2^k$ for $k > 0$.

The definition of a zone that we use in this paper is slightly different from definitions used before. We define $Zone(b; H)$ to be the set of all cells in the arrangement $\mathcal{A}(H)$ that intersect the hyperplane b, and the cells were defined to be open sets. Previously the zone used to be defined as the set of all cells *whose closure intersects b*. Let us call this set $Czone(b; H)$. Note that in degenerate cases $Czone(b; H)$ can be substantially more extensive than $Zone(b, H)$. Nevertheless the $O(n^{d-1})$ upper bound also applies to the sum of the complexities of the cells in $Czone(b; H)$. This can most easily be seen by observing that $Czone(b; H) = Zone(b^+; H) \cup Zone(b^-; H)$, where b^+ and b^- are two hyperplanes parallel to b, one on each side of b and sufficiently close to b.

Deriving the exact value of $z_k(n,d)$ looks like a fairly challenging problem. For a start, tight lower bounds on $z_k(n,3)$ are desired.

Finally, we remark that the proof techniques of this paper, in particular Lemma 2.2, bear some similarity to the so-called combination lemma techniques; see [7], [10], [16].

It would be interesting to see to what extent it can be generalized and applied to related discrete geometry problems. Recent successful applications have been achieved by Aronov et al. [1], who derive bounds on the sum of squares of cell complexities in a hyperplane arrangement, by Aronov and Sharir [2], who prove bounds on the complexity of the zone of a convex or fixed-degree algebraic hypersurface in a hyperplane arrangement, and by Houle and Tokuyama [14], who give bounds on the complexity of the zone of a flat in a hyperplane arrangement.

4. Appendix: Why the sweep proof fails in the general case and how it can be partially saved. In this appendix we first show what is wrong with the old sweep-based proof of the zone theorem as presented in [11] or [6], and then we show how the sweep approach can be used to still prove the zone theorem in the three-dimensional case, and a weak version of the theorem in the general d-dimensional case, which counts only facets.

Before we get to the details of the old proof let us change the problem slightly: Firstly we rename the base plane b of the previous sections h, and secondly we could the number of faces bounding the cells in $\mathcal{A}(H \cup \{h\})$ that lie on one side of h and have a facet in h. In the generic case (which we can again assume without loss of generality), even if we ignored the faces contained in h, this number, for at least one side of h, would be at least half the complexity of the zone of h in $\mathcal{A}(H)$ as defined in §1.

Choose a coordinate system so that h is the hyperplane $x_d = 0$ and consider the cells above h (in the half-space $x_d > 0$) that have a facet in h. Call these cells and their faces *active*. As above, when we count the active faces we will count a face once for each active cell it bounds. We therefore continue to use the notion of borders, and define 2^{d-k} *k-borders* (f, C) for each k-face f, one for each cell C it bounds. Call a k-border (f, C) *active* if and only if cell C is active.

4.1. The notion of a chain. The basic idea of the sweep proof is to move a hyperplane h_t continuously over the active cells. Think of $t \geq 0$ as the time and define $h_t : x_d = t$. So $h_0 = h$ and h_t moves upwards as t increases. At any point in time t the cross section of $\mathcal{A} = \mathcal{A}(H)$ within h_t is an arrangement defined by n hyperplanes in $\mathbf{R}^{d-1}$, which we denote as $\mathcal{A}_t$.

Let us index the vertices of $\mathcal{A}$ above h as $v_1, v_2, \ldots, v_m$ and define points in time u_i so that $v_i \in h_{u_i}$ for $1 \leq i \leq m$. We assume that $0 = u_0 < u_1 < u_2 < \ldots < u_m < u_{m+1} = \infty$. Unless $t = u_i$ for some i, $\mathcal{A}_t$ is a simple arrangement. If $t = u_i$ then exactly d of the hyperplanes defining $\mathcal{A}_t$ meet in a common point, and otherwise the hyperplanes are in a generic position. Let t and t' be so that $u_{i-1} < t < u_i < t' < u_{i+1}$ for some $1 \leq i \leq m$ and call the transition from $\mathcal{A}_t$ to $\mathcal{A}_{t'}$ an *elementary step.* Ignoring the positional differences of the hyperplanes in $\mathcal{A}_t$ and $\mathcal{A}_{t'}$, the only combinatorial difference between the two arrangements is that a $(d-1)$-simplex in $\mathcal{A}_t$ reverses its orientation in $\mathcal{A}_{t'}$. This is illustrated in Fig. 4.1, which shows three cross sections of an arrangement of three planes in $\mathbf{R}^3$.

Let $h_1, h_2, \ldots, h_d$ be the hyperplanes (in $\mathbf{R}^d$) that intersect at v_i. The faces of $\mathcal{A}$ that intersect h_t are the same as the faces that intersect $h_{t'}$, except for a group of faces that have v_i as their topmost vertex (they intersect h_t but not $h_{t'}$) and another group of faces that have v_i as their bottommost vertex (they intersect $h_{t'}$ but not h_t). For each $1 \leq k \leq d$, we *identify* a k-face f in the first group with a k-face f' in the second group if f and f' span the same k-flat. Similarly, we identify the k-borders (f, C) and (f', C') if f and f' are identified and C and C' lie on the same side of each hyperplane that contains f and f'. The identification effectively defines equivalence classes of faces and borders. We call an equivalence class of k-borders a *k-chain,* for $1 \leq k \leq d$. For example, a

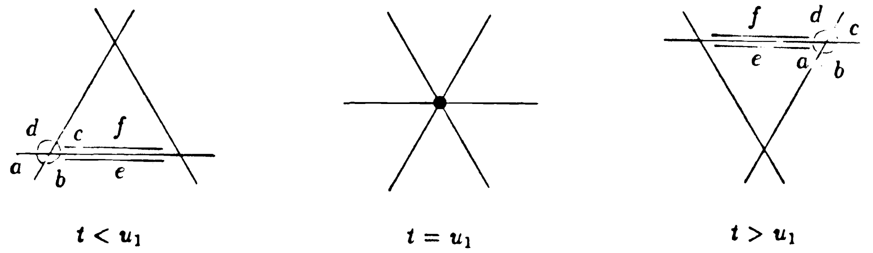


FIG. 4.1. *The triangle defined by three lines (the cross sections of three planes) changes orientation as h_t sweeps through the vertex where the three planes meet.*

1-chain is a sequence of 1-borders (sided edges) on a line, a 2-chain is a sequence of 2-borders in a common 2-flat, and so on and so forth. Figure 4.1 shows the cross sections of four 1-chains (the four-sided versions of a sequence of edges) and of two 2-chains in a three-dimensional arrangement.

4.2. 2-chains do not necessarily die. The sweep proof of the zone theorem hinges on the claim that whenever we sweep through an active vertex there is at least one chain that dies. A chain is said to be *dead* at time t if it contains no further active borders, that is, all borders of the chain that intersect some $h_{t'}$ with $t' > t$ are inactive. Since the number of chains is $O(n^{d-1})$, this claim implies the Zone Theorem. We show below that, unfortunately, this claim is incorrect starting in dimension $d = 3$.

For a vertex v call the cell for which v is the topmost vertex the *cell below* v, and consider the three types of elementary steps shown in Fig. 4.2. In type 1 the 3-chain whose first cell is the cell below v dies. Indeed, every 3-chain has only one active cell, namely, its first cell. In type 2 the 1-chain labeled a dies. This is because all future 1-borders of this 1-chain lie on an edge of a triangular cone disjoint from h and their associated cells lie inside this cone. We now demonstrate that in type 3 the 2-chain labeled ϵ does not die, contrary to the claim in [11].

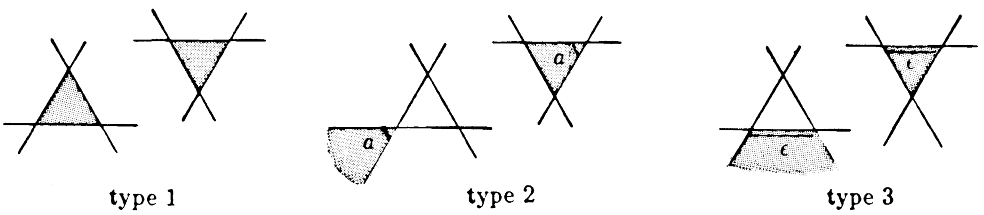


FIG. 4.2. *We distinguish three types of elementary steps associated with a vertex v in a three-dimensional arrangement. For each type the cross section of the active cell is shaded. In type 1 the cell below v is active, in type 2 the cell below v shares an edge with the active cell, and in type 3 the cell below v shares a 2-face with the active cell. Elementary steps where more than one cell bounded by v are active are decomposed into instances of the three basic types.*

The example that we use consists of four planes, h_1, h_2, h_3, h_4. We choose $h_1 : x_1 = 0$, $h_2 : x_2 = 0$, and $h_3 : x_1+x_2+x_3 = 1$. Now choose h_4 so that it meets the line $h \cap h_3$ at a point with negative x_2-coordinate and so that the central triangle (the 2-face that lies in h_3 and in front of h, h_1, and h_2) lies just slightly behind h_4. In Fig. 4.3 only the lines of intersection of h_4 with h and h_3 are shown. Of the three shaded 2-borders, which belong to a 2-chain, the first and the third are active and the middle one is inactive. Indeed, the middle 2-border is within the dead cone defined by h_1, h_2, h_3, but the third 2-border is not. When we sweep through vertex $v = h_1 \cap h_2 \cap h_3$ we have type 3 as shown in Fig. 4.2, but the 2-chain labeled e does not die.

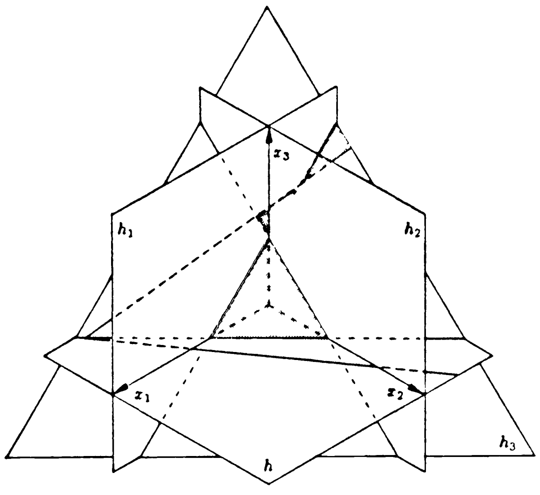


FIG. 4.3. *The shaded* 2*-borders (they face the observer) belong to a* 2*-chain. The first and the third* 2*-borders are active, the middle one is inactive.*

4.3. Fixing the sweep proof in three dimensions. In spite of the fact that in type 3 no chain dies we can still argue that the zone theorem is correct for $d = 3$ using the sweep approach and the type classification illustrated in Fig. 4.2. The reason is that 1-chains still die and 2-chains are linked to 1-chains in elementary steps of the sweep.

At time $t = 0$. $\mathcal{A}_0$ is a two-dimensional arrangement defined by n lines. Because we assume generic position it consists of $\nu_0 = \binom{n}{2}$ vertices, $\epsilon_0 = n^2$ edges, and $\phi_0 = \binom{n}{2} + n + 1$ 2-faces (see, e.g., [6, p. 7]). So we have $4\nu_0$ 1-borders each starting a 1-chain, $2\epsilon_0$ 2-borders each starting a 2-chain, and ϕ_0 cells each starting a 3-chain. All these 1-borders, 2-borders, and cells are active.

Type 1. Recall that in this case the cell below the vertex v that is swept over is active. The corresponding 3-chain dies. So type 1 can occur at most ϕ_0 times. Indeed, it occurs exactly once for each bounded 2-face in $\mathcal{A}_0$ and therefore exactly $\gamma_1 = \phi_0 - 2n = \binom{n}{2} - n{+}1$ times. Each time an elementary step is of type 1 we encounter a new active 0-border, and, respectively, three active 1-borders and 2-borders no longer intersect h_t.

Type 2. Here, the cell below v shares an edge with the active cell. One 1-chain on the corresponding line dies and a new active 2-border is encountered. Since there are only $4\nu_0$ 1-chains, this type can occur only $\gamma_2 \leq 4\nu_0$ times. Besides the new active 2-border we also encounter two new active 1-borders and one new 0-border at v.

Type 3. Here, the cell below v shares a 2-face with the active cell. The number of active 2-borders that intersect h_t decreases by one. Type 3 thus occurs $\gamma_3 \leq 2\epsilon_0 + \gamma_2 - 3\gamma_1 \leq 2n^2 + \binom{n}{2} + 3n - 3$ times because there are $2\epsilon_0$ active 2-borders initially, we get one more at each occurrence of type 2, and $3\gamma_1$ 2-faces disappear in elementary steps of type 1. Notice that we encounter one new active 1-border and one new active 0-border.

We count the active borders when they are encountered by h_t. Initially, we have $4\nu_0$ 0-borders, $2\epsilon_0$ 1-borders, and ϕ_0 2-borders, all active and all in h. We also have $4\nu_0$ 1-chains each starting with an active 1-border, and $2\epsilon_0$ 2-chains each starting with an active 2-border. So we get

$$4\nu_0+\gamma_1+\gamma_2+\gamma_3 \le 4\binom{n}{2}+\binom{n}{2}-n+1+4\binom{n}{2}+2n^2+\binom{n}{2}+3n-3 = 7n^2+O(n)$$

active 0-borders,

$$2\epsilon_0+4\nu_0+2\gamma_2+\gamma_3 \le 2n^2+4\binom{n}{2}+8\binom{n}{2}+2n^2+\binom{n}{2}+3n-3=\frac{21n^2}{2}+O(n)$$

active 1-borders, and

$$\phi_0+2\epsilon_0+\gamma_2 \le \binom{n}{2}+n+1+2n^2+4\binom{n}{2}=\frac{9n^2}{2}+O(n)$$

active 2-borders. This proves the zone theorem for $d=3$.

What kind of upper bounds does this yield for $z_k(n,3)$? Note that here we are counting the borders just on one side of the plane h, but we include the borders contained in h. Thus to get good bounds for $z_k(n,3)$ from the above bounds, we need to subtract the borders contained in h, multiply by two, and subtract the borders that intersect h. Doing this yields the bounds $z_1(n,3)\le 6n^2+O(n)$, $z_2(n,3)\le 15n^2+O(n)$, and $z_3(n,3)\le 10n^2+O(n)$. Note that the coefficients of the n^2 terms agree with the corresponding coefficients of Theorem 2.1 as proved in §2. Thus, with the improved coefficients mentioned in §3, we have better bounds than those derived above.

4.4. The sweep proof for facets. The fix of the sweep proof described in §4.3 does not extend beyond three dimensions. For example, in dimension $d=4$ the sweep links 1-chains with 3-chains and 2-chains with other 2-chains. Since 1-chains still die we can bound the number of active 3-borders, but currently we are not able to bound the number of active k-borders, $k\le 2$, using the sweep approach. Below we show how to bound the number of active $(d-1)$-borders for arbitrary dimension d.

As before, let H be a set of n hyperplanes in generic position and let $h\notin H$ be the hyperplane $x_d=0$. There are $\sum_{i=0}^{d-1}\binom{n}{i}$ active $(d-1)$-borders in h, and initially, there are $2\sum_{i=1}^{d-1} i\binom{n}{i}$ $(d-1)$-chains, each starting with an active $(d-1)$-border. We encounter a new active $(d-1)$-border at an elementary step if and only if it is of the type that the cell below the passed vertex shares (only) an edge with the active cell. However, in this case a 1-chain on the shared edge dies. So this type of elementary step can occur at most $2^{d-1}\binom{n}{d-1}$ times, once for each 1-chain. It follows that there are at most

$$\sum_{i=0}^{d-1}\binom{n}{i}+2\sum_{i=1}^{d-1}\binom{n}{i}+2^{d-1}\binom{n}{d-1}=(2^{d-1}+2d-1)\binom{n}{d-1}+O(n^{d-2})$$

active $(d-1)$-borders.

The bound given here implies the bound $z_1(n,d)\le(2^d+2d-2)\binom{n}{d-1}+O(n^{d-2})$, which in terms of its leading coefficient is substantially worse than the bound for $z_1(n,d)$ given in Theorem 2.1.

REFERENCES

[1] B. Aronov, J. Matoušek, and M. Sharir, *On the sum of squares of cell complexities in hyperplane arrangements*, in Proceedings of the 7th ACM Symposium on Computational Geometry, Association for Computing Machinery, New York, 1991, pp. 307–313.

[2] B. Aronov and M. Sharir, *On the zone of a surface in a hyperplane arrangement*, in Proceedings of the 2nd Workshop on Algorithms and Data Structures, 1991, pp. 13–19; Springer-Verlag, Lecture Notes in Computer Science 519, Berlin, New York, 1991.

[3] M. Bern, D. Eppstein, P. Plassmann, and F. Yao, *Horizon theorems for lines and polygons*, Discrete and Computational Geometry: Papers from the DIMACS Special Year, J. Goodman, R. Pollack and W. Steiger, eds., American Mathematical Society, Providence, RI, to appear.

[4] A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler, *Oriented Matroids*, in preparation.

[5] B. Chazelle, L. J. Guibas, and D. T. Lee, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.

[6] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.

[7] ———, *The upper envelope of piecewise linear functions: Tight bounds on the number of faces*, Discrete Comput. Geom., 4 (1989), pp. 337–343.

[8] H. Edelsbrunner and L. J. Guibas, *Topologically sweeping an arrangement*, J. Comput. System Sci., 38 (1989), pp. 165–194.

[9] H. Edelsbrunner, L. J. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir, *Arrangements of curves in the plane: Topology, combinatorics and algorithms*, in Proceedings of the 15th International Colloquium on Automata, Languages and Programming, 1988, pp. 214–229.

[10] H. Edelsbrunner, L. J. Guibas, and M. Sharir, *The complexity and construction of many faces in an arrangement of lines or of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.

[11] H. Edelsbrunner, J. O'Rourke, and R. Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. Comput., 15 (1986), pp. 341–363.

[12] B. Grünbaum, *Convex Polytopes*, John Wiley & Sons, London, 1967.

[13] M. Houle, *A note on hyperplane arrangements*, University of Tokyo, Tokyo, Japan, 1987, manuscript.

[14] M. Houle and T. Tokuyama, *On zones of flats in hyperplane arrangements*, University of Tokyo, Tokyo, Japan, 1991, manuscript.

[15] J. Matoušek, *A simple proof of the weak zone theorem*, Charles University, Prague, Czechoslovakia, 1990, manuscript.

[16] J. Pach and M. Sharir, *The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates: Combinatorial analysis*, Discrete Comput. Geom., 4 (1989), pp. 291–309.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 431–459, June 1993

LINEAR-PROCESSOR NC ALGORITHMS FOR PLANAR DIRECTED GRAPHS I: STRONGLY CONNECTED COMPONENTS*

MING-YANG KAO†

Abstract. Finding strongly connected components is a fundamental step in many algorithms for directed graphs. In sequential computation this problem has an optimal linear-time algorithm. In parallel computation, however, it remains an open problem whether polylogarithmic time and a *linear* number of processors are sufficient for computing the strongly connected components on a parallel random-access machine. This paper provides the first nontrivial partial solution to the open problem: for a planar directed graph of size n the strongly connected components can be computed deterministically in $O(\log^3 n)$ time with $n/\log n$ processors. The algorithm runs on a parallel random-access machine that allows concurrent reads and concurrent writes in its shared memory and, in case of a write conflict, permits an arbitrary processor to succeed.

Key words. linear-processor NC algorithms, strongly connected components, planar directed graphs, planar orientation, planar topology, edge cutting, vertex expansion, vertex contraction

AMS(MOS) subject classifications. 68Q22, 05C99

1. Introduction.

A directed graph is *strongly connected* if for all vertices u and v there exists a pair of directed paths from u to v and from v to u. A *strongly connected component* in a directed graph is a maximal strongly connected subgraph. The problem of finding strongly connected components is of fundamental importance in graph theory. In sequential computation this problem has optimal linear-time algorithms [2], [9]. In parallel computation on parallel random-access machines (PRAMs) the best current algorithm is based on matrix multiplication and for an n-vertex graph requires $O(\log^2 n)$ time if $n^{2.376}$ processors are used [7], [11], [16].

The total number of operations performed by an algorithm can be estimated from the product of its running time and its processor count. Thus there is a substantial complexity gap between the optimal sequential algorithms and the best current PRAM algorithm for strongly connected components. In view of this gap it is important to ask whether polylogarithmic time and a *linear* number of processors are sufficient for computing strongly connected components. This paper provides the first nontrivial partial solution to this open problem: for a planar directed graph of size n the strongly connected components can be computed deterministically in $O(\log^3 n)$ time with $n/\log n$ processors. The algorithm runs on a PRAM that allows arbitrary concurrent reads and concurrent writes (CRCWs) in its shared memory and, in case of a write conflict, permits an arbitrary processor to succeed.

A natural approach to computing strongly connected components is to iteratively contract directed cycles into single vertices. At the end of this contraction process the strongly connected components become single vertices. Then the strongly connected components are obtained by undoing the contractions and expanding each single vertex back to an original strongly connected component.

For this contraction process to be efficient it is necessary to find directed cycles efficiently. Unfortunately, it is in general very expensive to find directed cycles. In this

*Received by the editors January 24, 1990; accepted for publication (in revised form) February 26, 1992. An outline of this work appeared as one of the two parts of *Local Reorientation, Global Order, and Planar Topology*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Seattle, Washington, May 15–17, 1989, pp. 286–296.

†Department of Computer Science, Duke University, Durham, North Carolina 27706. This research was supported in part by National Science Foundation grants CCR-8909323 and CCR-9101385. Part of this work was done while the author was at the Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

paper the planar strongly connected component algorithm focuses on *cycle* faces; a cycle face in an embedded planar directed graph is one whose boundary forms a directed cycle. Note that given an embedded planar directed graph of size n, the cycle faces can be identified in $O(\log n)$ time with $n/\log n$ processors.

The concentration on cycle faces is coupled with useful observations about *compact* strongly connected components; a compact strongly connected component is one that encloses no other components. Observe that if a compact strongly connected component consists of more than one vertex, then this component contains a cycle face. Also observe that the contraction of a cycle face preserves the compactness of strongly connected components. Therefore, when cycle faces are iteratively contracted, all compact strongly connected components eventually become single vertices. If the strongly connected components do not enclose one another, then this face-contraction process can indeed compute the strongly connected components.

There are three major difficulties associated with the above process of contracting cycle faces. First, contracting cycle faces in general creates new cycle faces. The contraction process as described may take more than a polylogarithmic number of iterations. The number of iterations in the planar strongly connected component algorithm is limited by using an edge-cutting technique to compute and contract large neighborhoods of cycle faces. Edge cutting and vertex contraction are combined with a vertex-expansion technique and an orientation structure of primal and dual planar directed graphs. This combination of techniques has also led to the first linear-processor NC algorithm for computing directed spanning trees in strongly connected planar directed graphs [17].

Second, an embedded planar directed graph generally contains nested strongly connected components. It is possible that a planar directed graph has no cycle faces, but not every strongly connected component is a single vertex. Therefore, the face-contraction process is not enough for contracting all strongly connected components into single vertices. To flatten the nesting of strongly connected components, the component algorithm uses edge cutting and vertex contraction to manipulate the nesting. The manipulation is effected through the interaction between graphic and topological structures of planar directed graphs and is guided by a forestlike construct of the topological nesting of strongly connected components.

Third, although it is easy to see that the contraction of a cycle face preserves the compactness of strongly connected components, the contraction of more complex sets incurs complicated changes in the structure of planar directed graphs. To manage and analyze these changes, the component algorithm works with structures and properties of planar directed graphs that are invariant under vertex contraction.

This paper is organized as follows. Section 2 provides basic definitions of and facts about planar directed graphs, and §3 describes their orientation structures. Section 4 analyzes topological nestings of planar strongly connected components, §§5 and 6 discuss edge cutting and vertex contraction, respectively, and §7 details how to flatten the nesting of strongly connected components. Section 8 gives the planar strongly connected component algorithm, and §9 concludes with open problems.

2. Basics of planar directed graphs.

A *planar* directed graph is one that can be embedded on a plane such that the edges intersect only at common endpoints [4], [5], [14], [23]. An *embedded* planar directed graph is one with a given planar embedding.

Remark. The planar strongly connected component algorithm uses only spherical embeddings, but its analysis uses planar embeddings.

2.1. Canonical graphs. A *bridge* of a directed graph is an edge whose removal increases the number of undirected connected components in that graph.

A *canonical* graph is a planar directed graph that is connected, has at least one vertex, and may have multiple edges but contains no loop edges and no bridges.

Most discussions in this paper focus on canonical graphs for the following reasons. Permitting a planar graph to have multiple edges but no loop edges, except in the analysis of contraction invariants in §6.2, makes the effects and invariants of vertex contraction manageable. The planar strongly connected component algorithm works with planar graphs and their duals. Because bridges are duals of loop edges, they are also removed from planar graphs soon after they appear. Note that removing the bridges of a directed graph does not alter its strongly connected components.

2.2. Faces, boundaries, orientations, and cycle faces. Let G be an embedded canonical graph. If the vertices and edges of G are deleted from its embedding plane, then the plane is divided into disconnected regions. Exactly one of the regions is infinite; all others are finite. Each region is called a *face* of G. The infinite region is called the *external* face; the finite regions are called the *internal* faces.

Let f be a face of G. The *boundary* of f, denoted by $\mathcal{B}(f)$, is the set of edges and vertices surrounding f. Because G is connected and bridgeless, $\mathcal{B}(f)$ is connected and bridgeless. Thus if $\mathcal{B}(f)$ contains edges, it can be arranged into a unique edge-simple undirected cycle by having an observer stay inside f and walk around $\mathcal{B}(f)$ once. This cycle is called the *boundary* cycle of f.

Let d be a boundary edge of f. The *orientation* of d with respect to f is defined as follows:

• Case (1): f is the external face. The edge d is *positive* (respectively, *negative*) with respect to f if it points in the counterclockwise (respectively, clockwise) direction on the boundary cycle of f.

• Case (2): f is an internal face. The edge d is *positive* (respectively, *negative*) with respect to f if it points in the clockwise (respectively, counterclockwise) direction on the boundary cycle of f.

A *cycle* face is one with a directed boundary cycle. A cycle face is *positive* (respectively, *negative*) if its boundary edges are all positive (respectively, negative).

2.3. Subfaces, orientations, and cycle subfaces. Let G be an embedded canonical graph. Let f be a face of G. A *subface* g of f is a vertex-simple undirected subcycle of the boundary cycle of f.

For an edge d of g the *orientation* of d with respect to g is defined to be the same as the orientation of d with respect to f.

A *cycle* subface is one that is a directed cycle. A cycle subface is *positive* (respectively, *negative*) if its edges are all positive (respectively, negative).

A graph with no cycle faces may still have cycle subfaces. The nonexistence of cycle subfaces is crucial for obtaining desired structures of planar graphs. The planar strongly connected component algorithm works with subfaces instead of faces.

2.4. Holes, boundaries, orientations, cycle holes, and subholes. Let G be an embedded canonical graph. Let B be a connected subgraph of G with at least one vertex. If the vertices and edges of B are removed from the embedding plane of G, then the plane is divided into disconnected regions. Exactly one of the regions is infinite; the others are all finite. Each region is called a *hole* of B. The infinite region is called the *external* hole; the finite regions are called the *internal* holes.

The *content* of a hole is the set of vertices and edges in that hole. An *empty* (respectively, *nonempty*) hole is one whose content is empty (respectively, nonempty). Note that an empty hole of B is a face of G.

Let X be a hole of B. The *boundary* of X, denoted by $\mathcal{B}(X)$, is the set of vertices and edges surrounding X. Because B is connected, $\mathcal{B}(X)$ is connected. Thus if $\mathcal{B}(X)$ contains at least one edge, it can be arranged into a unique undirected cycle by having an observer stay inside X and walk around $\mathcal{B}(X)$ exactly once. This cycle is called the *boundary* cycle of X. It may not be edge-simple because $\mathcal{B}(X)$ may not be bridgeless.

Let d be a boundary edge of X. The *orientation* of d with respect to X is defined as follows:

• Case (1): X is the external hole of B. The edge d is *positive* (respectively, *negative*) with respect to X if it points in the counterclockwise (respectively, clockwise) direction on the boundary cycle of X.

• Case (2): X is an internal hole of B. The edge d is *positive* (respectively, *negative*) with respect to X if it points in the clockwise (respectively, counterclockwise) direction on the boundary cycle of X.

A *cycle* hole is one with a directed boundary cycle. A cycle hole is *positive* (respectively, *negative*) if its boundary edges are all positive (respectively, negative).

A *subhole* Z of X is a vertex-simple undirected subcycle of the boundary cycle of X. For an edge d on Z the *orientation* of d with respect to Z is defined to be the same as the orientation of d with respect to X.

2.5. Combinatorial embeddings and data structures. Let G be a canonical graph. Algorithmically, a planar embedding of G is encoded by the boundary of its external face and the clockwise cyclic order of the edges incident with each vertex. Such an encoding is called a *combinatorial* planar embedding of G. Topologically, a planar embedding is uniquely specified by its corresponding combinatorial embedding.

The cyclic edge incidence in a combinatorial embedding is further encoded by the following data structure: for each vertex there is a doubly linked circular list consisting of the edges incident with that vertex in the clockwise order. These lists can be used to efficiently trace the boundary cycles of the faces of G. They can also be used to trace the boundary cycles of the holes of a connected subgraph.

Given a canonical graph of size n, a combinatorial planar embedding can be computed in $O(\log n)$ time with $n \log\log n/\log n$ processors on a deterministic arbitrary-CRCW PRAM [24].

3. Planar orientation. This section describes orientation structures of primal and dual planar directed graphs.

3.1. Dual planar directed graphs. Let G be an embedded canonical graph. The *dual* of G, denoted by $\tilde{G}$, is the embedded canonical graph constructed below:

• For each face f in G there is a vertex, denoted by $\tilde{f}$, in $\tilde{G}$ that corresponds to f. The vertex $\tilde{f}$ is called the *dual* of f.

• For each edge d in G there is an edge, denoted by $\tilde{d}$, in $\tilde{G}$ that corresponds to d. The edge $\tilde{d}$ is called the *dual* of d and is determined as follows. Let f_1 and f_2 be the two faces in G that share d as a boundary edge. If d is positive (respectively, negative) with respect to f_1, then $\tilde{d}$ is a directed edge from $\tilde{f}_1$ to $\tilde{f}_2$ (respectively, from $\tilde{f}_2$ to $\tilde{f}_1$).

• The planar embedding of $\tilde{G}$ is derived from that of G as follows. For each face f of G the vertex $\tilde{f}$ is placed inside f. For each edge d of G the edge $\tilde{d}$ is obtained by turning d counterclockwise 90°.

Note that the above planar embedding may not be unique because its external face is not unique. For the purposes of this paper any planar embedding that fits this construction is suitable.

Also note that the dual of the dual of G is the same as G with all edge directions reversed. Consequently, each vertex in G also corresponds to a face in $\tilde{G}$. In particular, a source (respectively, sink) in $\tilde{G}$ corresponds to a positive (respectively, negative) cycle face in G.

3.2. Primal-dual orientations. The following theorems are crucial to the planar strongly connected component algorithm. Some of them have been highlighted in §1.

LEMMA 3.1 (Folklore). *Let G be an embedded canonical graph. If the two endpoints of an edge d are in the same strongly connected component of G, then the two endpoints of $\tilde{d}$ are in different strongly connected components of $\tilde{G}$.*

Proof. Because the two endpoints of d are in the same strongly connected component of G, there is a vertex-simple directed cycle C in G that contains d. C divides the plane into two regions. Let Δ_{in} and Δ_{out} be the sets of faces of G in the internal hole and the external hole of C, respectively. Let $\tilde{C}$ be the set of edges in $\tilde{G}$ that are the duals of the edges of C. Let $\tilde{\Delta}_{\text{in}}$ and $\tilde{\Delta}_{\text{out}}$ be the sets of vertices of $\tilde{G}$ that are the duals of the faces in Δ_{in} and Δ_{out}, respectively. Observe that by the definition of a dual graph one of the two endpoints of $\tilde{d}$ is in $\tilde{\Delta}_{\text{in}}$ and the other is in $\tilde{\Delta}_{\text{out}}$. For the remainder of the proof, it suffices to show for all vertices $u \in \tilde{\Delta}_{\text{in}}$ and $v \in \tilde{\Delta}_{\text{in}}$ that u and v are not in the same strongly connected component of $\tilde{G}$. By the definition of a dual graph if C is clockwise (respectively, counterclockwise), then the edges in $\tilde{C}$ all point from $\tilde{\Delta}_{\text{in}}$ to $\tilde{\Delta}_{\text{out}}$ (respectively, from $\tilde{\Delta}_{\text{out}}$ to $\tilde{\Delta}_{\text{in}}$). By this unidirection of $\tilde{C}$ no strongly connected component of $\tilde{G}$ contains both a vertex from $\tilde{\Delta}_{\text{in}}$ and one from $\tilde{\Delta}_{\text{out}}$. □

THEOREM 3.2. *Let G be an embedded canonical graph. Then G is strongly connected if and only if $\tilde{G}$ is acyclic.*

Proof. The proof has two directions. One is to show that if G is strongly connected, then $\tilde{G}$ is acyclic. Because G is strongly connected, the endpoints of all edges of G are in the same strongly connected component of G. By Lemma 3.1 the two endpoints of each edge of $\tilde{G}$ are in different strongly connected components of $\tilde{G}$. Thus no strongly connected component of $\tilde{G}$ can contain any edge, i.e., $\tilde{G}$ is acyclic.

The other direction is to show that if G is not strongly connected, then $\tilde{G}$ is not acyclic. Because G is not strongly connected, there is a strongly connected component S in G such that the edges between S and $G - S$ all point from S to $G - S$. Because $G - S$ cannot be empty, either there are vertices in the external hole of S or there are vertices in internal holes of S. These two cases are symmetric, and so it suffices to discuss the case for which there are vertices in the external hole of S. Let U be the set of these vertices. Because G is connected, there are edges between S and U. Let D be the set of these edges. Let $\tilde{D}$ be the set of dual edges of D in $\tilde{G}$. Because D contains all edges around the boundary of the external hole of S and because this boundary is connected, $\tilde{D}$ forms a undirected cycle in $\tilde{G}$. By the unidirection of D the cycle $\tilde{D}$ is a directed one. Thus $\tilde{G}$ is not acyclic. □

COROLLARY 3.3. *Let G be an embedded canonical graph with more than one vertex. If G is strongly connected, then G contains at least one positive face and one negative face.*

Proof. Note that $\tilde{G}$ has at least two vertices. Also, $\tilde{G}$ is acyclic by the strong connectivity of G and Theorem 3.2. Thus $\tilde{G}$ has at least one sink and one source. Because G contains more than one vertex, a source (respectively, sink) in $\tilde{G}$ corresponds to a positive (respectively, negative) face in G. Therefore, G contains at least one positive face and one negative face. □

4. Planar topology.

This section analyzes topological nestings of strongly connected components.

4.1. Enclosure, compactness, indices, and enclosure forests.

Let G be an embedded canonical graph. Let S and T be two strongly connected components of G. S *encloses* T if T is in an internal hole of S. S *immediately* encloses T if S encloses T and no other strongly connected component both encloses T and is enclosed by S.

A *compact* strongly connected component is one that does not enclose any strongly connected components.

The *index* of a noncompact strongly connected component S is the number of noncompact strongly connected components immediately enclosed by S.

The *enclosure forest* of G, denoted by $\mathcal{F}(G)$, is constructed as follows:

- Each strongly connected component of G is a vertex in $\mathcal{F}(G)$.
- There is a directed edge in $\mathcal{F}(G)$ from S to T if and only if S immediately encloses T.

LEMMA 4.1. *The following statements are true:*

1. *$\mathcal{F}(G)$ is a directed forest.*
2. *The compact strongly connected components of G are the leaves of $\mathcal{F}(G)$.*
3. *The noncompact strongly connected components of G are the internal vertices of $\mathcal{F}(G)$. The index of a noncompact strongly connected component S in G is the number of nonleaf children of S in $\mathcal{F}(G)$.*

Proof. The proof is straightforward. □

The strongly connected component algorithm focuses on converting the index-0 and index-1 strongly connected components of G into compact ones. The next theorem confirms that there are enough noncompact components of these two indices.

THEOREM 4.2. *Let G be an embedded canonical graph. If not all strongly connected components of G are compact, then more than half of the noncompact strongly connected components are of index* 0 *or* 1.

Proof. This theorem follows from Lemma 4.1 and the fact that if a rooted tree has at least two vertices, then more than half of the internal vertices have zero or one nonleaf child each. □

4.2. Cycle faces and compact strongly connected components.

The next lemma was used in §1.

LEMMA 4.3. *Let G be an embedded canonical graph. Let S be a compact strongly connected component of G with more than one vertex. Then S contains the boundary of a cycle face of G.*

Proof. S can be regarded both as a subgraph of G and as a strongly connected component. By its strong connectivity the subgraph S is an embedded canonical graph. Because S has more than one vertex, by Corollary 3.3 the subgraph S contains at least one positive face and one negative face. If the external face of the subgraph S is a cycle face, then the subgraph S has at least one internal cycle face; otherwise, the subgraph S has at least two internal cycle faces. Because the strongly connected component S is compact in G, an internal face of the subgraph S is also a face in G. Thus at least one of the cycle faces in the subgraph S is a cycle face in G. □

4.3. Structures of singular compact strongly connected components.

The reason for focusing on index-0 and index-1 components is that the strongly connected components enclosed by them have useful structures and thus can be efficiently separated. Some of these structures are analyzed below.

Let G be an embedded canonical graph. A compact strongly connected component S of G is *inward* (respectively, *outward*) *singular* if there is no directed path from a noncompact strongly connected component to S (respectively, from S to a noncompact strongly connected component).

LEMMA 4.4. *Let T be a strongly connected component of G. Let X be a hole of T that contains no noncompact strongly connected components. Then every strongly connected component in X is a singular compact strongly connected component in G.*

Proof. Let S be a compact strongly connected component in X. To prove the lemma by contradiction, assume that S is not singular. Then there are two directed paths P and Q with P going from S to a noncompact strongly connected component in G and Q going from a noncompact strongly connected component to S in G. Because all noncompact strongly connected components in G are outside X, P, and Q go through T. Consequently, S and T are in the same strongly connected component, contradicting the condition that S is in X. □

4.4. Structures of index-0 strongly connected components.

LEMMA 4.5. *Let G be an embedded canonical graph. For each index-*0 *strongly connected component S of G, every strongly connected component in the internal holes of S is a singular compact strongly connected component.*

Proof. This lemma follows from Lemma 4.4 and the fact that an internal hole of an index-0 component contains only compact strongly connected components. □

4.5. Structures of index-1 strongly connected components. Let G be an embedded canonical graph. Let S be a strongly connected component of G. Let X be a hole of S in G. The following notations are used for analyzing index-1 components in this and other sections:

- $\mathcal{E}_X(S)$ is the set of edges between S and the vertices in X.
- $\mathcal{E}_{\text{in}}(S)$ is the set of edges between S and the vertices in its internal holes.
- $\mathcal{E}_{\text{ns}}(S)$ is the set of edges in $\mathcal{E}_{\text{in}}(S)$ except those between S and the singular compact strongly connected components immediately enclosed by S.

LEMMA 4.6. *Let S be an index-*1 *strongly connected component of G. Let X be the internal hole of S that contains a noncompact strongly connected component. The following statements are true:*

1. *Every strongly connected component in the internal holes of S except possibly X is a singular compact strongly connected component.*
2. *The edges in $\mathcal{E}_{\text{ns}}(S)$ either all point from S or all point to S.*

Proof. The first statement follows from Lemma 4.4 and the fact that no internal hole of S except X contains a noncompact strongly connected component.

The second statement is proved as follows. Let T be the unique noncompact strongly connected component immediately enclosed by S. It suffices to show the following property: for each edge $d \in \mathcal{E}_{\text{ns}}(S)$, d can be extended into a directed path between S and T such that if d points to (respectively, from) S, then that path goes from T to S (respectively, from S to T). Because S and T are different strongly connected components, this extension property ensures that $\mathcal{E}_{\text{ns}}(S)$ cannot contain two edges with one pointing to S and the other pointing from S.

The edge d is extended as follows. Let $d = u \to v$. There are two cases: either $u \in S$ or $v \in S$. Because these two cases are symmetric, only the case of $u \in S$ is described. If $v \in T$, then d itself is already a directed path from S to T. Thus, without loss of generality, assume that $v \notin T$. Then v is in some nonsingular compact strongly connected component R immediately enclosed by S. Because R is nonsingular, there is a directed path P from R to some noncompact strongly connected component. Because S is of index 1, the noncompact strongly connected components of G except S and T are either outside S or inside T. Thus P goes through either S or T. Because d goes from S to R and because S and R are different strongly connected components, P cannot go through S and must go through T. Then the edge d, a directed path in R, and a subpath of P together form a directed path from S through R to T. □

5. Edge cutting and vertex expansion. This section describes a procedure called CutSegment that cuts edges in an embedded canonical graph. CutSegment is coupled with another procedure called ExpandVertex that eliminates undesirable large-degree vertices. These two procedures are extremely simple but, surprisingly, together reveal very useful structures of planar directed graphs: *clusters*, *territories*, and *chambers*. These procedures and structures are detailed in §5.1 through §5.5, respectively.

5.1. ExpandVertex. The three structures exposed by CutSegment subtly depend on the nonexistence of vertices with degree greater than three. For this reason, a vertex of degree four or more is called a *large-degree* vertex.

A useful operation for eliminating large-degree vertices is to replace each of them by a positive face or a negative face. This operation is called *vertex expansion*. An example of vertex expansion is shown in Fig. 1. The procedure ExpandVertex in Algorithm 1 details how to expand vertices into negative faces.

ALGORITHM 1. *Procedure for expanding large-degree vertices into negative faces.*

```
Procedure ExpandVertex
Input: an embedded canonical graph $G$ and a set $U$ of large-degree vertices.
Output: $G$ with each vertex in $U$ expanded into a negative face.
begin
    for each vertex $u \in U$ do
        begin
            1. Let $d_1, \ldots, d_s$ be the edges incident with $u$ in the counterclockwise order.
            2. Create $s$ copies of $u$, namely, $u_1, \ldots, u_s$.
            3. Replace the $u$-end of each $d_i$ by $u_i$.
            4. Link the vertices $u_i$ into a counterclockwise directed cycle, i.e., $u_1 \rightarrow$
               $u_2 \rightarrow \cdots \rightarrow u_{s-1} \rightarrow u_s \rightarrow u_1$.
        end.
    return $G$ with the above changes.
end.
```

By symmetry a procedure for expanding large-degree vertices into positive faces can be obtained by reversing the orientations in ExpandVertex.

THEOREM 5.1. *The following statements are true:*

1. *The output graph of* ExpandVertex *is an embedded canonical graph and has the same strongly connected components as the input graph does.*

2. ExpandVertex *runs in* $O(\log n)$ *time with* $n/\log n$ *processors for an input graph of size* n.

Proof. The first statement is straightforward. As for the second statement, ExpandVertex can be performed in a straightforward manner with fundamental techniques, including prefix computation [18], [19], and list ranking [3], [7], [15]. □

5.2. CutSegment. Let G be an embedded canonical graph. Let g be a noncycle subface of G. The *positive* (respectively, *negative*) *breakpoints* of g are the following vertices:

• the local sources and sinks of g;

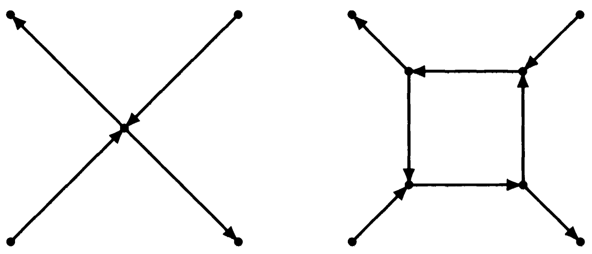


FIG. 1. *Example of expanding a large-degree vertex into a negative face.*

• the vertices shared by g and positive (respectively, negative) subfaces.

Remarks. Cycle subfaces are not assigned breakpoints. A vertex may be a breakpoint for a noncycle subface but not for another.

A *segment* of g is the subpath of g between two consecutive breakpoints with the same orientation. Observe that a segment is a directed path. In light of this, a *positive* (respectively, *negative*) segment P of g is one with the following properties.

• The segment P is between two consecutive positive (respectively, negative) breakpoints of g.

• Every edge of P is a positive (respectively, negative) edge of g.

With the above definitions Algorithm 2 details the procedure CutSegment.

ALGORITHM 2. *Procedure for edge cutting.*

```
Procedure CutSegment
Input: an embedded canonical graph $G$ and two symbolic parameters $x$ = "first" or
"last" and $y$ = "positive" or "negative."
Output: $G$ without the $x$ edge of each $y$ segment.
begin
    1. Find all $y$ breakpoints of $G$.
    2. Find the $x$ edge of each $y$ segment in $G$.
    3. Delete from $G$ all the edges found above.
    4. return $G$.
end.
```

Figure 2 provides an example for CutSegment.

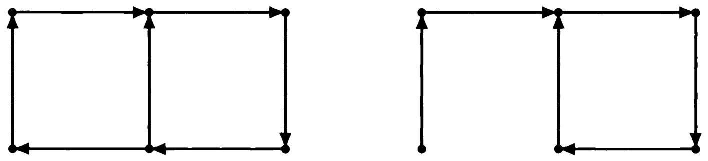


FIG. 2. *Example of cutting the first edge of each positive segment.*

LEMMA 5.2. *Let G be an embedded canonical graph of size n. The following can be computed in $O(\log n)$ time with $n/\log n$ processors:*

1. *the subfaces of G with their orientations,*
2. *the positive (or negative) subfaces of G, and*

3. *the positive* (*or negative*) *breakpoints of the noncycle subfaces of G.*

Proof. Observe that the subfaces of a face f are just the biconnected components of $\mathcal{B}(f)$. Similarly, the subfaces of G are the face boundaries of the subgraphs that are induced by the biconnected components of G. On the basis of these observations, the subfaces of G can be computed in a straightforward manner in $O(\log n)$ time with $n/\log n$ processors. This computation uses optimal algorithms for planar connectivity [14], planar biconnectivity [14], list ranking [3], [7], [15], and prefix computation [18], [19]. Once the subfaces of G are obtained, it is straightforward to identify the cycle subfaces of G and the breakpoints of the noncycle subfaces in $O(\log n)$ time with $n/\log n$ processors. □

THEOREM 5.3. CutSegment *runs in $O(\log n)$ time with $n/\log n$ processors for an embedded canonical graph of size n.*

Proof. Let G be the input graph to CutSegment. By symmetry it suffices to prove the case of deleting the first edge of each positive segment. The subfaces and the positive breakpoints of G are obtained by means of Lemma 5.2. The first edges of the positive segments are simply the positive edges outgoing from positive breakpoints on the subfaces. These edges can be easily found and deleted in $O(\log n)$ time with $n/\log n$ processors. Thus CutSegment runs within the stated complexity. □

5.3. Clusters. Let G be an embedded canonical graph. Two positive (respectively, negative) subfaces h_1 and h_2 of G are *linked* if there is a sequence of positive (respectively, negative) subfaces $g_1, \dots, g_s$ such that $h_1 = g_1$, $h_2 = g_s$, and for $1 \le i \le s-1$, g_i and g_{i+1} share at least one vertex.

A *positive* (respectively, *negative*) *cluster* of G is a maximal set of positive (respectively, negative) subfaces that are linked.

LEMMA 5.4. *Let G be an embedded canonical graph of size n. Then the positive* (*or negative*) *clusters of G can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The positive (respectively, negative) subfaces of G are computed by means of Lemma 5.2. Then the positive (respectively, negative) clusters can be found in $O(\log n)$ time with $n/\log n$ processors by using an optimal parallel algorithm for planar connectivity [14]. □

The following lemma gives a preliminary analysis of CutSegment(G, first, positive). Similar lemmas for the other parameter combinations can be obtained by symmetry.

LEMMA 5.5. *The following statements are true:*

1. *A cluster of G is contained in a strongly connected component of G.*
2. CutSegment(G, *first, positive*) *contains every positive cluster of G.*
3. CutSegment(G, *first, positive*) *has no edges outgoing from positive clusters.*
4. *If the degree of a vertex u is at most three in G, then the outdegree of u is at most one in* CutSegment(G, *first, positive*).

Proof. The statements are proved as follows.

Statement 1. This statement follows from the fact that a cluster consists of linked directed cycles.

Statement 2. By definition a positive subface has no positive breakpoints and thus no positive segments. Thus no edge of a positive subface is cut, and every positive cluster is in CutSegment(G, first, positive).

Statement 3. Let d be an edge from a positive cluster Φ of G to a vertex not in Φ. It suffices to show that d is cut in CutSegment(G, first, positive). Let $d = u \rightarrow v$ with $u \in \Phi$ and $v \notin \Phi$. Let g_t and g_b be the two noncycle subfaces that contain d such that d is positive on g_t and negative on g_b. Let P_t be the positive segment of g_t that contains d.

Then d is cut in CutSegment(G, first, positive) because u is a positive breakpoint and d is the first edge of P_t.

Statement 4. There are two cases based on whether u is on a positive subface of G or not.

• Case (1): u is on a positive subface g. The statement is trivially true if the outdegree of u is at most one in G. Thus, without loss of generality, assume that the outdegree of u is two or three in G. Then, because u has at most three incident edges in G and at least two of them are outgoing, u has exactly two outgoing edges and one incoming edge. Exactly one of the outgoing edges is not on any positive subface and thus is an outgoing edge from the positive cluster containing g. By the third statement of this lemma that edge is cut.

• Case (2): u is not on a positive subface. This case is further divided into subcases. Let Case (i_1, i_2, i_3) denote the case that the degree, indegree, and outdegree of u in G are i_1, i_2, and i_3, respectively. The statement is trivially true for $i_3 \leq 1$. Thus it suffices to discuss Cases (2,0,2), (3,1,2), and (3,0,3).

– Case (2,0,2). Let $d_1 = u \rightarrow v_1$ and $d_2 = u \rightarrow v_2$ be the two edges adjacent with u. Let g_t and g_b be the noncycle subfaces that contain d_1 and d_2 such that d_1 is positive on g_t and negative on g_b. Let P_t be the positive segment of g_t that contains d_1. Let P_b be the positive segment of g_b that contains d_2. The edge d_1 is cut in CutSegment(G, first, positive) because it is the first edge of P_t. Similarly, d_2 is cut.

– Case (3,1,2). Let $d_1 = u \leftarrow v_1$, $d_2 = u \rightarrow v_2$, $d_3 = u \rightarrow v_3$ be the edges adjacent with u in the clockwise order around u. Let $g_{1,2}$, $g_{2,3}$, $g_{3,1}$ be the noncycle subfaces that contain, respectively, d_1 and d_2, d_2, and d_3, and d_3 and d_1. Let $P_{2,3}$ be the positive segment of $g_{2,3}$ that contains d_2. Let $P_{3,1}$ be the positive segment of $g_{3,1}$ that contains d_1 and d_3. The edge d_1 may or may not be cut in CutSegment(G, first, positive), depending on whether v_1 is a positive breakpoint of $g_{3,1}$. The edge d_2 is cut because it is the first edge of $P_{1,2}$. The edge d_3 is not cut because it cannot be the first edge of $P_{3,1}$.

– Case (3,0,3). Let $d_1 = u \rightarrow v_1$, $d_2 = u \rightarrow v_2$, $d_3 = u \rightarrow v_3$ be the edges adjacent with u in the clockwise order around u. Let $g_{1,2}$, $g_{2,3}$, $g_{3,1}$ be the noncycle subfaces that contain, respectively, d_1 and d_2, d_2 and d_3, and d_3 and d_1. Let $P_{1,2}$ be the positive segment of $g_{1,2}$ that contains d_1. Let $P_{2,3}$ be the positive segment of $g_{2,3}$ that contains d_2. Let $P_{3,1}$ be the positive segment of $g_{3,1}$ that contains d_3. The edge d_1 is cut in CutSegment(G, first, positive) because it is the first edge of $P_{1,2}$. Similarly, d_2 and d_3 are cut. □

5.4. Territories. Let G be an embedded canonical graph that has no large-degree vertices except possibly in positive clusters. A *territory* of a positive cluster of G is a vertex subset as defined by the procedure in Algorithm 3.

ALGORITHM 3. *Procedure for computing the positive territories.*

```
Procedure ComputePositiveTerritories
Input: an embedded canonical graph $G$ with no large-degree vertices except possibly in
positive clusters.
Output: the positive territories of $G$.
begin
  1. Let $G_c$ = CutSegment($G$, first, positive).
  2. Let $G_d$ = CutSegment($G$, last, positive).
  3. for each positive cluster $\Phi$ of $G$ do
       begin
         3-1. Let $C_c(\Phi)$ be the connected component in $G_c$ that contains $\Phi$.
```

```
        3-2. Let $\mathcal{C}_d(\Phi)$ be the connected component in $G_d$ that contains $\Phi$.
        3-3. Let $\mathcal{S}(\Phi)$ be the subgraph of $G$ induced from the vertices shared by
             $\mathcal{C}_c(\Phi)$ and $\mathcal{C}_d(\Phi)$.
        3-4. Let $\mathcal{T}(\Phi)$, the territory of $\Phi$, be the connected component in $\mathcal{S}(\Phi)$
             that contains $\Phi$.
    end.
  4. return all $\mathcal{T}(\Phi)$ computed above.
end.
```

For brevity a territory of a positive cluster is called a *positive* territory. A *negative* territory is defined by substituting "negative" for "positive" in the above definition.

Territories are the key neighborhoods of directed cycles contracted in the planar strongly connected component algorithms. The following analysis focuses on positive territories. A similar analysis for negative territories can be obtained by symmetry.

The next theorem provides the basis for contracting territories in the planar strongly connected component algorithm.

THEOREM 5.6. *The following statements are true:*

1. *$\mathcal{C}_c(\Phi)$, $\mathcal{C}_d(\Phi)$, and $\mathcal{T}(\Phi)$ exist.*

2. *Each connected component in G_c (or G_d) contains at most one positive cluster of G. Consequently, distinct positive clusters have disjoint territories.*

3. *Each $\mathcal{T}(\Phi)$ is connected and is in the strongly connected component of G that contains Φ.*

Proof. The statements are proved as follows.

Statement 1. The existence of $\mathcal{C}_c(\Phi)$ follows from Lemma 5.5(2) and the connectivity of Φ. By symmetry $\mathcal{C}_d(\Phi)$ exists. The existence of $\mathcal{T}(\Phi)$ readily follows from that of $\mathcal{C}_c(\Phi)$ and $\mathcal{C}_d(\Phi)$.

Statement 2. By symmetry it suffices to show the statement for G_c. To prove by contraction, assume that G_c contains an undirected vertex-simple path $P = u_1, \dots, u_s$ between two distinct positive clusters Φ_1 and Φ_2 of G with $u_1 \in \Phi_1$ and $u_s \in \Phi_2$. Without loss of generality, further assume that the internal vertices of P are not in any positive cluster of G.

Because u_2 is not in Φ_1, by Lemma 5.5(3) the edge of P between u_1 and u_2 points from u_2 to u_1. Similarly, the edge of P between u_{s-1} and u_s points from u_{s-1} to u_s. Therefore, P has at least three vertices, and at least one internal vertex u_i of P is a local source of P. By the vertex-simplicity of P the outdegree of u_i is at least two in G_c. On the other hand, because u_i is not in any positive cluster of G, the degree of u_i is at most three in G by the degree condition of G. Thus by Lemma 5.5(4) the outdegree of u_i in G_c is at most one. This contradicts the above conclusion that the outdegree of u_i is at least two in G_c.

Statement 3. The connectivity of $\mathcal{T}(\Phi)$ is required by the definition of a territory. For the remainder of the proof, by Lemma 5.5(1) it suffices to show that $\mathcal{C}_c(\Phi)$ (respectively, $\mathcal{C}_d(\Phi)$) contains a directed path P from each vertex u to Φ (respectively, from Φ to each vertex u). By symmetry it suffices to prove this claim only for $\mathcal{C}_c(\Phi)$. Let $P = v_1, \dots, v_s$ be an undirected path in $\mathcal{C}_c(\Phi)$ with $v_1 = u$ and $v_s \in \Phi$ such that P is vertex-simple and intersects with Φ only at v_s. The goal is to show that P is a directed path.

If $s = 1$, then P is obviously a directed path. So, without loss of generality, assume that $s > 1$. Then by the second statement of this theorem $v_1, \dots, v_{s-1}$ are not in any positive cluster of G. By the degree condition of G the degrees of $v_1, \dots, v_{s-1}$ are at most three in G. Because v_{s-1} is not in Φ, by Lemma 5.5(3) the edge of P between v_{s-1} and v_s points to v_s. Furthermore, by Lemma 5.5(4) the outdegree of v_{s-1} in $\mathcal{C}_c(\Phi)$ is at most one. Because v_{s-1} already has an edge pointing to v_s, the edge of P between v_{s-2} and v_{s-1} points to v_{s-1}. Repeating this argument inductively shows that P is indeed a directed path. □

THEOREM 5.7. *Let n be the size of G. The positive territories of G can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The complexity of the procedure in Algorithm 3 is analyzed as follows. Steps 1 and 2 are computed by means of Theorem 5.3. To start Step 3 the positive clusters of G are computed by means of Lemma 5.4. Then Steps 3-1–3-4 can be computed in $O(\log n)$ time with $n/\log n$ processors. These four steps use Theorem 5.6(2) and an optimal parallel algorithm for planar connectivity [14]. Thus the positive territories can be computed with the stated complexity. □

The next theorem is crucial for analyzing the contraction procedures of the planar strongly connected component algorithm.

THEOREM 5.8. *Let g be a noncycle subface in G. Let P be a directed subpath of g that has at least one internal vertex and consists of positive edges of g. If the endpoints of P are in positive territories of G but its internal vertices are not, then the endpoints of P are in different positive territories.*

Proof. Let $P = p_1, \ldots, p_s$. To prove by contradiction, assume that p_1 and p_s are in the same positive territory $\mathcal{T}(\Phi)$. It suffices to show $P \subseteq \mathcal{T}(\Phi)$, contradicting the condition that the internal vertices of P are not in any positive territory.

Because the internal vertices of P are not in any positive territory of G, they are not in any positive cluster of G. Because P is a directed path, the internal vertices of P are not local sources or sinks on g. Thus the internal vertices of P are not positive breakpoints for g. Furthermore, because g is a noncycle subface and P is positive on g, the path P is a subpath of a positive segment Q of g.

Let $Q = q_1, \ldots, q_t$. After the first edge of Q is cut, the path $Q_c = q_2, \ldots, q_t$ remains in G_c. Because the path $P_c = p_2, \ldots, p_s$ is a subpath of Q_c, the path P_c is connected to p_s in G_c. Because $p_s \in \mathcal{T}(\Phi) \subseteq \mathcal{C}_c(\Phi)$, the path P_c is in $\mathcal{C}_c(\Phi)$. The vertex p_1 is also in $\mathcal{C}_c(\Phi)$ because $p_1 \in \mathcal{T}(\Phi) \subseteq \mathcal{C}_c(\Phi)$. Thus P is in $\mathcal{C}_c(\Phi)$. By symmetry P is also in $\mathcal{C}_d(\Phi)$. Consequently, P is in $\mathcal{S}(\Phi)$. Because P is connected to $\mathcal{T}(\Phi)$, the path P lies in $\mathcal{T}(\Phi)$. □

5.5. Chambers. Let G be an embedded canonical graph. Let S be a strongly connected component in G with more than one vertex. A *chamber* of S is a nonempty hole X of S with the following properties:

- The edges in $\mathcal{E}_X(S)$ either all point from S or all point to S.
- X is a cycle hole.

Remark. As shown by Lemma 4.6, index-1 components hold a unidirectional property very similar to the first property above. In fact, a key stage of the planar strongly connected component algorithm is to derive chambers from index-1 components.

The edges in $\mathcal{E}_X(S)$ are called the *chamber* edges of X. The chamber X is *inward* (respectively, *outward*) if the edges in $\mathcal{E}_X(S)$ all point to X (respectively, all point to S). The chamber X is *positive* (respectively, *negative*) if it is a positive (respectively, negative) hole.

Note that if the chamber edges of X can be found efficiently, then these edges can be cut to efficiently separate S from the content of X. This section discusses how to use CutSegment to find the chamber edges of G. The discussion focuses on positive outward chambers. The first part of the discussion assumes that G has no large-degree vertices. The second part allows G to have large-degree vertices.

5.5.1. Chamber edges of graphs without large-degree vertices.

THEOREM 5.9. *Let G be an embedded canonical graph with no large-degree vertices. Let Δ be the set of biconnected components in* CutSegment(G, *first, positive*) *that each contain at least two edges. Then each component in Δ is a directed cycle. Moreover, Δ consists of the positive subfaces in G and the boundary cycles of positive outward chambers.*

Proof. This theorem is based on the following claims:

1. CutSegment(G, first, positive) contains all positive subfaces and all boundary cycles of positive outward chambers of G.

2. A vertex-simple undirected cycle in CutSegment(G, first, positive) is either a positive subface of G or the boundary cycle of a positive outward chamber.

3. Two distinct vertex-simple undirected cycles cannot share a vertex in the graph CutSegment(G, first, positive).

With the above claims the theorem is shown as follows. If a biconnected component of CutSegment(G, first, positive) contains more than one edge, then that component contains at least one vertex-simple undirected cycle. By the third claim, that component contains exactly one such cycle. By the second claim, that cycle is a positive subface or the boundary cycle of a positive outward chamber of G. Conversely, by the first claim CutSegment(G, first, positive) preserves the positive subfaces and the boundary cycles of the positive outward chambers in G. These cycles are preserved in the biconnected components of CutSegment(G, first, positive) that each contain more than one edge.

Claim 1. The claim for positive subfaces follows from Lemma 5.5(2). The claim for chambers is proved as follows. Let X be a positive outward chamber in G. Let C be the boundary cycle of X. Because G has no large-degree vertices, C is vertex-simple. Assume by symmetry that X is an internal hole of a strongly connected component. Then C is a clockwise-directed cycle and the chamber edges of X are inside C. Because the chamber edges point to C, the heads of the chamber edges divide C into directed subpaths. Let d_1 be a chamber edge. Let d_2 be the chamber edge next to d_1 in the clockwise order on C. Let P be the subpath of C from the head of d_1 to the head of d_2. Observe that because d_1 points to C, the edge d_1 and the path P are contained in a positive segment Q. Because d_1 precedes P in Q, the boundary edges of X in P cannot be the first edge of Q and thus cannot be cut in CutSegment(G, first, positive).

Claim 2. Let C be a vertex-simple undirected cycle in G. It is equivalent to show that if C is neither a positive subface nor the boundary cycle of a positive outward chamber, then at least one edge of C is deleted in CutSegment(G, first, positive). There are two cases based on whether C is a directed cycle or not.

- Case (1): C is not a directed cycle. Then there is a local source on C with two outgoing edges. Because G has no large-degree vertices, by Lemma 5.5(4) CutSegment deletes at least one of these two edges.

- Case (2): C is a directed cycle but is neither a positive subface nor the boundary cycle of a positive outward chamber. There are two subcases based on whether C runs clockwise or counterclockwise. Assume by symmetry that C runs clockwise. Because C is not a positive subface, C encloses some edges. In particular, there is an edge d_1 incident with C and inside C. Because C is not the boundary cycle of a positive outward chamber of G, there is an edge residing inside C and pointing from C. Without loss of generality, assume that d_1 points from a vertex u of C. Let d_2 and d_3 be the two edges on C with d_2 entering u and d_3 leaving u. Because G has no large-degree vertices, the edges d_i are the only edges incident with u in G. Observe that d_1 and d_3 are on a noncycle subface and d_3 is the first edge of a positive segment of that subface. Thus d_3 is cut in CutSegment(G, first, positive).

Claim 3. To prove by contradiction, assume that there are two vertex-simple undirected cycles C_1 and C_2 in CutSegment(G, first, positive) that share at least one vertex. Because G has no large-degree vertices, C_1 and C_2 share at least one edge. By the second claim C_1 and C_2 are directed cycles. There are three cases: (1) both C_1 and C_2 run clockwise, (2) both C_1 and C_2 run counterclockwise, or (3) C_1 and C_2 are in the opposite directions. Because these three cases are symmetric, only Case (1) is discussed. Because C_1 and C_2 share at least one edge, one cycle is inside the other. Assume by symmetry that C_2 is inside C_1. Then C_1 is not a positive subface. Also, the internal hole of C_1 is not a positive outward chamber because C_2 violates the unidirectional property of that hole as a chamber. These two conclusions contradict the second claim. □

THEOREM 5.10. *Let G be an embedded canonical graph with no large-degree vertices. Let n be the size of G. The procedure in Algorithm* 4 *computes the chamber edges of positive outward chambers of G in $O(\log n)$ time with $n/\log n$ processors.*

ALGORITHM 4. *Procedure for computing the chamber edges of positive outward chambers of graphs without large-degree vertices.*

```
Procedure ComputePOCE
Input: an embedded canonical graph $G$ with no large-degree vertices.
Output: the chamber edges of positive outward chambers in $G$.
begin
    1. Let $G_c$ = CutSegment($G$, first, positive).
    2. Let $\Delta$ be the set of biconnected components of $G_c$ that contain at least two edges
       each.
    3. Delete from $\Delta$ the positive subfaces of $G$.
    4. Find the chamber edges of the positive outward chambers of $G$, whose boundary
       cycles are now in $\Delta$.
    5. return all edges computed at Step 4.
end.
```

Proof. The correctness of ComputePOCE follows directly from Theorem 5.9. As for its complexity, Steps 1–4 of the procedure can be performed in $O(\log n)$ time with $n/\log n$ processors as follows. Step 1 is performed by means of Theorem 5.3. Step 2 uses an optimal parallel algorithm for planar biconnectivity [14]. Step 3 is performed by means of Lemma 5.2(2). To collect the desired chambers edges, Step 4 uses list ranking and prefix computation to process the doubly linked circular lists of the combinatorial embedding of G. Thus the total complexity of ComputePOCE is as stated. □

5.5.2. Chamber edges of graphs with large-degree vertices.

THEOREM 5.11. *Let G be an embedded canonical graph that may have large-degree vertices. Let G' be the graph constructed by expanding each large-degree vertex of G into a negative face. The following statements are true:*

1. *G' is an embedded canonical graph with no large-degree vertices.*

2. *The chamber edges of the positive outward chambers of G' are originally the edges incident with the large-degree sinks of G and the chamber edges of the positive outward chambers of G.*

Proof. The first statement is obtained by straightforward observation. To prove the second statement, observe that G and G' have the same strongly connected components. Also, each strongly connected component has the same nonempty holes in G and G'. Let S be a strongly connected component in G. Let X be a nonempty hole of S in G. Let S' and X' be the versions of S and X in G'. Observe that $\mathcal{E}_X(S)$ and $\mathcal{E}_{X'}(S')$ consist of the same edges with the same directions to or from S and S'.

These observations permit the second statement to be proved by showing that the positive outward chambers of G' are originally the external holes of the large-degree sinks of G and the positive outward chambers of G. This claim follows directly from the following case analysis on $\mathcal{B}(X)$ for the second property of a chamber.

If $\mathcal{B}(X)$ contains no large-degree vertices, then $\mathcal{B}(X') = \mathcal{B}(X)$. Otherwise, vertex expansion inserts new positive edges into $\mathcal{B}(X)$. There are three cases.

• Case (1): If $\mathcal{B}(X)$ consists of a large-degree vertex, then X' is the external hole of S' and is a positive one.

• Case (2): If X is a positive hole, then X' is also a positive hole.

• Case (3): If X has more than one boundary vertex but is not a positive hole, then X' is not a positive hole. □

THEOREM 5.12. *Let G be an embedded canonical graph. Let n be the size of G. The procedure in Algorithm 5 computes the chamber edges of positive outward chambers of G in $O(\log n)$ time with $n/\log n$ processors.*

ALGORITHM 5. *Procedure for computing the chamber edges of positive outward chambers.*

```
Procedure ComputePositiveOutwardChamberEdges
Input: an embedded canonical graph $G$.
Output: the chamber edges of positive outward chambers in $G$.
begin
    1. Let $G'$ be the graph constructed by expanding each large-degree vertex of $G$ into
       a negative face.
    2. Let $D'$ be the set of the chamber edges of positive outward chambers of $G'$ com-
       puted by the procedure in Algorithm 4.
    3. Let $D$ be the set of edges in $G$ that correspond to those in $D'$ and are not the
       edges incident with the large-degree sinks of $G$.
    4. return the edges in $D$.
end.
```

Proof. The correctness of the procedure follows from Theorems 5.10 and 5.11. As for its complexity, Steps 1 through 3 of the procedure can be performed in $O(\log n)$ time with $n/\log n$ processors as follows. Step 1 is performed by means of Theorem 5.1(2). Step 2 is performed by means of Theorem 5.10. Step 3 is performed by identifying the sinks of G in $O(\log n)$ time with $n/\log n$ processors. Thus the total complexity of the procedure is as stated. □

The above discussion has described how to compute the chamber edges of positive outward chambers. The chamber edges of the other groups of chambers can be similarly computed on the basis of the symmetry summarized in Table 1.

6. Vertex contraction.

Throughout this paper vertex contraction always satisfies the following specifications:

• Vertex contraction keeps all multiple edges and deletes all loop edges that it creates.

• Vertex contraction contracts only connected vertex subsets of strongly connected components.

Note that deleting all loop edges simplifies contracted strongly connected components. Keeping all multiple edges maintains crucial contraction invariants. In particular, each uncontracted edge keeps a unique identity because its link remains unchanged, whereas its endpoints may be altered by contraction.

TABLE 1
Symmetry for computing chamber edges.

Chambers		Vertex Expansion	CutSegment		Discarded Edges
positive	outward	negative faces	first	positive	sinks
positive	inward	negative faces	last	positive	sources
negative	outward	positive faces	first	negative	sinks
negative	inward	positive faces	last	negative	sources

Also note that contracting only connected vertex subsets preserves planarity. Contracting only subsets of strongly connected components preserves the strongly connected components.

In fact, the planar strongly connected component algorithm contracts only territories and vertex subsets expanded from large-degree vertices. Theorem 5.6(3) ensures that a territory satisfies the second specification above. The vertex subset expanded from a large-degree vertex clearly satisfies the second specification.

The detailed discussion of vertex contraction is divided into four parts. Section 6.1 gives the definition of a planar embedding induced by vertex contraction, §6.2 discusses contraction invariants, §6.3 describes contraction procedures, and §6.4 analyzes contraction effects.

6.1. Planar embeddings induced by vertex contraction. Let G be an embedded canonical graph. Let B be the subgraph induced by a connected vertex subset of a strongly connected component of G. Let G' be the graph constructed from G by contracting B into a vertex B'. If B consists of at most one vertex, then G' and G are the same. Otherwise, a planar embedding for G' is specified as follows:

• For every vertex $u \notin B$ the clockwise cyclic order of the edges incident with u is the same in G and G'.

• The edges around each nonempty hole X of B stay together around B', and their clockwise cyclic order around B' is the same as their cyclic order around the boundary cycle of X in the negative direction of X.

• All uncontracted edges on the boundary of the external face of G remain on that of G' and have the same orientations with respect to both external faces.

• If B contains a boundary vertex of the external face of G, then B' is on the boundary of the external face of G'.

In general, such a planar embedding is not unique. For the purposes of this paper any planar embedding that fits this construction is suitable.

THEOREM 6.1. *Given an embedded canonical graph of size n, a planar embedding induced by contracting a disjoint family of connected vertex subsets can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The edges around a new vertex of G' are collected by processing the doubly linked circular lists of the combinatorial embedding of G. The computation takes $O(\log n)$ time and $n/\log n$ processors and uses optimal parallel algorithms for list ranking [3], [7], [15], prefix computation [18], [19], tree contraction [1], [6], [9], [12], [17], [20], and planar connectivity [14]. □

6.2. Contraction invariants. The next four lemmas describe contraction invariants for subfaces, paths, holes, and subholes. The lemmas for subfaces, paths, and subholes are false if multiple edges are not kept.

To prove the invariant lemmas, note that the above definition of vertex contraction is not directly suitable for analyzing contraction invariants because it permits many structural changes to occur simultaneously. This difficulty is resolved by observing that vertex contraction can be done by performing a sequence of contracting a nonloop edge and deleting a loop edge. The planar embedding induced by contracting a nonloop edge is constructed by topologically shrinking that edge into a single vertex. The planar embedding induced by deleting a loop edge is constructed by simply deleting that loop.

The proofs of the invariant lemmas are obtained by straightforward induction based on direct observations about the effects of deleting a single loop edge and contracting a single nonloop edge. For contracting a connected vertex subset there may be several sequences of contracting and deleting edges. Different sequences may result in different planar embeddings. These planar embeddings all satisfy the definition of vertex contraction, and thus every sequence of deletions and contractions suffices for proving the invariant lemmas.

6.2.1. Subface invariants. Let G be an embedded canonical graph. Let G' be a contracted version of G. Let g be a subface in G. Let D be the set of edges of g. Let D' be the set of edges in D that remain in G'.

LEMMA 6.2. *There is a face g' in G' such that the subfaces of g' can be partitioned into two collections Δ_{new} and Δ_{old} with the following properties:*

1. *The subfaces in Δ_{new} contain no edges from D'.*
2. *The subfaces in Δ_{old} consist of exactly the edges in D', and each edge in D' has the same orientation with respect to g and g'.*

Proof. See the above discussion. □

6.2.2. Path invariants. Let G be an embedded canonical graph. Let G' be a contracted version of G. Let g' be a subface in G'. Let P' be a subpath of g' whose internal vertices are original vertices in G.

LEMMA 6.3. *P' originally is a subpath of a subface g in G, and each edge of P' has the same orientation on g and g'.*

Proof. See the above discussion. □

6.2.3. Hole invariants. Let G be an embedded canonical graph. Let G' be a contracted version of G. Let S and T be two strongly connected components in G. Let S' and T' be the versions of S and T in G'.

LEMMA 6.4. *The following statements are true:*

1. *The content of an internal hole of S' is the union of the contents of some internal holes of S.*
2. *The content of the external hole of S' is the union of the contents of the external hole of S and possibly some internal holes of S.*
3. *Distinct holes of S' do not share the content from the same hole of S.*
4. *If S does not enclose T, then S' does not enclose T'.*
5. *If S is compact, then S' is compact.*
6. *If S has exactly one nonempty internal hole, then S' either is compact or has exactly one nonempty internal hole.*

Proof. See the above discussion. □

6.2.4. Subhole invariants. Let G be an embedded canonical graph. Let G' be a contracted version of G. Let S be a strongly connected component in G. Let X be the external hole of S. Let S' and X' be the versions of S and X in G'. Let D be the set of boundary edges of X. Let D' be the set of edges in D that remain in G'.

LEMMA 6.5. *The subholes of X' can be partioned into two collections Δ_{new} and Δ_{old} with the following properties:*

1. *The subholes in Δ_{new} contain no edges from D'. These subholes are all subfaces in G', and each of them has the same orientation in both contexts.*

2. *The subholes in Δ_{old} consist of exactly the edges in D', and each edge in D' has the same orientation with respect to X and X'.*

Proof. See the above discussion. □

6.3. Contraction procedures. The planar strongly connected component algorithm uses three high-level procedures for vertex contraction: ContractPositive, ContractNegative, and ContractBoth.

ContractPositive is described in Algorithm 6.

ALGORITHM 6. *Procedure for contracting positive subfaces.*

```
Procedure ContractPositive
Input: an embedded canonical graph $G$.
Output: an embedded canonical graph that is a contracted version of $G$ with no positive
subfaces.
begin
    repeat
        1. Expand into a negative face each large-degree vertex of $G$ that is not on any
           positive subfaces.
        2. Contract each positive territory of $G$ into a single vertex.
        3. Contract back into single vertices the vertices in $G$ that are duplicates of the
           vertices expanded at Step 1.
    until $G$ has no positive subfaces.
    return the resulting $G$.
end.
```

ContractNegative is obtained by reversing all orientations in the description of ContractPositive. ContractBoth is simply the composition of ContractNegative and ContractPositive, i.e., ContractBoth(G) = ContractNegative(ContractPositive(G)).

Note that the output specifications of these procedures contain some subtleties that are prone to overexpectations and may result in mistaken theorems or proofs. In particular, although the output graph of ContractNegative has no negative subface, it may have positive subfaces even if its input graph has no positive subfaces. Consequently, ContractBoth(G) may have positive subfaces or even compact strongly connected components that contain positive subfaces.

The following analysis focuses on ContractPositive. A similar analysis for ContractNegative can be obtained by symmetry.

The next lemma examines the size of G at different steps of ContractPositive and measures the progress made by an iteration of the repeat loop in ContractPositive.

LEMMA 6.6. *At a given iteration of the repeat loop in* ContractPositive *let G_0 be the input graph to Step* 1. *For $i \in \{1,2,3\}$ let G_i be the output graph from Step i. Let n be the initial size of G. For $i \in \{0,1,2,3\}$ let n_i be the size of G_i and let β_i be the number of positive clusters in G_i. The following statements are true:*

1. $n_i \leq 5n$ *for* $i \in \{0,1,2,3\}$.

2. $\beta_3 \leq \beta_0/2$.

Proof. The first statement is obtained by induction by using the fact that $n_1 \leq 5n_0$, $n_2 \leq n_1$, and $n_3 \leq n_0$. To prove the second statement, it suffices to show that $\beta_1 = \beta_0$, $\beta_2 \leq \beta_1/2$, and $\beta_3 \leq \beta_2$ as follows.

$\beta_1 = \beta_0$. This is true because Step 1 does not create, destroy, split, or merge positive clusters.

$\beta_2 \leq \beta_1/2$. It suffices to show that each positive subface in G_2 contains at least two contracted positive territories of G_1. To prove by contradiction, assume that there is a positive subface g_2 in G_2 that contains at most one contracted positive territory of G_1. If g_2 contains no contracted positive territory of G_1, then g_2 is a positive subface in G_1, contradicting the fact that every positive cluster of G_1 is contracted at Step 2. Thus, without loss of generality, assume that g_2 contains exactly one vertex u that is a contracted positive territory of G_1. Consider g_2 as a path P_2 from u to u. By Lemma 6.3 the path P_2 is originally a positive subpath P_1 of a noncycle subface in G_1 such that the endpoints of P_1 are in positive territories of G_1 but its internal vertices are not. Because G_2 has no loop edges, P_1 has at least one internal vertex. By Theorem 5.8 the endpoints of P_1 are in different positive territories of G_1, contradicting the proof assumption that g_2 contains only one contracted positive territory of G_1.

$\beta_3 \leq \beta_2$. By Lemma 6.2 each negative face created by vertex expansion at Step 1 remains unchanged, becomes a collection of smaller negative subfaces, or disappears altogether in G_2. Step 3 in effect contracts these remaining negative subfaces. Contracting a negative subface does not create new positive subfaces and may merge old positive clusters. Therefore, the desired inequality is true. □

THEOREM 6.7. ContractPositive *correctly computes an embedded canonical graph that is a contracted version of the input graph with no positive subfaces. The procedure runs in* $O(\log^2 n)$ *time with* $n/\log n$ *processors for an input graph of size* n.

Proof. Steps 2 and 3 of ContractPositive ensure that the output graph is a contracted version of the input graph. Moreover, the output graph is an embedded canonical graph because vertex contraction preserves connectivity and bridgelessness. The termination condition of the repeat loop ensures that the output graph contains no positive subfaces.

Because G has at most n positive clusters, by Lemma 6.6(2) in $\lceil \log_2(n+1) \rceil$ iterations of the repeat loop G has no positive cluster and thus no positive subface. For the remainder of the proof, the following analysis shows that Steps 1 through 3 and the termination condition can all be done in $O(\log n)$ time with $n/\log n$ processors. Step 1 is performed by means of Theorem 5.1(2) and Lemmas 5.2(2) and 6.6(1). Next, by Theorem 5.1(1) and by the effect of vertex expansion the graph produced by Step 1 is an embedded canonical graph with no large-degree vertices except possibly in positive clusters. Thus Step 2 can be performed by means of Lemma 6.6(1) and Theorems 5.7 and 6.1. Step 3 is performed by means of Theorem 6.1 and an optimal parallel algorithm for planar connectivity [14]. The termination condition is tested by means of Lemmas 5.2(2) and 6.6(1). □

6.4. Contraction effects. Section 6.4.1 examines the effects of vertex contraction on compact strongly connected components. Section 6.4.2 analyzes the effects on singular components and their contracted versions, *singular* vertices. These components and vertices are closely related to index-0 and index-1 components. Section 6.4.3 details the effects on strongly connected components that each have exactly one nonempty internal hole. These components can be derived from index-1 components.

6.4.1. Effects of contraction on compact components.

THEOREM 6.8. *Let G be an embedded canonical graph. Then every compact strongly connected component of G becomes a single vertex in* ContractBoth(G).

Remark. This theorem is false if the loop edges created by contraction are kept.

Proof. Let G_1 = ContractPositive(G). Let G_2 = ContractNegative(G_1). Let S be a compact strongly connected component in G. Let S_1 and S_2 be the versions of S in G_1 and G_2. Let X_1 and X_2 be the external holes of S_1 and S_2. The goal is to show that S_2 is a single vertex. To prove by contradiction, assume that S_2 is not a single vertex. Then S_1 is also not a single vertex.

S_1 can be regarded both as a subgraph of G_1 and as a strongly connected component. Because S is compact in G, by Lemma 6.4 the component S_1 is compact in G_1. Thus every internal subface of the subgraph S_1 is an internal subface of G_1 with the same orientation in both contexts. Similarly, the boundary of the external face of the subgraph S_1 is the boundary of the external hole of the component S_1 with the same orientation in both contexts.

Because the subgraph S_1 is strongly connected and has at least two vertices, by Corollary 3.3 it has at least one positive face and one negative face. Because G_1 has no positive subface, the subgraph S_1 has no internal positive subface and thus no internal positive face. Therefore, the external face of the subgraph S_1 is a positive one, i.e., X_1 is a positive hole. By symmetry X_2 is a negative hole because G_2 has no negative subface. Then because G_2 is contracted from G_1, by Lemma 6.5(2) $\mathcal{B}(X_2)$ contains no edge from $\mathcal{B}(X_1)$. Next, by Lemma 6.5(1) each subhole Z of X_2 is also a subface in G_2 with the same orientation in both contexts. Consequently, Z is a negative subface in G_2, contradicting the fact that G_2 = ContractNegative(G_1). □

6.4.2. Effects of contraction on singular components and vertices. An *inward* (respectively, *outward*) *singular* vertex is one such that there is no directed path from any directed cycle to that vertex (respectively, from that vertex to any directed cycle). Singular vertices and components are related by means of the following theorem.

THEOREM 6.9. *Let G be an embedded canonical graph. Then every inward (respectively, outward) singular compact strongly connected component of G becomes an inward (respectively, outward) singular vertex in* ContractBoth(G).

Proof. By symmetry it suffices to prove the theorem for the inward case. Let G' = ContractBoth(G). Let S be an inward singular compact strongly connected component in G. Let S' be the version of S in G'. Because S is compact in G, by Theorem 6.8 S' is a single vertex in G'. To prove by contradiction, assume that S' is not an inward singular vertex in G'. Then G' contains a directed path from a directed cycle C' to the vertex S'. Consequently, by Theorem 6.8 C' must be part of a noncompact strongly connected component T in G. Furthermore, there is a directed path from T to S in G because vertex contraction preserves reachability. This contradicts the condition that S is an inward singular component in G. □

The reason for working with singular vertices is as follows. For computing the strongly connected components it is useful and easy to determine whether a vertex is a source or a sink or neither. If a vertex is a source or a sink, it is a strongly connected component by itself. Note that a source is inward singular and a sink is outward singular. Moreover, if the sources are removed, new sources may appear, and if the sinks are removed, new sinks may appear. The inward singular vertices are exactly the sources created by iteratively removing sources, and the outward singular vertices are exactly the sinks created by iteratively removing sinks. The iterative process of removing sources and sinks can be parallelized by means of the following theorem.

THEOREM 6.10. *Let G be an embedded canonical graph. Let $\tilde{G}$ be the dual of G. Let D be the set of edges in G incident with the outward and inward singular vertices. Let $\tilde{D}$ be the set of duals of the edges in D. Then the edges of $\tilde{D}$ all disappear in* ContractBoth($\tilde{G}$).

Proof. Let $\tilde{G}_1$ = ContractPositive($\tilde{G}$). Let $\tilde{G}_2$ = ContractNegative($\tilde{G}_1$). Let U_o and U_i be the sets of outward and inward singular vertices in G. Let D_o and D_i be the sets of edges in G incident with U_o and U_i. Let $\tilde{D}_o$ and $\tilde{D}_i$ be the sets of duals of the edges in D_o and D_i. Note that $\tilde{G}_2$ = ContractBoth($\tilde{G}$) and $\tilde{D} = \tilde{D}_o \cup \tilde{D}_i$. Because $\tilde{G}_2$ is contracted from $\tilde{G}_1$, it suffices to show that the edges of $\tilde{D}_o$ disappear in $\tilde{G}_1$ and that the edges of $\tilde{D}_i$ disappear in $\tilde{G}_2$. Because these two claims are symmetric, only the former claim is proved, as follows.

Because U_o as an induced subgraph of G is an acyclic directed graph, it can be regarded as a partial order. By induction on this partial order, D_o consists of the incoming edges to U_o in G. To prove by contradiction, assume that for some vertex $u \in U_o$ and for some incoming edge d of u in G, the dual edge $\tilde{d}$ of d remains in $\tilde{G}_1$. Without loss of generality, further assume that u is a minimal element in the partial order U_o with such a property.

By the minimality of u, $\tilde{G}_1$ contains no duals of the incoming edges of the descendants of u in U_o. Because the outgoing edges of u are incoming edges of its descendants in U_o, the duals of outgoing edges of u all disappear in $\tilde{G}_1$. In other words, only incoming edges of u may have duals remaining in $\tilde{G}_1$.

Let D' be the set of incoming edges of u in G whose duals remain in $\tilde{G}_1$. Let $\tilde{D}'$ be the set of duals of the edges in D'. Recall that the dual $\tilde{u}$ of the vertex u is a face in $\tilde{G}$. Because the edges in D' are incoming edges of u, the edges in $\tilde{D}'$ are positive boundary edges of $\tilde{u}$. Also, the edges in $\tilde{D}'$ are the only remaining boundary edges of $\tilde{u}$ in $\tilde{G}_1$. Therefore, by Lemma 6.2 $\tilde{D}'$ forms positive subfaces in $\tilde{G}_1$, contradicting the fact that $\tilde{G}_1$ = ContractPositive($\tilde{G}$). □

6.4.3. Effects of contraction on one-hole components. In the next lemma let G be an embedded canonical graph. Let G_1 = ContractPositive(G), and then let G_2 = ContractNegative(G_1). Let S be a strongly connected component in G with exactly one nonempty internal hole. Let S_1 and S_2 be the versions of S in G_1 and G_2.

LEMMA 6.11. *If S_2 is not compact in G_2, then the following statements hold:*

1. *S_1 and S_2 each have exactly one nonempty internal hole.*
2. *At least one of the nonempty internal holes of S_1 and S_2 is a cycle hole.*

Proof. Because S_2 is not compact in G_2, by Lemma 6.4 S_1 is not compact in G_1. Because S has exactly one nonempty internal hole, by Lemma 6.4 S_1 and S_2 each have exactly one nonempty internal hole. This proves the first statement.

To prove the second statement, let X_1 and X_2 be the external holes of S_1 and S_2. Let Y_1 and Y_2 be the nonempty internal holes of S_1 and S_2. The goal is to show that $\mathcal{B}(Y_1)$ or $\mathcal{B}(Y_2)$ forms a directed cycle. To prove by contradiction, assume that neither $\mathcal{B}(Y_1)$ nor $\mathcal{B}(Y_2)$ forms a directed cycle.

S_1 can be regarded both as a subgraph of G_1 and as a strongly connected component. The boundary of X_1 is that of the external face h of the subgraph S_1 with the same orientation in both contexts. The boundary Y_1 is that of an internal face k of the subgraph S_1 with the same orientation in both contexts. Because $\mathcal{B}(Y_1)$ does not form a directed cycle, k is not a positive face of the subgraph S_1. If a face f of the subgraph S_1 is neither h nor k, then f is also a face of G_1 with the same orientation in both contexts. Moreover, because G_1 has no positive subfaces, f is not a positive face of the subgraph S_1.

Because the subgraph S_1 is strongly connected with more than one vertex, by Corollary 3.3 it has at least one positive face. Thus h is a positive face of the subgraph S_1, i.e.,

X_1 is a positive hole. Symmetrically, X_2 is a negative hole because G_2 has no negative subfaces. Because $\mathcal{B}(X_1)$ and $\mathcal{B}(X_2)$ have the opposite orientations, by Lemma 6.5(2) $\mathcal{B}(X_2)$ contains no edges from $\mathcal{B}(X_1)$. By Lemma 6.5(1) $\mathcal{B}(X_2)$ forms negative subfaces of G_2, contradicting the fact that G_2 = ContractNegative(G_1). □

7. Flattening the nesting of strongly connected components.

As highlighted in §1, a major difficulty in computing planar strongly connected components is that these components may be nested within one another. This section details a procedure called FlattenNesting that breaks the nesting of strongly connected components by contracting and separating them. The procedure is detailed in Algorithm 7, and its overview and analysis are given below.

ALGORITHM 7. *Procedure for breaking the nesting of strongly connected components.*

```
Procedure FlattenNesting
Input: an embedded canonical graph $G_{\text{in}}$.
Output: an embedded planar directed graph $G_{\text{out}}$ with the following properties:
    1. $G_{\text{out}}$ is constructed from $G_{\text{in}}$ by edge cutting and vertex contraction and has the
       same strongly connected components as $G_{\text{in}}$ does.
    2. Each connected component of $G_{\text{out}}$ is an embedded canonical graph.
    3. If the strongly connected components of $G_{\text{in}}$ are not all compact, then the total
       number of noncompact strongly connected components in the connected com-
       ponents of $G_{\text{out}}$ is fewer than half that of $G_{\text{in}}$.
    4. Every singular compact strongly connected component of $G_{\text{in}}$ becomes an iso-
       lated vertex in $G_{\text{out}}$.
begin
    Stage 1: contracting compact strongly connected components.
        1-1. Let $G_1$ be ContractBoth($G_{\text{in}}$).
    Stage 2: cutting off singular vertices.
        2-1. Let $\tilde{G}_1$ be the dual graph of $G_1$.
        2-2. Let $\tilde{G}'_1$ be ContractBoth($\tilde{G}_1$).
        2-3. Let $D_2$ be the set of edges in $G_1$ whose duals disappear in $\tilde{G}'_1$.
        2-4. Let $G'_1$ be $G_1$ without the edges in $D_2$.
        2-5. Let $G_2$ be $G'_1$ without its bridges.
    Stage 3: creating chambers and cutting off chamber edges.
        3-1. Let $G'_2$ be the graph constructed by applying ContractPositive to every con-
             nected component of $G_2$.
        3-2. Let $G''_2$ be the graph constructed by applying ContractNegative to every con-
             nected component of $G'_2$.
        3-3. Let $D_3$ be the set of chamber edges in $G'_2$ and $G''_2$.
        3-4. Let $G'''_2$ be $G''_2$ without the edges in $D_3$.
        3-5. Let $G_{\text{out}}$ be $G'''_2$ without its bridges.
    return $G_{\text{out}}$.
end.
```

7.1. An overview of FlattenNesting.

In the planar strongly connected component algorithm FlattenNesting is iteratively applied to each connected component of its output graph.

The first output property of FlattenNesting ensures that the strongly connected components of the input graph remain the same throughout iterations of FlattenNesting. The second output property permits such iterations by ensuring that each connected component of an output graph is canonical and therefore may be an input to FlattenNesting. The third output property measures the progress of FlattenNesting in breaking the nesting of strongly connected components. The fourth output property measures the progress of FlattenNesting in contracting and separating strongly connected components.

Based on Theorem 4.2, FlattenNesting focuses on compact, index-0, and index-1 strongly connected components of G_{in}. These components correspond to the leaves, and the internal vertices of index 0 or 1 in the enclosure forest $\mathcal{F}(G_{\text{in}})$. They are processed in three stages as outlined below.

• Stage 1 contracts all compact strongly connected components of G_{in} into single vertices. Let G_1 be the resulting graph. Intuitively, this stage preprocesses the leaves of $\mathcal{F}(G_{\text{in}})$. It has one major effect:

– All singular compact strongly connected components of G_{in} become singular vertices in G_1. These components include all those enclosed by index-0 components in G_{in} and almost all those enclosed by index-1 components.

• Stage 2 cuts off the singular vertices in G_1. Let G_2 be the resulting graph. Intuitively, this stage cuts off the edges in $\mathcal{F}(G_{\text{in}})$ between the internal vertices with index 0 or 1 and their leaf children. This stage has three major effects:

– Every singular compact strongly connected component of G_{in} becomes an isolated vertex in G_2.

– Every index-0 strongly connected component of G_{in} becomes compact in a connected component of G_2.

– All remaining nonempty internal holes of index-1 strongly connected components of G_{in} become very similar to chambers in G_2.

• Stage 3 converts into chambers all remaining nonempty internal holes of index-1 components of G_{in} and then cuts off their chamber edges. Let G_{out} be the resulting graph. Intuitively, this stage cuts the edges in $\mathcal{F}(G_{\text{in}})$ between the index-1 internal vertices and their nonleaf children. This stage has one major effect:

– Every index-1 strongly connected component of G_{in} becomes compact in a connected component of G_{out}.

The three stages of FlattenNesting are analyzed in §§7.2, 7.3, and 7.4, respectively. These analyses are summarized in §7.5.

7.2. Analysis of Stage 1 of FlattenNesting.

LEMMA 7.1. *Let n be the size of G_{in}. The following statements are true:*

1. *Stage* 1 *of* FlattenNesting *runs in $O(\log^2 n)$ time with $n/\log n$ processors.*
2. *G_1 has the same strongly connected components as G_{in} does.*
3. *G_1 is an embedded canonical graph.*

Proof. This lemma follows directly from Theorem 6.7 and a symmetric theorem for ContractNegative. □

LEMMA 7.2. *The following statements are true:*

1. *Every compact strongly connected component of G_{in} becomes a single vertex in G_1.*
2. *Every singular compact strongly connected component of G_{in} becomes a singular vertex in G_1.*
3. *For each index-*0 *strongly connected component S of G_{in} every strongly connected component in the internal holes of S becomes a singular vertex in G_1.*
4. *Let T be an index-*1 *strongly connected component of G_{in}. Let X be the internal hole of T that contains a noncompact strongly connected component. Then the strongly connected components in internal holes of T except possibly X all become singular vertices in G_1.*

Proof. The first statement follows from Theorem 6.8, the second statement follows from Theorem 6.9, the third statement follows from Lemma 4.5 and Theorem 6.9, and the fourth statement follows from Theorem 6.9 and Lemma 4.6(1). □

7.3. Analysis of Stage 2 of FlattenNesting.

LEMMA 7.3. *Let n be the size of G_{in}. The following statements are true:*

1. *Stage* 2 *of* FlattenNesting *runs in $O(\log^2 n)$ time with $n/\log n$ processors.*

2. *G_2 has the same strongly connected components as G_1 does.*

3. *Every connected component of G_2 is an embedded canonical graph.*

Proof. The third statement is straightforward. The proofs of the other statements are detailed below.

Statement 1. Step 2-1 uses list ranking and prefix computation to process the doubly linked circular lists of the combinatorial embedding of G_1. This takes $O(\log n)$ time with $n/\log n$ processors. Step 2-2 is performed in $O(\log^2 n)$ time with $n/\log n$ processors by means of Lemma 7.1(3), Theorem 6.7, and a symmetric theorem for ContractNegative. Steps 2-3 and 2-4 are straightforward and take $O(\log n)$ time with $n/\log n$ processors. Step 2-5 is performed in $O(\log n)$ time with $n/\log n$ processors and an optimal parallel algorithm for planar biconnectivity [14]. Thus the total complexity is as stated.

Statement 2. By the second specification of vertex contraction, the dual of each edge in D_2 is in a strongly connected component of $\tilde{G}_1$. Then by Lemma 3.1 each edge in D_2 is between different strongly connected components of G_1. Therefore, G_1' and G_1 have the same strongly connected components. Next, because removing bridges does not change the strongly connected components of a directed graph, G_1' and G_2 have the same strongly connected components. Thus G_1 and G_2 have the same strongly connected components. □

LEMMA 7.4. *The following statements are true:*

1. *Every singular vertex of G_1 is isolated in G_2.*

2. *Every index-*0 *strongly connected component of G_{in} becomes compact in a connected component of G_2.*

3. *Every index-*1 *strongly connected component of G_{in} either becomes compact or has exactly one nonempty internal hole in a connected component of G_2.*

4. *Let S be an index-*1 *strongly connected component in G_{in}. Let S_2 be the version of S in G_2. Then the edges in $\mathcal{E}_{\text{in}}(S_2)$ either all point from S_2 or all point to S_2.*

Proof. The statements are shown as follows.

Statement 1. By Theorem 6.10 D_2 includes all edges in G_1 that are incident with singular vertices. Thus all singular vertices of G_1 become isolated in G_1'. Because G_2 is G_1' without its bridges, the singular vertices of G_1 remain isolated in G_2.

Statement 2. The proof readily follows from the first statement of this lemma and Lemmas 7.2(3) and 6.4.

Statement 3. The proof readily follows from the first statement of this lemma and Lemmas 7.2(4) and 6.4.

Statement 4. Because G_2 is obtained from G by edge cutting and vertex contraction, by Lemma 6.4 $\mathcal{E}_{\text{in}}(S_2) \subseteq \mathcal{E}_{\text{in}}(S)$. Note that $\mathcal{E}_{\text{in}}(S) - \mathcal{E}_{\text{ns}}(S)$ consists of the edges between S and the singular compact strongly connected components enclosed in S. By Lemma 7.2(2) and the first statement of this lemma these edges are cut in Stage 2. Thus $\mathcal{E}_{\text{in}}(S_2) \subseteq \mathcal{E}_{\text{ns}}(S)$. By Lemma 4.6(2) this statement is true. □

7.4. Analysis of Stage 3 of FlattenNesting.

LEMMA 7.5. *Let n be the size of G_{in}. The following statements are true:*

1. *Stage* 3 *of* FlattenNesting *runs in $O(\log^2 n)$ time with $n/\log n$ processors.*

2. *G_{out} has the same strongly connected components as G_2 does.*

3. *Every connected component of G_{out} is an embedded canonical graph.*

Proof. The third statement is straightforward. The proofs of the other statements are detailed below.

Statement 1. Because the contraction procedures are applied to disjoint parts of G, by Lemma 7.3(3) and Theorem 6.7 and by symmetry Steps 3-1 and 3-2 take $O(\log^2 n)$ time and $n/\log n$ processors. By Theorem 5.12 and by symmetry Step 3-3 takes $O(\log n)$ time and $n/\log n$ processors. Step 3-4 can obviously be performed in $O(\log n)$ time and with $n/\log n$ processors. Step 3-5 is performed in $O(\log n)$ time with $n/\log n$ processors by using an optimal parallel algorithm for planar biconnectivity [14]. Thus the complexity of Stage 3 is as stated.

Statement 2. The proof follows from the fact that a chamber edge in D_3 is between different strongly connected components of G_2 and that a bridge in G_2''' is between different strongly connected components of G_2. □

LEMMA 7.6. *Every index-1 strongly connected component of G_{in} becomes compact in a connected component of G_{out}.*

Proof. Let S be an index-1 strongly connected component in G_{in}. Let S_2 be the version of S in G_2. Let H_2 be the connected component of G_2 that contains S_2. Let $H_2' = \text{ContractPositive}(H_2)$. Let $H_2'' = \text{ContractNegative}(H_2')$. Let S_2' and S_2'' be the versions of S_2 in H_2' and H_2''.

To prove the lemma, by Steps 3-3 and 3-4 of FlattenNesting it suffices to show that at least one of the following statements is true:

1. S_2'' is compact in H_2''.
2. S_2' has exactly one nonempty internal hole, and this hole is a chamber in H_2'.
3. S_2'' has exactly one nonempty internal hole, and this hole is a chamber in H_2''.

For proof of this claim it suffices to show that if S_2'' is not compact in H_2'', then either the second statement or the third statement is true. Because S_2'' is not compact in H_2'' and because S is index-1, by Lemmas 7.4(3) and 6.4 S_2' and S_2'' have exactly one nonempty internal hole each. Let X' and X'' be the nonempty internal holes of S_2' and S_2''. Then by Lemma 6.11 either $\mathcal{B}(X')$ is a directed cycle or $\mathcal{B}(X'')$ is a directed cycle. Futhermore, because $\mathcal{E}_{X'}(S_2') \subseteq \mathcal{E}_{\text{in}}(S_2)$, by Lemma 7.4(4) $\mathcal{E}_{X'}(S_2')$ is unidirectional. Similarly, $\mathcal{E}_{X''}(S_2'')$ is unidirectional. Thus either X' is a chamber in H_2' or X'' is a chamber in H_2''. □

7.5. Summary analysis of FlattenNesting.

THEOREM 7.7. FlattenNesting *correctly computes an output graph as specified. It runs in $O(\log^2 n)$ time with $n/\log n$ processors for an input graph of size n.*

Proof. The complexity of FlattenNesting follows from Lemmas 7.1(1), 7.3(1), and 7.5(1). The correctness of FlattenNesting is shown as follows. Property 1 of the output follows from Lemmas 7.1(2), 7.3(2), and 7.5(2). Property 2 of the output follows from Lemma 7.5(3). Property 3 of the output follows from Theorem 4.2 and Lemmas 6.4, 7.4(2), and 7.6. Property 4 of the output follows from Lemmas 7.2(2) and 7.4(1). □

8. Computing strongly connected components.

The main results of this paper are stated in the next two theorems. One is for embedded planar graphs, and the other is for planar graphs without given embeddings.

8.1. Strongly connected components of graphs with embeddings.

THEOREM 8.1. *Let G be an embedded canonical graph. Let n be the size of G. Let α be the number of noncompact strongly connected components in G. Then Algorithm 8 correctly computes the strongly connected components of G in $O(\lceil \log_2(\alpha + 2)\rceil \cdot \log^2 n)$*

*time with $n/\log n$ processors on a deterministic arbitrary-*CRCW PRAM. *Note that because $\alpha < n$, the time complexity is $O(\log^3 n)$.*

ALGORITHM 8. *Procedure for computing the strongly connected components of an embedded planar graph.*

```
Procedure ComputeSCC
Input: an embedded canonical graph $G$.
Output: the strongly connected components of $G$.
begin
  1. repeat
         Apply FlattenNesting to each connected component of $G$.
     until every vertex in $G$ is isolated.
  2. Expand every vertex in the current $G$ into a strongly connected component of the
     initial $G$.
  3. return the strongly connected components computed above.
end.
```

Proof. The correctness of ComputeSCC is proved as follows. Because G is bridgeless, it can be input to FlattenNesting at the beginning of Step 1. Then by the second output property of FlattenNesting Step 1 can iteratively apply FlattenNesting to each connected component of its output graphs. By the first output property of FlattenNesting the strongly connected components of G remain the same throughout the iterative applications of FlattenNesting. Therefore, at the end of Step 1 every isolated vertex is indeed contracted from an original strongly connected component of G. Consequently, Step 2 can expand each isolated vertex into an original strongly connected component of G.

The complexity of ComputeSCC is analyzed as follows. Step 2 of the algorithm can be performed by tree contraction in $O(\log n)$ time with $n/\log n$ processors. Thus it suffices to show that Step 1 takes $O(\lceil\log_2(\alpha+2)\rceil\cdot\log^2 n)$ time with $n/\log n$ processors.

By the third output property of FlattenNesting, within $\lceil\log_2(\alpha/2+1)\rceil$ iterations of FlattenNesting at Step 1 the strongly connected components of G all become compact in their respective connected components of the current G. The absence of noncompact strongly connected components ensures that all strongly connected components of G are singular in their respective connected components. Thus by the fourth output property of FlattenNesting, with one more application of FlattenNesting the strongly connected components of G all become isolated vertices. Therefore, the repeat loop of Step 1 iterates at most $\lceil\log_2(\alpha+2)\rceil$ times.

Because at each iteration of the repeat loop at Step 1 FlattenNesting is applied to disjoint parts of G, the connected components of G at each iteration of the repeat loop can be computed in $O(\log n)$ with $n/\log n$ processors [14]. Similarly, by Theorem 7.7 the body of the repeat loop takes $O(\log^2 n)$ time and $n/\log n$ processors. Checking the termination condition of the repeat loop takes only $O(\log n)$ time and $n/\log n$ processors. Thus each iteration of the repeat loop takes $O(\log^2 n)$ time and $n/\log n$ processors, and the complexity of Step 1 is as desired. □

8.2. Strongly connected components of graphs without embeddings.

THEOREM 8.2. *Let G be a planar directed graph of size n. Then Algorithm 9 correctly computes the strongly connected components of G in $O(\log^3 n)$ time with $n/\log n$ processors on a deterministic arbitrary-*CRCW PRAM.

ALGORITHM 9. *Procedure for computing the strongly connected components of a planar directed graph.*

```
Procedure ComputeStronglyConnectedComponents
Input: a planar directed graph $G$.
Output: the strongly connected components of $G$.
begin
    1. Delete all bridges from $G$.
    2. Compute a planar embedding for each connected component of $G$.
    3. Compute the strongly connected components of $G$ by applying the procedure in
       Algorithm 8 to each connected component of $G$.
    4. return the strongly connected components computed above.
end.
```

Proof. The correctness of Algorithm 9 follows from Theorem 8.1 and the fact that a strongly connected component contains no bridges. The complexity of the algorithm is analyzed as follows. Step 1 takes $O(\log n)$ time and $n/\log n$ processors when an optimal parallel algorithm for planar biconnectivity is used [14]. Step 2 takes $O(\log n \log\log n)$ time and $n/\log n$ processors [24]. Because ComputeSCC is applied to disjoint parts of G, by Theorem 8.1 Step 3 takes $O(\log^3 n)$ time and $n/\log n$ processors. Thus the total complexity is as stated. □

9. Open problems. This paper has shown that for a planar directed graph of size n the strongly connected components can be computed in $O(\log^3 n)$ time with $n/\log n$ processors. A subsequent paper will show that given a strongly connected planar directed graph of size n, a directed spanning tree rooted at a specified vertex can be built in $O(\log^2 n)$ time with $n/\log n$ processors. There are several fundamental problems left open in this paper. Perhaps the most important is to efficiently compute planar breadth-first search. Currently, the best algorithm for this problem is given by Pan and Reif [23]. Their algorithm computes the shortest paths in $O(\log^2 n)$ time with $n^{1.5}/\log n$ processors for undirected planar graphs. The algorithm is based on matrix operations and uses a randomized planar separator algorithm given by Gazit and Miller [10]. It would be significant to reduce to linear the processor complexity of planar breadth-first search.

Acknowledgments. The author wishes to thank Philip Klein, Gary Miller, Vijaya Ramachandran, and Greg Shannon for many helpful discussions and suggestions and to thank Fang Wan for helping to prepare some of the figures.

REFERENCES

[1] K. ABRAHAMSON, N. DADOUN, D. G. KIRKPATRICK, AND T. PRZYTYCKA, *A simple tree contraction algorithm*, J. Algorithms, 10 (1989), pp. 287–302.

[2] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[3] R. J. ANDERSON AND G. L. MILLER, *Deterministic parallel list ranking*, Algorithmica, 6 (1991), pp. 859–868.

[4] C. BERGE, *Graphs*, 2nd revised ed., North-Holland, New York, 1985.

[5] B. BOLLOBÁS, *Graph Theory*, Springer-Verlag, Berlin, New York, 1979.

[6] R. COLE AND U. VISHKIN, *The accelerated centroid decomposition technique for optimal tree evaluation in logarithmic time*, Algorithmica, 3 (1988), pp. 329–346.

[7] ———, *Faster optimal prefix sums and list ranking*, Information and Computation, 81 (1989), pp. 334–352.

[8] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[9] T. H. CORMEN, C. L. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.

[10] H. GAZIT AND G. L. MILLER, *A parallel algorithm for finding a separator in planar graphs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 238–248.

[11] ———, *An improved parallel algorithm that computes the BFS numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.

[12] H. GAZIT, G. L. MILLER, AND S. H. TENG, *Optimal tree contraction in the EREW model*, in Concurrent Computations: Algorithms, Architecture, and Technology, S. T. Dickinson, B. W. Dickinson, and S. Schwartz, eds., Plenum Press, New York, 1988, pp. 139–156.

[13] A. M. GIBBONS AND W. RYTTER, *An optimal parallel algorithm for dynamic expression evaluation and its applications*, in Proceedings of the 6th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 241, Springer-Verlag, Berlin, New York, 1986, pp. 453–469.

[14] T. HAGERUP, *Optimal parallel algorithms on planar graphs*, Inform. Comput., 84 (1990), pp. 71–96.

[15] Y. HAN, *An optimal linked list prefix algorithm on a local memory computer*, IEEE Transactions on Computers, 40 (1991), pp. 1149–1153.

[16] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[17] M.-Y. KAO AND G. E. SHANNON, *Linear-processor NC algorithms for planar directed graphs* II: *Directed spanning trees*, SIAM J. Comput., 22 (1993), pp. 460–481.

[18] R. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, New York, 1990, pp. 869–941.

[19] S. R. KOSARAJU AND A. L. DELCHER, *Optimal parallel evaluation of tree-structured computations by raking*, in Proceedings of the 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures, J. H. Reif, ed., Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988, pp. 101–110.

[20] C. P. KRUSKAL, L. RUDOLPH, AND M. SNIR, *The power of parallel prefix*, IEEE Trans. Comput., C-34 (1985), pp. 965–968.

[21] R. E. LADNER AND M. J. FISCHER, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980), pp. 831–838.

[22] G. L. MILLER AND J. H. REIF, *Parallel tree contractions and its applications*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 478–489.

[23] V. PAN AND J. H. REIF, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.

[24] V. RAMACHANDRAN AND J. H. REIF, *An optimal parallel algorithm for graph planarity*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 282–287.

[25] W. TUTTE, *Graph Theory*, Vol. 21 of Encyclopedia of Mathematics and Its Applications, Addison-Wesley, Reading, MA, 1984.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 460–481, June 1993

LINEAR-PROCESSOR NC ALGORITHMS FOR PLANAR DIRECTED GRAPHS II: DIRECTED SPANNING TREES*

MING-YANG KAO† AND GREGORY E. SHANNON‡

Abstract. It is a fundamental open problem whether polylogarithmic time and a *linear* number of processors are sufficient for computing the strongly connected components of a directed graph and constructing directed spanning trees for these components. This paper provides the first nontrivial partial solution to the tree problem: for a strongly connected planar directed graph of size n a directed spanning tree rooted at a specified vertex can be computed in $O(\log^2 n)$ time with $n/\log n$ processors. This result complements an algorithm by Kao that computes the strongly connected components of a planar directed graph in $O(\log^3 n)$ time with $n/\log n$ processors. Both algorithms run on a deterministic parallel random-access machine that permits concurrent reads and concurrent writes in its shared memory and, in case of a write conflict, allows an arbitrary processor to succeed.

Key words. linear-processor NC algorithms, directed spanning trees, strong connectivity, planar directed graphs, planar orientation, vertex expansion, vertex contraction, edge cutting, tree rerooting, duplicate removal

AMS(MOS) subject classifications. 68Q22, 05C99

1. Introduction.

The problems of finding directed spanning trees and strongly connected components frequently appear in more complex problems involving directed graphs. In sequential computation both problems have optimal linear-time algorithms [2], [10]. In parallel computation on a parallel random-access machine (PRAM), the best algorithms for these two problems are based on matrix multiplication and require $O(\log^2 n)$ time and $n^{2.376}$ processors for an n-vertex directed graph [8], [12], [17].

The work of an algorithm can be estimated by the product of its time and processor complexities. Thus there is a substantial gap between the work done by the above sequential and parallel algorithms. In light of the gap it is a fundamental problem whether the strongly connected components and their directed spanning trees can be computed in polylogarithmic time with a *linear* number of processors.

This paper offers the first nontrivial partial solution to the spanning-tree problem: for a strongly connected planar directed graph of size n, a directed spanning tree rooted at a given vertex can be computed in $O(\log^2 n)$ time with $n/\log n$ processors. This result complements an algorithm by Kao that computes the strongly connected components of a planar directed graph in $O(\log^3 n)$ time with $n/\log n$ processors [18]. Both algorithms run on a deterministic PRAM that allows concurrent reads and concurrent writes (CRCW) in its shared memory and, in case of a write conflict, permits an arbitrary processor to succeed.

An insight used in the tree algorithm is as follows. A *cycle* face of the input graph is one whose boundary forms a directed cycle. A *positive* face is a cycle face whose boundary cycle runs clockwise. If the graph has only one positive face and its maximum degree is at most three, then a directed spanning tree is obtained by deleting a boundary

*Received by the editors April 4, 1990; accepted for publication (in revised form) February 26, 1992. An extended abstract of this work appeared as one of the two parts of *Local Reorientation, Global Order, and Planar Topology*, in Proc. 21st Annual Association for Computing Machinery Symposium on Theory of Computing, Seattle, Washington, May 15–17, 1989, pp. 286–296.

†Department of Computer Science, Duke University, Durham, North Carolina 27706. This research was supported in part by National Science Foundation grants CCR-8909323 and CCR-9101385. Part of this work was done while the author was at the Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

‡Department of Computer Science, Indiana University, Bloomington, Indiana 47405.

edge of the positive face and the *first* edge of each *positive segment* (a maximal clockwise-directed boundary path of a noncycle face); see Fig. 1.

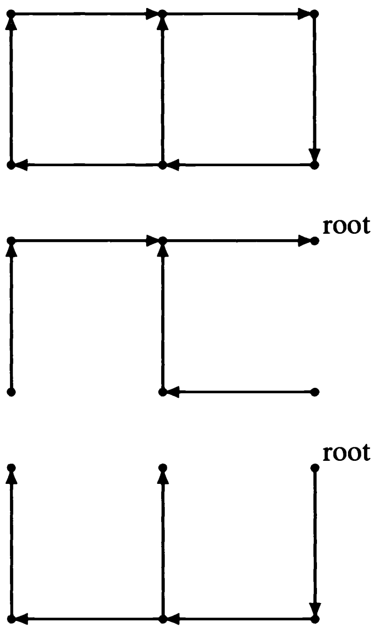


FIG. 1. *CD-pair of spanning trees obtained by edge cutting.*

The correctness of the technique derives from a fundamental orientation property: if an embedded planar directed graph is strongly connected, then its dual directed graph is acyclic. This acyclicity induces a partial order on the faces of the primal graph. This order, in turn, induces an ear decomposition for the primal graph with the boundary of the positive face being the first ear and the positive segments being the remaining ears. Then, for computation of a directed spanning tree for the primal graph, it suffices to remove an edge of the first ear and the first edge of each remaining ear.

A directed tree is *convergent* (respectively, *divergent*) if each edge points from a vertex to its parent (respectively, child). A *CD-pair* of spanning trees is a convergent spanning tree and a divergent spanning tree with the same root. The spanning tree constructed above is convergent. A divergent spanning tree can be computed by deleting a boundary edge of the positive face and the *last* edge of each positive segment. These two spanning trees form a CD-pair of spanning trees if the two boundary edges deleted from the positive face are chosen appropriately (see Fig. 1).

It is efficient to *reroot* this CD-pair, i.e., to compute another CD-pair of spanning trees rooted at a specified vertex. Therefore, the edge-cutting technique in effect computes a spanning tree with a specified direction and a specified root.

The edge-cutting technique fails in two cases. If the maximum degree of the input graph is four or greater, then the edge-cutting technique may produce a subgraph containing directed cycles (see Fig. 2). If the graph has two or more positive faces, then the edge-cutting technique may produce a CD-pair of spanning *forests* by deleting an appropriate pair of boundary edges of each positive face and the first and the last edge of each positive segment. (A CD-pair of spanning forests is a convergent spanning forest and a divergent spanning forest with the same roots.) (See Fig. 3.)

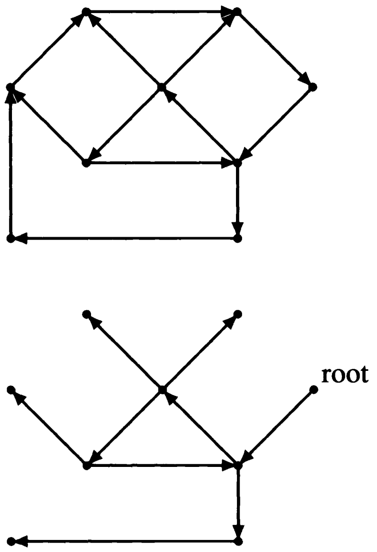


FIG. 2. *Effect of large-degree vertices on edge cutting.*

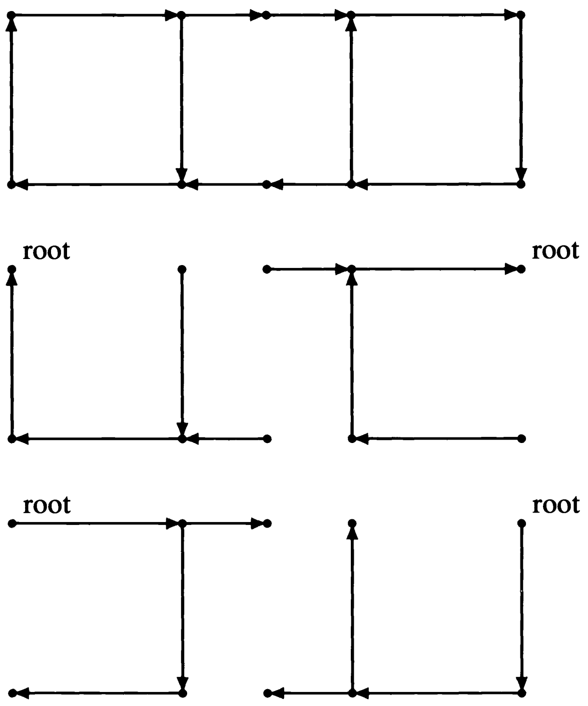


FIG. 3. *Effect of multiple positive faces on edge cutting.*

The tree algorithm resolves these two difficult cases by reducing the input graph to one with exactly one positive face and a maximum degree at most three.

Reducing the number of positive faces is coupled with merging the CD-pair of spanning forests produced by the edge-cutting technique. Let T_1 and T_2 be two disjoint divergent trees. Let $d = u_1 \rightarrow u_2$ be a directed edge from T_1 to T_2. Merging T_2 and T_1

through d involves two cases. If u_2 is the root of T_2, then a larger divergent tree is readily formed by d, T_1, and T_2. Otherwise, merging T_2 to T_1 through d requires changing the root of T_2 to u_2.

A key property of the CD-pair of spanning forests produced by the edge-cutting technique is that the boundary of each positive face is covered by a unique convergent tree and a unique divergent tree. These trees cover different vertices but share a well-structured neighborhood of that positive face. Their common root and opposite orientations may be used to reroot them efficiently at any vertex in the shared neighborhood. Moreover, this neighborhood can be contracted to reduce by half the number of positive faces. Of course, vertex contraction may produce large-degree vertices, and the two difficult cases of the edge-cutting technique must be resolved in concert.

To remove large-degree vertices, the spanning-tree algorithm expands them each into a cycle face (see Fig. 4). Let G be the input graph. Let G' be obtained from G by expanding the large-degree vertices into cycle faces. The tree algorithm first computes a spanning tree T' for G'. Then the problem is that T' contains several duplicates for each large-degree vertex of G. The tree algorithm uses a duplicate-removal technique to convert T' into a spanning tree of G.

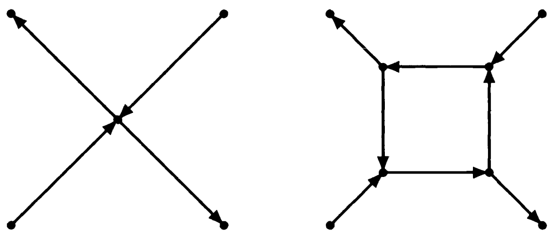


FIG. 4. *Example of expanding a large-degree vertex into a negative face.*

This paper is organized as follows. Section 2 gives basics of planar directed graphs and CD-pairs of trees, and §3 describes the orientation structure of primal and dual planar directed graphs. Section 4 presents the vertex expansion technique, §5 discusses the vertex-contraction technique, §6 analyzes the edge-cutting technique, and §7 elaborates on the techniques of tree rerooting and duplicate removal. Section 8 uses the preceding techniques to design the directed spanning tree algorithm, and §9 concludes with open problems.

2. Basics of planar directed graphs and CD-pairs of trees.

The following definitions and facts are useful.

2.1. CD-pairs of trees and forests.

A *divergent* (respectively, *convergent*) tree is a directed tree in which every edge points from a vertex to its child (respectively, parent). A *divergent* (respectively, *convergent*) forest is a directed forest in which every tree is divergent (respectively, convergent).

A CD-pair of trees is a convergent tree and a divergent tree with the same root. A CD-pair of spanning trees for a directed graph G is a CD-pair of trees that are also spanning trees of G.

A *covering* tree of a vertex subset U is a tree of G that contains at least the vertices in U. A CD-pair of covering trees for U is a CD-pair of trees that are also covering trees of U.

A CD-pair of forests is a convergent forest and a divergent forest with the same roots. A CD-pair of spanning forests for G is a CD-pair of forests that are also spanning forests of G.

2.2. Normal graphs. A *planar* directed graph is one that can be embedded on a sphere such that in the embedding the edges intersect only at common endpoints [5], [6], [15], [24]. An *embedded* planar directed graph is one with a given spherical embedding.

A *normal* graph is a planar directed graph that is strongly connected, has at least two vertices and may have multiple edges but has no loop edges.

Note that both multiple edges and loop edges can be deleted to simplify a strongly connected directed graph without affecting the existence of directed spanning trees. In this paper, however, the multiple edges created by vertex contraction must be kept in order to manage contraction invariants. For this reason most discussions in this paper focus on normal graphs.

2.3. Combinatorial embeddings and data structures. Let G be a normal graph. Algorithmically, a spherical embedding of G is encoded by the clockwise cyclic order of the edges incident with each vertex. Such an encoding is called a *combinatorial* spherical embedding of G. Topologically, a spherical embedding is uniquely specified by its corresponding combinatorial embedding.

The cyclic edge incidence in a combinatorial embedding is further encoded by the following data structure: for each vertex there is a doubly linked circular list consisting of the edges incident with that vertex in the clockwise order.

Given a normal graph of size n, a combinatorial spherical embedding can be computed in $O(\log n)$ time with $n \log\log n/\log n$ processors on a deterministic arbitrary-CRCW PRAM [26].

2.4. Faces, boundaries, orientations, cycle faces, subfaces, and clusters. Let G be an embedded normal graph. If the vertices and edges of G are deleted from its embedding sphere, then the sphere is divided into disconnected regions. Each region is called a *face* of G.

Let f be a face of G. The *boundary* of f, denoted by $\mathcal{B}(f)$, is the set of edges and vertices surrounding f. $\mathcal{B}(f)$ can be arranged into a unique edge-simple undirected cycle by having an observer stay inside f and walk around $\mathcal{B}(f)$ once. This cycle is called the *boundary* cycle of f. A *boundary* path of f is a subpath of its boundary cycle.

Let d be a boundary edge of f. The *orientation* of d with respect to f is defined as follows: the edge d is *positive* (respectively, *negative*) with respect to f if it points in the clockwise (respectively, counterclockwise) direction on the boundary cycle of f with respect to an observer standing inside f.

A face is called a *cycle* face if its boundary cycle is a directed cycle. A cycle face f is called a *positive* (respectively, *negative*) face if every boundary edge of f is positive (respectively, negative) with respect to f.

A *subface* of a face f is a vertex-simple undirected subcycle of the boundary cycle of f. The boundary cycle of f can be partition into subfaces. Furthermore, if f is a cycle face, then its subfaces are all directed cycles.

Two positive (respectively, negative) faces g_1 and g_2 of G are *linked* if there is a sequence of positive (respectively, negative) faces $f_1, \ldots, f_s$ such that $g_1 = f_1$, $g_2 = f_s$, and for $1 \leq i \leq s-1$, f_i and f_{i+1} share at least one boundary vertex.

A *positive* (respectively, *negative*) *cluster* of G is the set of boundary cycles of a maximal family of linked positive (respectively, negative) faces.

The planar directed spanning tree algorithm is described mainly in terms of positive faces and clusters. By symmetry the algorithm can also be described by using mainly the negative orientation.

LEMMA 2.1. *Let G be an embedded normal graph of size n. The following can be computed in $O(\log n)$ time with $n/\log n$ processors:*

1. *the boundary cycles of the faces,*
2. *the boundary cycles of the positive faces,*
3. *the positive clusters, and*
4. *the subfaces of the positive faces.*

Proof. The boundary cycles of the faces can be computed by tracing the doubly linked circular lists of the combinatorial embedding of G. This is done by list ranking [3], [8], [16] in $O(\log n)$ time with $n/\log n$ processors. Next it is straightforward to determine the boundary cycles of the positive faces in $O(\log n)$ time with $n/\log n$ processors. Then the positive clusters can be computed in $O(\log n)$ time with $n/\log n$ processors by using an optimal parallel algorithm for planar connectivity [15].

To compute the subfaces of the positive faces, observe that the subfaces of a face f are just the biconnected components of $\mathcal{B}(f)$. Similarly, the subfaces of G are the face boundaries of the subgraphs that are induced by the biconnected components of G. On the basis of these observations the subfaces of G can be computed in a straightforward manner in $O(\log n)$ time with $n/\log n$ processors. This computation uses optimal algorithms for planar connectivity [15], planar biconnectivity [15], list ranking [3], [8], [16], and prefix computation [21], [22]. $\square$

2.5. Holes, boundaries, and orientations. Let G be an embedded normal graph. Let B be a connected subgraph of G with at least two vertices. If the vertices and edges of B are removed from the embedding sphere of G, then the sphere is divided into disconnected regions. Each region is called a *hole* of B.

Let X be a hole of B. The *boundary* of X, denoted by $\mathcal{B}(X)$, is the set of vertices and edges surrounding X. $\mathcal{B}(X)$ can be arranged into a unique undirected cycle by having an observer stay inside X and walk around $\mathcal{B}(X)$ exactly once. This cycle is called the *boundary* cycle of X. It may not be edge-simple.

Let d be a boundary edge of X. The *orientation* of d with respect to X is defined as follows: the edge d is *positive* (respectively, *negative*) with respect to X if it points in the clockwise (respectively, counterclockwise) direction on the boundary cycle of X with respect to an observer standing inside X.

3. Planar orientation. This section describes orientation structures of primal and dual planar directed graphs.

3.1. Dual planar directed graphs. Let G be an embedded normal graph. The *dual* of G, denoted by $\tilde{G}$, is the embedded normal graph constructed below:

• For each face f in G there is a vertex, denoted by $\tilde{f}$, in $\tilde{G}$ that corresponds to f. The vertex $\tilde{f}$ is called the *dual* of f.

• For each edge d in G there is an edge, denoted by $\tilde{d}$, in $\tilde{G}$ that corresponds to d. The edge $\tilde{d}$ is called the *dual* of d and is determined as follows. Let f_1 and f_2 be the two faces in G that share d as a boundary edge. If d is positive (respectively, negative) with respect to f_1, then $\tilde{d}$ is a directed edge from $\tilde{f}_1$ to $\tilde{f}_2$ (respectively, from $\tilde{f}_2$ to $\tilde{f}_1$).

• The spherical embedding of $\tilde{G}$ is derived from that of G as follows. For each face f of G the vertex $\tilde{f}$ is placed inside f. For each edge d of G the edge $\tilde{d}$ is obtained by turning d counterclockwise 90°.

Note that the dual of the dual of G is the same as G with all edge directions reversed. Consequently, each vertex in G also corresponds to a face in $\tilde{G}$. In particular, a source (respectively, sink) in $\tilde{G}$ corresponds to a positive (respectively, negative) cycle face in G.

3.2. Primal-dual orientations. The following theorem is crucial to the planar directed spanning-tree algorithm.

THEOREM 3.1. *Let G be an embedded normal graph. Then $\tilde{G}$ is acyclic. Consequently, G has at least one positive face and one negative face.*

Proof. For a detailed proof, see the companion paper on computing planar strongly connected components [18]. □

4. Vertex expansion. The planar directed spanning-tree algorithm is sensitive to vertices of degree greater than three. For this reason such vertices are called the *large-degree* vertices.

A useful operation for eliminating large-degree vertices is to replace them each by a positive face or a negative face, as shown in Fig. 4. This operation is called *vertex expansion*.

In another important application vertex expansion is used to destroy cycle faces and may apply to vertices of degree two. Moreover, for technical uniformity, if the graph in question consists of a single vertex, then this vertex is expanded into a cycle face with two boundary edges.

The procedure in Algorithm 1 details how to expand vertices of degree two or more into negative faces.

ALGORITHM 1. *Procedure for expanding vertices into negative faces.*

```
Procedure ExpandVertex
Input: an embedded normal graph $G$ and a set $U$ of vertices.
Output: $G$ with each vertex in $U$ expanded into a negative face.
begin
    for each vertex $v \in U$ do
        begin
            1. Let $d_1, \dots, d_s$ be the edges incident with $v$ in the counterclockwise order.
            2. Create $s$ copies of $v$, namely, $v_1, \dots, v_s$.
            3. Replace the $v$-end of each $d_i$ by $v_i$.
            4. Link the vertices $v_i$ into a counterclockwise directed cycle, i.e., $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{s-1} \rightarrow v_s \rightarrow v_1$.
        end.
    return $G$ with the above changes.
end.
```

The procedure for expanding vertices into positive faces can be obtained by symmetry.

THEOREM 4.1. *The following statements are true:*

1. *The output graph of* ExpandVertex *is an embedded normal graph.*

2. ExpandVertex *runs in $O(\log n)$ time with $n/\log n$ processors for an input graph of size n.*

Proof. The first statement is straightforward. As for the second statement, ExpandVertex can be done in a straightforward manner with fundamental techniques, including prefix computation [21], [22] and list ranking [3], [8], [16]. □

5. Vertex contraction. Throughout this paper vertex contraction always satisfies the following specifications:

• Vertex contraction keeps all multiple edges and deletes all loop edges that it creates.

• Vertex contraction contracts only connected vertex subsets of an embedded normal graph.

Contracting only connected vertex subsets preserves planarity. Deleting all loop edges simplifies a contracted graph. Keeping all multiple edges maintains crucial contraction invariants. In particular, each uncontracted edge keeps a unique identity because its link remains unchanged although its endpoints may be altered by contraction.

5.1. Spherical embeddings induced by vertex contraction. Let G be an embedded normal graph. Let B be the subgraph induced by a connected vertex subset of G. Let G' be the graph constructed from G by contracting B into a vertex B'. If B consists of at most one vertex, then G' and G are the same. If B consists of all the vertices of G, then G' is a single vertex. Otherwise, a spherical embedding for G' is specified as follows:

• For every vertex $v \notin B$ the clockwise cyclic order of the edges incident with v is the same in G and G'.

• The edges around each nonempty hole X of B stay together around B', and their clockwise cyclic order around B' is the same as their cyclic order around the boundary cycle of X in the negative direction of X.

In general, such a spherical embedding is not unique. For the purposes of this paper any spherical embedding that fits this construction is suitable.

THEOREM 5.1. *Given an embedded normal graph of size n, a spherical embedding induced by contracting a disjoint family of connected vertex subsets can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The edges around a new vertex of G' are collected by processing the doubly linked circular lists of the combinatorial embedding of G. The computation takes $O(\log n)$ time and $n/\log n$ processors, using optimal parallel algorithms for list ranking [3], [8], [16], prefix computation [21], [22], tree contraction [1], [10], [13], [18], [21], and planar connectivity [15]. □

5.2. Contraction invariants. Let G be an embedded normal graph. Let G' be a contracted version of G. Let f' be a face in G'. Let P' be a boundary path of f' whose internal vertices are original vertices in G.

LEMMA 5.2. *P' originally is a boundary path of a face f in G, and each edge of P' has the same orientation with respect to f and f'.*

Note. This lemma is false if multiple edges created by contractions are not kept.

Proof. Vertex contraction can be done by performing a sequence of contracting a nonloop edge and deleting a loop edge. The proof of the lemma is by straightforward induction and is based on direct observations about the effects of deleting a loop edge and contracting a nonloop edge. □

6. Edge cutting. Section 6.1 describes a procedure, called CutSegment, for the edge-cutting technique. Section 6.2 uses CutSegment to find a CD-pair of spanning trees for a positive cluster.

Section 6.3 uses CutSegment to compute a CD-pair of spanning forests for an embedded normal graph with no large-degree vertices. Section 6.4 uses these forests to find key neighborhoods of positive faces called *biconnected territories*. These neighborhoods are contracted in the planar directed spanning tree algorithms.

6.1. CutSegment. The procedure CutSegment is detailed in Algorithm 2, which is based on a generalization of the standard ear decomposition [23].

ALGORITHM 2. *Procedure for edge cutting.*

```
Procedure CutSegment
Input:
    1. a directed graph $G$,
    2. a symbolic parameter $x$ = "first" or "last,"
    3. the set $\Pi$ of ears in a multihead ear decomposition of $G$. Remark: The partial
       order of the decomposition is not part of the input.
Output: $G$ without the $x$ edge of each path in $\Pi$.
begin
    1. Delete from $G$ the $x$ edge of each path in $\Pi$.
    2. return the resulting $G$.
end.
```

Let G be a directed graph. An *ear* of G is an internally vertex-simple directed path, i.e., the internal vertices appear only once in the path. A *decomposition* of G is a partition of its edges into ears. An *ordered* decomposition is one whose ears are arranged into a partial order. A *head* is an ear that is a minimal element in the partial order. A *multihead ear decomposition* of G is an ordered decomposition with the following properties:

• The endpoints of each head are the same. Two distinct heads may not intersect at all.

• The endpoints of each nonhead ear must be in some smaller ears, but its internal vertices may not be in any smaller or incomparable ears.

Remark. A standard ear decomposition is a multihead ear decomposition with a total order [23].

THEOREM 6.1. *Let n be the size of G. Let G_c =* CutSegment(G, first, Π). *Also, let G_d =* CutSegment(G, last, Π). *Then the following statements are true:*

1. CutSegment *runs in $O(\log n)$ time with $n/\log n$ processors.*
2. *G_c and G_d form a* CD*-pair of spanning forests for G with the properties below:*
(a) *The vertices of each head are in exactly one tree in each forest.*
(b) *In each forest each tree contains the vertices of exactly one head.*
(c) *Each tree is rooted at the endpoint of its corresponding head.*

Proof. The first statement is straightforward. The second statement is proved by induction on the partial order of the ears. The idea is to add to an initially empty graph the ears one by one, starting with the heads. □

6.2. Computing spanning trees for positive clusters. Algorithm 3 is a procedure ComputeClusterSegments that computes the ears in a one-head ear decomposition for a positive cluster Φ.

ALGORITHM 3. *Procedure for computing an ear decomposition with one head for each positive cluster.*

```
Procedure ComputeClusterSegments
Input: a positive cluster $\Phi$ of an embedded normal graph $G$ that may have large-degree
vertices. Remark: $G$ is not part of the input.
Output: the set $\Pi$ of ears in a multihead ear decomposition of $\Phi$ with only one head.
begin
    1. Compute an auxiliary graph $\Phi'$ for $\Phi$ as follows:
        1-1. Let $\Gamma$ be the set of subfaces of the faces in $\Phi$.
        1-2. Let $U$ be the set of vertices shared by cycles in $\Gamma$.
        1-3. $\Phi'$ is the connected planar undirected graph defined below:
            • The vertex set of $\Phi'$ consists of the vertices in $U$ and the cycles in $\Gamma$ with
              each cycle treated as a single vertex.
            • For $C \in \Gamma$ and $u \in U$ there is an undirected edge in $\Phi'$ between $C$ and
              $u$ if and only if $u$ is a vertex in $C$.
    2. Impose a total order on the cycles in $\Gamma$ as follows:
      2-1. Compute an arbitrary undirected spanning tree $T$ of $\Phi'$.
      2-2. Compute a preorder numbering of $T$. Remark: This preorder numbering
           naturally induces a total order on the cycles in $\Gamma$.
    3. Use the above total order to construct $\Pi$:
      3-1. Compute the *breakpoints* of each cycle in $\Gamma$ defined as follows:
            • For the cycle with the smallest preorder number the *breakpoint* of that
              cycle is an arbitrary vertex in it.
            • For each other cycle the *breakpoints* are the vertices shared by that cycle
              and those with smaller preorder numbers.
           Remark: A breakpoint is defined with respect to a cycle. A vertex may be a
           breakpoint for a cycle but not for another.
      3-2. Compute the *segments* of each cycle $C$ in $\Gamma$ defined as follows:
            • If $C$ has only one breakpoint, then the *segment* of $C$ is the directed sub-
              path of $C$ from that breakpoint to itself.
            • If $C$ has two or more breakpoints, then a *segment* of $C$ is the directed
              subpath of $C$ between two consecutive breakpoints.
      3-3. Let $\Pi$ be the set of segments computed above.
    4. **return** $\Pi$.
**end**.
```

Based on Theorem 6.1, this procedure and CutSegment together can compute a CD-pair of spanning trees for Φ.

THEOREM 6.2. *Let n be the size of Φ. Then the procedure in Algorithm* 3 *correctly computes an output as specified in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The correctness of the procedure is shown as follows. Because the cycles in Γ are vertex-simple, the segments are internally vertex-simple. By the construction of Φ' and by the preorder of T, every cycle in Γ except the smallest-numbered one shares a vertex with a smaller-numbered cycle. Thus every cycle in Γ has at least one breakpoint and is partioned into segments, i.e., Π is a decomposition of Φ.

Π is an ordered decomposition with respect to the following partial order $\prec$: for $L_1, L_2 \in \Pi$ that are, respectively, segments of $C_1, C_2 \in \Gamma$, $L_1 \prec L_2$ if and only if the preorder number of C_1 is smaller than that of C_2.

The segment of the smallest-numbered cycle in Γ is the only head. Thus the first property for a multihead ear decomposition is satisfied. Because the preorder on Γ is a total order, by the choice of the breakpoints the second property for a multihead ear decomposition is also satisfied.

As for the complexity of the procedure, it suffices to show that Steps 1-1 through 3-3 each take $O(\log n)$ time with $n/\log n$ processors. By Lemma 2.1, Step 1-1 runs within the desired complexity. Step 1-2 is obvious. Step 1-3 is also straightforward; it need not compute a combinatorial embedding of Φ'. By the planarity of Φ', Step 2-1 uses an optimal algorithm for planar connectivity [15]. Because the size of Φ' is $O(n)$, Step 2-1 runs within the desired complexity. Step 2-2 uses the Euler tour [4]. Steps 3-1 through 3-3 are straightforward. $\square$

THEOREM 6.3. *For an embedded normal graph of size n, a CD-pair of spanning trees for each positive cluster can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The procedure in Algorithm 4 computes a CD-pair of spanning trees for each positive cluster within the desired complexity.

ALGORITHM 4. *Procedure for computing a* CD-*pair of spanning trees for each positive cluster.*

```
Procedure ComputeClusterCDTrees
Input: an embedded normal graph $G$.
Output: a CD-pair of spanning trees for each positive cluster of $G$.
begin
    1. Compute the positive clusters of $G$.
    2. for each positive cluster $\Phi$ do
          begin
             2-1. Let $\Pi$ = ComputeClusterSegments($\Phi$).
             2-2. Let $\Phi_c$ = CutSegment($\Phi$, first, $\Pi$).
             2-3. Let $\Phi_d$ = CutSegment($\Phi$, last, $\Pi$).
          end.
    3. return $\Phi_c$ and $\Phi_d$ for each $\Phi$.
end.
```

The correctness and the complexity of the procedure follow from Lemma 2.1 and Theorems 6.1 and 6.2. □

6.3. Computing spanning forests for embedded normal graphs. The procedure ComputePositiveSegments in Algorithm 5 computes a useful set of ears for an embedded normal graph without large degree vertices.

ALGORITHM 5. *Procedure for computing the ears in an ear decomposition for an embedded normal graph.*

```
Procedure ComputePositiveSegments
Input: an embedded normal graph $G$ with no large-degree vertices.
Output: the set $\Pi$ of ears in a multihead ear decomposition of $G$. Remark: The partial
order of the decomposition is not part of the output.
begin
    1. Compute the boundary cycles of the faces of $G$ and those of the positive faces.
       Remark: Because $G$ has no large-degree vertices, these cycles are vertex-simple.
    2. Compute the *breakpoints* of each face of $G$ defined as follows:
          • For a positive face the *breakpoint* of that face is an arbitrary boundary vertex
            of that face. A negative face has no breakpoint.
          • For a noncycle face the *breakpoints* of that face are the local sources and
            sinks on its boundary cycle.
       Remark: A breakpoint is defined with respect to a face. A vertex may be a break-
       point for a face but not for another.
    3. Compute the *positive segments* of each face $f$ of $G$ defined as follows:
          • If $f$ has only one breakpoint, then the *segment* of $f$ is the directed boundary
            path of $f$ from that breakpoint to itself.
          • If $f$ has two or more breakpoints, then a *segment* is the directed boundary
            path of $f$ between two consecutive breakpoints.
          • A *positive* segment of $f$ is one whose edges are all positive with respect to $f$.
    4. Let $\Pi$ be the set of positive segments computed above.
    5. return $\Pi$.
end.
```

Based on Theorem 6.1, this procedure and CutSegment together compute a useful CD-pair of spanning forests.

THEOREM 6.4. *The following statements are true:*

1. *For an input graph of size n the procedure in Algorithm 5 runs in $O(\log n)$ time with $n/\log n$ processors.*

2. *The output of the procedure consists of the ears in a multihead ear decomposition whose heads are exactly the segments of the positive faces of G.*

Proof. The two statements are proved as follows.

Statement 1. By Lemma 2.1, Step 1 takes $O(\log n)$ time with $n/\log n$ processors. Then Steps 2–4 can be done in a straightforward manner in $O(\log n)$ time on $n/\log n$ processors. Thus the total complexity is as stated.

Statement 2. Because the boundary cycles of the faces of G are vertex-simple, the positive segments are internally vertex-simple. Because an edge is positive on exactly one face, the positive segments partition the edges of G. Thus Π is a decomposition of G.

As for a suitable partial order on Π, by Theorem 3.1 $\tilde{G}$ is acyclic and can be regarded as a partial order $\prec_f$ on the faces of G with the positive faces being the minimal elements. The order $\prec_f$ induces the following partial order $\prec_s$ on Π: let f_1 and f_2 be two faces. Let L_1 and L_2 be two positive segments of f_1 and f_2, respectively. Then $L_1 \prec_s L_2$ if and only if $f_1 \prec_f f_2$.

Because the positive faces are the minima in $\prec_f$, their segments are the heads. Because a positive face has only one breakpoint, the two endpoints of a head are the same. Because G has no large-degree vertex, no two positive faces share a boundary vertex. Thus, the heads are disjoint, satisfying the first property for a multihead ear decomposition.

As for the second property of a multihead ear decomposition, let L_1 be a nonhead ear. Let L_2 be a distinct segment intersecting L_1 at a vertex v. There are two cases: (1) v is an endpoint of L_1 or (2) v is an internal vertex of L_1. These two cases are symmetric. Case (1) leads to the first half of the second property. Case (2) leads to the second half of that property. Only Case (1) is discussed below.

Let f_1 be the face of which L_1 is a positive segment. By definition v is a source or a sink of $\mathcal{B}(f_1)$; by symmetry assume that v is a source. By the strong connectivity and the degree condition of G the degree of v is three. Let w_1, w_2, w_3 be the vertices incident with v in the clockwise order. Assume that $w_1, w_2 \in \mathcal{B}(f_1)$ but $w_3 \notin \mathcal{B}(f_1)$. Then the edges incident with v are $d_1 = v \rightarrow w_1$, $d_2 = v \rightarrow w_2$, $d_3 = w_3 \leftarrow v$. Because $d_1, d_2 \in \mathcal{B}(f_1)$, L_1 is the positive segment of f_1 that contains d_1. Let f_2 be the face whose boundary contains d_2 and d_3. Then L_2 is the positive segment of f_2 that contains d_2 and d_3. Notice that $f_2 \prec_f f_1$. Thus $L_2 \prec_s L_1$. □

THEOREM 6.5. *The following statements are true:*

1. *For an input graph of size n the procedure in Algorithm 6 runs in $O(\log n)$ time with $n/\log n$ processors.*

2. *The output G_c and G_d of the procedure form a* CD*-pair of spanning forests for G with the following properties:*

(a) *For each positive face the boundary vertices of that face are contained in exactly one tree in each forest.*

(b) *In each forest each tree contains the boundary vertices of exactly one positive face.*

(c) *Each tree is rooted at the breakpoint of its corresponding positive face.*

ALGORITHM 6. *Procedure for computing a* CD*-pair of spanning forests for an embedded normal graph.*

```
Procedure ComputeCDForests
Input: an embedded normal graph $G$ with no large-degree vertices.
Output: a CD-pair of spanning forests $G_c$ and $G_d$ for $G$.
begin
    1. Let $\Pi$ = ComputePositiveSegments($G$).
    2. Let $G_c$ = CutSegment($G$, first, $\Pi$).
    3. Let $G_d$ = CutSegment($G$, last, $\Pi$).
    4. return $G_c$ and $G_d$.
end.
```

Proof. The theorem follows from Theorems 6.1 and 6.4. Note that the partial orders $\prec_f$ and $\prec_s$ are implicit and require no computation at all. □

THEOREM 6.6. *Given an embedded normal graph of size n with exactly one positive face and no large-degree vertices, a CD-pair of spanning trees can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. This a corollary of Theorem 6.5. Because the given graph has exactly one positive face, there is only one tree in each of G_c and G_d. □

6.4. Biconnected territories. Let G be an embedded normal graph with no large-degree vertices. A *biconnected territory* of a positive face of G is a vertex subset as defined by the procedure in Algorithm 7.

ALGORITHM 7. *Procedure for computing the biconnected territories.*

```
Procedure ComputeBiconnectedTerritories
Input: an embedded normal graph $G$ with no large-degree vertices.
Output: the biconnected territories of $G$.
begin
    1. Let $\Pi$ = ComputePositiveSegments($G$).
    2. Let $G_c$ = CutSegment($G$, first, $\Pi$).
    3. Let $G_d$ = CutSegment($G$, last, $\Pi$).
    4. for each positive face $f$ of $G$ do
        begin
            4-1. Let $\mathcal{C}_c(f)$ be the connected component in $G_c$ that contains the boundary vertices of $f$.
            4-2. Let $\mathcal{C}_d(f)$ be the connected component in $G_d$ that contains the boundary vertices of $f$.
            4-3. Let $\mathcal{S}(f)$ be the subgraph of $G$ induced from the vertices shared by $\mathcal{C}_c(f)$ and $\mathcal{C}_d(f)$.
            4-4. Let $\mathcal{BT}(f)$, the biconnected territory of $f$, be the biconnected component in $\mathcal{S}(f)$ that contains $\mathcal{B}(f)$.
        end.
    5. return all $\mathcal{BT}(f)$ computed above.
end.
```

6.4.1. Computing biconnected territories. The next theorem analyzes the procedure in Algorithm 7.

THEOREM 6.7. *The following statements are true:*

1. *$\mathcal{C}_c(f)$, $\mathcal{C}_d(f)$, and $\mathcal{BT}(f)$ exist.*

2. *The subgraphs $\mathcal{C}_c(f)$, $\mathcal{C}_d(f)$, and $\mathcal{BT}(f)$ for distinct positive faces are respectively disjoint.*

3. *Let n be the size of G. The biconnected territories of G can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The statements are proved as follows.

Statement 1. The existence of $\mathcal{C}_c(f)$ and $\mathcal{C}_d(f)$ follows from Theorem 6.5(2a). The existence of $\mathcal{BT}(f)$ derives from that of $\mathcal{C}_c(f)$ and $\mathcal{C}_d(f)$ and the fact that $\mathcal{B}(f)$ is biconnected because G has no large-degree vertices.

Statement 2. The disjointness of $\mathcal{C}_c(f)$'s and the $\mathcal{C}_d(f)$'s follows from Theorem 6.5(2b). The disjointness of $\mathcal{BT}(f)$'s follows from that of $\mathcal{C}_c(f)$'s and $\mathcal{C}_d(f)$'s.

Statement 3. The complexity of the procedure in Algorithm 7 is analyzed as follows. Steps 1 through 3 are performed by means of Theorem 6.5(1). Step 4 is started by computing the positive faces of G via Lemma 2.1. Steps 4-1 through 4-4 can be computed in $O(\log n)$ time with $n/\log n$ processors by using the second statement of this theorem and optimal parallel algorithms for planar connectivity and biconnectivity [15]. Thus the biconnected territories can be computed with the stated complexity. □

6.4.2. Contracting biconnected territories. Contracting biconnected territories is a key step in reducing the number of positive faces.

Let G be an embedded normal graph with no large-degree vertices. Let G' be the graph constructed from G by contracting each biconnected territory into a vertex.

Because G has no large-degree vertices, its positive faces have disjoint boundaries. The contraction of the biconnected territories may create large-degree vertices in G'. Thus positive faces in G' may cluster together. The next two propositions relate the number of positive clusters in G' to that of positive faces in G.

LEMMA 6.8. *Let f be a noncycle face in G. Let P be a directed boundary path of f that has at least two edges and consists of positive edges of f. If the endpoints of P are in biconnected territories of G but its internal vertices are not, then the endpoints of P are in different biconnected territories.*

Proof. Let $P = p_1, \ldots, p_s$. To prove by contradiction, assume that p_1 and p_s are in the same biconnected territory $\mathcal{BT}(f)$. It suffices to show that $P \subseteq \mathcal{BT}(f)$, contradicting the condition that the internal vertices of P are not in any biconnected territory.

Note that P is a subpath of a positive segment Q of f. Let $Q = q_1, \ldots, q_t$. After the first edge of Q is cut, the subpath $Q' = q_2, \ldots, q_t$ remains in G_c. Because the path $P_c = p_2, \ldots, p_s$ is a subpath of Q', the path P_c is connected to p_s in G_c. Because $p_s \in \mathcal{BT}(f) \subseteq \mathcal{C}_c(f)$, the path P_c is in $\mathcal{C}_c(f)$. On the other hand, because $p_1 \in \mathcal{BT}(f) \subseteq \mathcal{C}_c(f)$, p_1 is also in $\mathcal{C}_c(f)$. Thus P is in $\mathcal{C}_c(f)$. By symmetry P is also in $\mathcal{C}_d(f)$. Thus P is in $\mathcal{S}(f)$. By the proof assumption P is connected to $\mathcal{BT}(f)$ at both p_1 and p_s. Because G has no large-degree vertex, the union of P and $\mathcal{BT}(f)$ is biconnected in $\mathcal{S}(f)$. Consequently, P lies in $\mathcal{BT}(f)$. □

THEOREM 6.9. *The number of positive clusters in G' is at most half the number of positive faces in G.*

Proof. It suffices to show that the boundary of each positive face in G' contains at least two contracted biconnected territories of G. To prove by contradiction, assume that there is a positive face f' in G' whose boundary contains at most one contracted biconnected territory of G. If $\mathcal{B}(f')$ contains no contracted biconnected territory of G, then f' is a positive face in G, contradicting the fact that every positive face of G is contracted in G'. Thus, without loss of generality, assume that $\mathcal{B}(f')$ contains exactly one vertex v' that is a contracted biconnected territory of G. Let P' be a vertex-simple directed boundary path of f' from v' to v' with at least one edge. By Lemma 5.2 P' is originally a positive boundary path P of a noncycle face in G such that the two endpoints of P are in biconnected territories of G but its internal vertices are not. Because G' has no loop edges, P has at least two edges. By Lemma 6.8 the endpoints of P are in different biconnected territories of G, contradicting the proof assumption that $\mathcal{B}(f')$ contains only one contracted biconnected territory of G. □

7. Tree rerooting and duplicate removal.

7. Tree rerooting and duplicate removal. Tree rerooting and duplicate removal are useful techniques for computing directed spanning trees. When they are used together, duplicate removal provides crucial flexibility to tree rerooting so that forests can be merged efficiently. These two techniques are detailed in §§7.1 and 7.2. Their combination and applications are described in §§7.3 and 7.4.

7.1. Tree rerooting. The procedure in Algorithm 8 describes the technique of changing the roots of trees and the next theorem analyzes the procedure.

ALGORITHM 8. *Procedure for changing the root of a* CD*-pair of trees.*

```
Procedure RerootCDTrees
Input:
    1. a CD-pair of trees $T_c$ and $T_d$ of a directed graph $G$ (remark: $G$ is not part of the
       input),
    2. the common root $r$ of the two trees,
    3. a vertex $r'$ shared by the two trees.
Output: a CD-pair of trees $T'_c$ and $T'_d$ of $G$ with the following properties:
    1. $T'_c$ and $T'_d$ are both rooted at $r'$.
    2. $T'_c$ and $T'_d$ share at least the vertices originally shared by $T_c$ and $T_d$.
    3. $T'_c$ and $T'_d$ together contain at most the vertices originally contained in $T_c$ and $T_d$
       together.
begin
    1. Compute $T'_c$ as follows:
        1-1. Let $P_d$ be the tree path in $T_d$ from $r$ to $r'$.
        1-2. Let $T'_c$ be $T_c$ with $P_d$ added.
        1-3. For every vertex $v \in T'_c$, if $v$ has two outgoing edges, delete the one not in
             $P_d$. Also, delete from $T'_c$ the outgoing edge of $r'$.
    2. Compute $T'_d$ symmetrically.
    3. return $T'_c$ and $T'_d$.
end.
```

THEOREM 7.1. *Let n_c and n_d be the numbers of vertices in T_c and T_d. The procedure in Algorithm* 8 *correctly computes an output as specified in $O(\log(n_c+n_d))$ time with $(n_c+n_d)/\log(n_c+n_d)$ processors.*

Proof. The correctness of the procedure is straightforward. As for the complexity, the key fact is that P_d can be computed in the desired complexity by using tree contraction [1], [10], [13], [18], [21]. □

7.2. Duplicate removal. The procedure in Algorithm 9 describes the technique for removing duplicates from a tree and the next theorem analyzes the procedure.

ALGORITHM 9. *Procedure for removing the duplicates from a tree.*

```
Procedure RemoveDuplicates
Input:
    1. a directed graph $G$,
    2. a directed spanning tree $T$ of $G$,
    3. disjoint vertex subsets $U_1, \dots, U_s$ of $G$,
    4. the directed graph $G'$ constructed from $G$ by contracting each $U_i$ into a vertex.
```

```
Output: a directed spanning tree $T'$ of $G'$ with the same orientation and root as those of $T$.
begin
  1. For all $i$ let $u_i$ be the highest vertex of $U_i$ in $T$; if there are two vertices or more
     at the same smallest depth, choose one of them arbitrarily.
  2. For all $i$ delete from $T$ all edges between the vertices in $U_i - \{u_i\}$ and their parents.
  3. Construct $T'$ from the above $T$ by contracting $U_i$ into $u_i$.
  4. return $T'$.
end.
```

THEOREM 7.2. *The procedure in Algorithm 9 correctly computes an output as specified in $O(\log n)$ time with $n/\log n$ processors for an input graph of size n.*

Proof. The correctness of the procedure is straightforward. As for the complexity, the key fact is that the representatives u_i can be found in $O(\log n)$ time with $n/\log n$ processors by using tree contraction [1], [10], [13], [18], [21]. □

7.3. Combining tree rerooting and duplicate removal. The procedure in Algorithm 10 combines the techniques of tree rerooting and duplicate removal and the next theorem analyzes the procedure.

ALGORITHM 10. *Procedure for combining the techniques of tree rerooting and duplicate removal.*

```
Procedure RerootAndRemoveDuplicates
Input:
  1. a strongly connected directed graph $G$,
  2. disjoint vertex subsets $U_1, \dots, U_s$ of $G$,
  3. a CD-pair of forests for $G$ that consists of a CD-pair of covering trees $S_{i,c}$ and
     $S_{i,d}$ for each $U_i$,
  4. the directed graph $G'$ constructed from $G$ by contracting each $U_i$ into a supervertex $u'_i$,
  5. a CD-pair of spanning trees $T'_c$ and $T'_d$ for $G'$.
Output: a CD-pair of spanning trees $T_c$ and $T_d$ for $G$.
begin
  1. Compute $T_c$ as follows:
    1-1. Find a local root $r_i$ for each $U_i$:
           • For each supervertex $u'_i$, if $u'_i$ is the root of $T'_c$, then let $r_i$ be an arbitrary
             vertex in $U_i$.
           • Otherwise, let $d'$ be the outgoing edge of $u'_i$ in $T'_c$. Let $d$ be the original
             of $d'$ in $G$. Let $r_i \in U_i$ be the start vertex of $d$.
    1-2. For each $u'_i$ apply the procedure in Algorithm 8 to $S_{i,c}$ and $S_{i,d}$ to find a
         convergent tree $R_i$ of $G$ covering $U_i$ and rooted at $r_i$.
    1-3. Create duplicates to prepare for expanding $u'_i$ in $T'_c$ into $R_i$:
         For each vertex $v$ in $G$ replace each occurrence of $v$ in the trees $R_i$ with a
         distinct duplicate of $v$. Also collect the duplicates for $v$.
    1-4. Undo the contractions in $T'_c$ by expanding each $u'_i$ into $R_i$:
           • For each edge $d' \in T'_c$ let $d = v \to w$ be its original in $G$. If $v$ is in some
             $U_i$, then replace the $v$-end of $d$ by the duplicate of $v$ for $R_i$. Perform the
             same replacement to the $w$-end. Let $T''_c$ be the resulting $T'_c$.
           • Let $T_c$ be the tree formed by $T''_c$ and all trees $R_i$. Remark: The dupli-
             cates prevent cycles from being created in $T_c$; in fact, $T_c$ is a convergent
             spanning tree of $G$ with duplicates.
    1-5. Use the procedure in Algorithm 9 to remove duplicates from $T_c$.
  2. Compute $T_d$ symmetrically. To maintain the same root for $T_c$ and $T_d$, if the root
     of $T'_c$ and $T'_d$ is a supervertex $u'_i$, then choose the same $r_i$ for $U_i$ at Step 1-1 for
     both $T_c$ and $T_d$.
  3. return $T_c$ and $T_d$.
end.
```

THEOREM 7.3. *Let n be the size of G. Then the procedure in Algorithm* 10 *correctly computes an output as specified in $O(\log n)$ time with $n/\log n$ processors.*

Proof. The correctness of the procedure is analyzed as follows. First, the procedure in Algorithm 8 can be used at Step 1-2 because $r_i \in U_i \subseteq S_{i,c} \cap S_{i,d}$. Next, at Step 1-4, T_c is a convergent spanning tree of G with duplicates. This tree property follows from the creation of duplicates, the choice of r_i, and the fact that R_i covers U_i and is rooted at r_i. After the duplicates are removed at Step 1-5, the resulting T_c is a convergent spanning tree of G. By symmetry T_d found by Step 2 is a divergent spanning tree of G. If the root of T'_c and T'_d is a supervertex u'_i, then by the choice of r_i at Step 2, T_c and T_d have the same root. If the root of T'_c and T'_d is not a supervertex u'_i, then T_c, T_d, T'_c, and T'_d all have the same root.

As for the complexity, by symmetry it suffices to show that Steps 1-1–1-5 each take $O(\log n)$ time with $n/\log n$ processors. Step 1-1 is straightforward. By Theorem 7.1, Step 1-2 takes $O(\log n)$ time with $n/\log n$ processors. Because the trees $S_{i,c}$'s and the trees $S_{i,d}$'s are respectively disjoint, by the third output property in Algorithm 8 each vertex of G has at most three copies in T'_c and the trees R_i. Thus Steps 1-3 and 1-4 each take $O(\log n)$ time with $n/\log n$ processors. Then by Theorem 7.2, Step 1-5 takes $O(\log n)$ time with $n/\log n$ processors. □

7.4. Applications of tree rerooting and duplicate removal. Four applications are given below.

7.4.1. Changing the root of a CD-pair of spanning trees. Let G be a strongly connected directed graph of size n.

COROLLARY 7.4. *Given a* CD*-pair of spanning trees for G, a* CD*-pair of spanning trees for G rooted at a specified vertex can be computed in $O(\log n)$ time with $n/\log n$ processors.*

Proof. This is a straightforward corollary of Theorem 7.1. □

7.4.2. Undoing vertex expansion. Let G be an embedded normal graph of size n. Let G' be constructed from G by replacing some vertices each with a positive face or a negative face.

COROLLARY 7.5. *A* CD*-pair of spanning trees for G can be computed in $O(\log n)$ time with $n/\log n$ processors given G, G', and a* CD*-pair of spanning trees for G'.*

Proof. This is a corollary of Theorem 7.2 with G and G' in reversed roles. The key observation is that for each expanded vertex of G there is a U_i consisting of the vertices of the corresponding cycle face in G'. As for the complexity, note that the size of G' is at most $5n$. □

7.4.3. Undoing the contraction of biconnected territories. Let G be an embedded normal graph without large-degree vertices. Let n be the size of G. Let G' be the graph constructed by contracting each biconnected territory of G into a vertex. Let G_c and G_d be the graphs constructed by removing, respectively, the first edge and the last edge of each positive segment of G.

COROLLARY 7.6. *A* CD*-pair of spanning trees for G can be computed in $O(\log n)$ time with $n/\log n$ processors given G, its biconnected territories, G_c, G_d, G', and a* CD*-pair of spanning trees for G'.*

Proof. This is a corollary of Theorem 7.3 with each U_i being a biconnected territory. By the definition of a biconnected territory and by Theorem 6.5, G_c and G_c consist of a CD-pair of covering trees for each U_i. □

7.4.4. Undoing the contraction of positive clusters. Let G be an embedded normal graph of size n. Let G' be the graph constructed by contracting each positive cluster of G into a vertex.

COROLLARY 7.7. *A* CD*-pair of spanning trees for G can be computed in $O(\log n)$ time with $n/\log n$ processors given G, its positive clusters, a* CD*-pair of spanning trees for each positive cluster of G, the graph G', and a* CD*-pair of spanning trees for G'.*

Proof. This is a corollary of Theorem 7.3 with each U_i consisting of the boundary vertices of a positive cluster. □

8. Computing directed spanning trees. Section 8.1 gives a directed spanning tree algorithm for normal graphs without large-degree vertices, and §8.2 generalizes this algorithm for normal graphs that may have large-degree vertices.

8.1. Spanning trees for graphs without large-degree vertices. Algorithm 11 describes a directed spanning-tree algorithm for embedded normal graphs without large-degree vertices, and the algorithm is analyzed below.

ALGORITHM 11. *Procedure for computing* CD*-pairs of spanning trees for graphs with no large-degree vertex.*

```
Procedure ComputeCDT
Input: an embedded normal graph $G$ without large-degree vertices.
Output: a CD-pair of spanning trees for $G$.
begin
  if $G$ has exactly one positive face
    then $\Upsilon \leftarrow$ CutSegment($G$)
    else
      begin
        Stage 1:
          1-1. Let $G_c$ be the graph constructed by deleting the first edge of each
               positive segment of $G$.
          1-2. Let $G_d$ be the graph constructed by deleting the last edge of each
               positive segment of $G$.
          1-3. Find the biconnected territories of $G$ from $G_c$ and $G_d$.
          1-4. Construct $G_1$ from $G$ by contracting each biconnected territory
               into a single vertex.
        Stage 2:
          2-1. Compute the positive clusters of $G_1$.
          2-2. Compute a CD-pair of spanning trees for each positive cluster of
               $G_1$.
          2-3. Construct $G_2$ from $G_1$ by contracting each positive cluster into a
               single vertex.
        Stage 3:
          3-1. Construct $G_3$ from $G_2$ by expanding into a positive face each con-
               tracted positive cluster of $G_1$. Remark: As stated in §4, for techni-
               cal uniformity, if $G_2$ is a single vertex, it is expanded into a positive
```

```
            face with two boundary edges. Also, if a contracted positive clus-
            ter of $G_1$ is of degree two in $G_2$, it is also expanded into a positive
            face with two boundary edges.
        3-2. Construct $G_4$ from $G_3$ by expanding each large-degree vertex into
            a negative face.
      Stage 4:
        4-1. $\Upsilon_4 \leftarrow$ ComputeCDT($G_4$).
        4-2. Compute from $\Upsilon_4$ a CD-pair $\Upsilon_3$ of spanning trees for $G_3$ via Corol-
            lary 7.5.
        4-3. Compute from $\Upsilon_3$ a CD-pair $\Upsilon_2$ of spanning trees for $G_2$ via Corol-
            lary 7.5.
        4-4. Compute a CD-pair $\Upsilon_1$ of spanning trees for $G_1$ via Corollary 7.7
            from $\Upsilon_2$, the positive clusters of $G_1$, and the CD-pairs of spanning
            trees computed at Step 2-2.
        4-5. Compute a CD-pair $\Upsilon$ of spanning trees for $G$ via Corollary 7.6
            from $\Upsilon_1$, $G_c$, $G_d$, and the biconnected territories of $G$.
    end.
  return $\Upsilon$.
end.
```

LEMMA 8.1. *Let n be the size of G. Let n_i be the size of G_i for $i \in \{1,2,3,4\}$. Let β and β_4 be the numbers of positive faces in G and G_4. Then the following statements are true:*

1. *G_4 is an embedded normal graph without large-degree vertices.*
2. *$\beta_4 \leq \beta/2$.*
3. *$n_i \leq n$ for $i \in \{1,2,3,4\}$.*

Proof. The statements are shown below.

Statement 1. Step 3-1 ensures that G_4 contains at least two vertices. Next, because G_4 is obtained from G by vertex expansion and vertex contraction, G_4 remains an embedded normal graph. Step 3-2 ensures that G_4 has no large-degree vertices.

Statement 2. Let β_1 (respectively, β_3) be the number of positive clusters (respectively, faces) in G_1 (respectively, G_3). By Theorem 6.9, $\beta_1 \leq \beta/2$. Step 2-3 destroys the positive faces of G_1 but may create new positive faces in G_2. Next, because the boundary of each positive face in G_2 must contain at least one contracted positive cluster of G, Step 3-1 destroys all positive faces of G_2. Thus $\beta_3 = \beta_1$. Step 3-2 cannot destroy or create positive faces, and so $\beta_4 = \beta_3$. In sum, the statement is true.

Statement 3. Because vertex contraction does not increase the size of a graph, $n_2 \leq n_1 \leq n$. Also, $n_2 \leq n_3 \leq n_4$ because vertex expansion does not decrease the size of a graph. It remains to show $n_4 \leq n$.

Note that G_4 is in effect expanded from G_2. Also, G_2 is in effect contracted from G. Let u be a vertex of G_2 that is expanded in G_4. Let U be the connected vertex subset of G that is contracted into u.

U is shown to be biconnected in G as follows. A vertex in G_2 can be expanded at most once either at Step 3-1 or Step 3-2. Thus there are two cases based on which step expands u.

- Case (1): u is expanded at Step 3-1. Then u is contracted from a positive cluster of G_1 that contains contracted biconnected territories of G. Thus by the small degree of G the set U is biconnected in G.
- Case (2): u is expanded at Step 3-2. Then u is a contracted biconnected territory of G. Thus U is biconnected.

The biconnectivity of U is used to estimate the change from n to n_4 as follows. For an edge (respectively, vertex) set X let $|X|$ be the number of edges (respectively, vertices) in X. Let D_b (respectively, D_o) be the set of edges in G with both endpoints (respectively, only one endpoint) in U. By the small degree of G and the biconnectivity of U, $|D_o| \leq |D_b|$ and $|D_o| \leq |U|$. The contraction of U loses U and D_b, gains u, and keeps D_o. The expansion of u keeps D_o, loses u, and gains $|D_o|$ new edges and $|D_o|$ new vertices. Thus in the conversion from G to G_4 the net gain concerning U is $2|D_o| - |U| - |D_b|$, which is at most zero by the above two inequalities. Therefore, $n_4 \leq n$. □

THEOREM 8.2. *Let G be an embedded normal graph without large-degree vertices. Let n be the size of G. Let β be the number of positive faces in G. Then Algorithm* 11 *computes a* CD*-pair of spanning trees for G in $O(\lceil\log(\beta+1)\rceil \log n)$ time using $n/\log n$ processors on a deterministic arbitrary*-CRCW PRAM.

Proof. By Lemma 8.1(1), Step 4-1 can recurse on G_4. Then the correctness of the algorithm follows from Theorem 6.6 and Corollaries 7.5, 7.6, and 7.7.

As for the complexity, by Lemma 8.1(2) the recursion depth is $\lceil\log(\beta+1)\rceil$. Thus it suffices to show that each stage runs in $O(\log n)$ time with $n/\log n$ processors. By Lemma 8.1(3) the size of the graph processed by each step at each level of recursion is at most n. Based on this bound, the desired complexity for the stages follows from the implementation below.

The if-condition is tested by means of Lemma 2.1(1). The then-statement is performed by means of Theorem 6.1(1). Stage 1 is done by means of Theorems 6.1(1), 6.7(3), and 5.1. Stage 2 is done by means of Lemma 2.1(3) and Theorems 6.3 and 5.1. Stage 3 is performed by means of Theorem 4.1(2). Stage 4 is performed by means of Corollaries 7.5, 7.6, and 7.7. □

8.2. Spanning trees for graphs with large-degree vertices. Algorithm 12 describes a directed spanning-tree algorithm for embedded normal graphs that may have large-degree vertices, and the algorithm is analyzed below.

ALGORITHM 12. *Procedure for computing a* CD*-pair of spanning trees.*

```
Procedure ComputeCDTrees
Input: an embedded normal graph $G$.
Output: a CD-pair of spanning trees for $G$.
begin
    Stage 1:
      1-1. Compute the positive clusters of $G$.
      1-2. Compute a CD-pair of spanning trees for each positive cluster of $G$.
      1-3. Construct $G_1$ from $G$ by contracting each positive cluster into a single
           vertex.
      1-4. Construct $G_2$ from $G_1$ by expanding into a positive face each contracted
           positive cluster of $G$. Remark: As stated in §4, for technical uniformity, if
           $G_1$ is a single vertex, it is expanded into a positive face with two boundary
           edges. Also, if a contracted positive cluster of $G$ is of degree two in $G_1$,
           it is also expanded into a positive face with two boundary edges.
      1-5. Construct $G'$ from $G_2$ by expanding each large-degree vertex into a neg-
           ative face.
    Stage 2:
      2-1. $\Upsilon' \leftarrow$ ComputeCDT($G'$).
      2-2. Compute a CD-pair $\Upsilon_2$ of spanning trees for $G_2$ via Corollary 7.5 from
           $\Upsilon'$.
```

```
        2-3. Compute a CD-pair $\Upsilon_1$ of spanning trees for $G_1$ via Corollary 7.5 from
             $\Upsilon_2$.
        2-4. Compute a CD-pair $\Upsilon$ of spanning trees for $G$ via Corollary 7.7 from $\Upsilon_1$,
             the positive clusters of $G$, and the CD-pairs of spanning trees computed
             at Step 1-2.
    return $\Upsilon$.
end.
```

LEMMA 8.3. *Let n, n_1, n_2, and n' be the sizes of G, G_1, G_2, and G'. The following statements are true:*

1. *G' is an embedded normal graph without large-degree vertices.*
2. *The number of positive faces in G' equals that of positive clusters in G.*
3. $n'/5 \le n_2 \le n_1 \le n$.

Proof. The third statement is straightforward. As for the other two statements, note that Stage 1 of Algorithm 12 combines Stages 2 and 3 of Algorithm 11. Thus the proof for the first statement is similar to that for Lemma 8.1(1). The proof for the second statement is similar to that for Lemma 8.1(2). □

THEOREM 8.4. *Let G be an embedded normal graph. Let n be the size of G. Let β be the number of positive clusters in G. Then Algorithm 12 computes a* CD*-pair of spanning trees for G in $O(\lceil \log(\beta+1) \rceil \log n)$ time with $n/\log n$ processors on a deterministic arbitrary-*CRCW PRAM. *Consequently, by Corollary* 7.4 *a* CD*-pair of spanning trees rooted a specified vertex can be computed with the same complexity.*

Proof. The proof is similar to that of Theorem 8.2. The key points are as follows. By Lemma 8.3(1), Step 2-1 may apply ComputeCDT to G'. The cost of this step dominates the complexity of the algorithm. By Lemmas 8.3(2) and 8.3(3) this step takes $O(\lceil \log(\beta+1) \rceil \log n)$ time with $n/\log n$ processors. □

THEOREM 8.5. *For a normal graph of size n without a given embedding, a* CD*-pair of spanning trees rooted at a specified vertex can be computed in $O(\log^2 n)$ time with $n/\log n$ processors on a deterministic arbitrary-*CRCW PRAM.

Proof. To use Theorem 8.2, a spherical embedding for the input graph is first computed in $O(\log n \log\log n)$ time with $n/\log n$ processors [26]. The complexity follows from the fact that there are at most n positive clusters in the input graph with any embedding. □

9. Open problems. This paper has shown that for a strongly connected planar directed graph of size n, a directed spanning tree rooted at a specified vertex can be computed in $O(\log^2 n)$ time with $n/\log n$ processors. This result complements the linear-processor NC algorithm by Kao for computing the strongly connected components of a planar directed graph [18]. There are several fundamental problems left open in this paper. Perhaps the most important one is to compute planar breadth-first search trees efficiently. Currently, the best algorithm for this problem is by Pan and Reif [25]. Their algorithm computes the shortest paths in undirected planar graphs in $O(\log^2 n)$ time with $n^{1.5}/\log n$ processors. The algorithm is based on matrix operations and uses an almost optimal randomized algorithm for planar separators of Gazit and Miller [11]. Reducing to linear the processor complexity of planar breadth-first search would be of significant impact.

Acknowledgments. The authors thank Fang Wan for helping to prepare some of the figures. M.-Y. Kao gratefully acknowledges Philip Klein, Gary Miller, and Vijaya Ramachandran for many helpful discussions and suggestions.

REFERENCES

[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, *A simple tree contraction algorithm*, J. Algorithms, 10 (1989), pp. 287–302.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[3] R. J. Anderson and G. L. Miller, *Deterministic parallel list ranking*, Algorithmica, 6 (1991), pp. 859–868.

[4] M. Atallah and U. Vishkin, *Finding Euler tours in parallel*, J. Comput. System Sci., 29 (1984), pp. 330–337.

[5] C. Berge, *Graphs*, 2nd revised ed., North-Holland, New York, 1985.

[6] B. Bollobás, *Graph Theory*, Springer-Verlag, Berlin, New York, 1979.

[7] R. Cole and U. Vishkin, *The accelerated centroid decomposition technique for optimal tree evaluation in logarithmic time*, Algorithmica, 3 (1988), pp. 329–346.

[8] ———, *Faster optimal prefix sums and list ranking*, Information and Computation, 81 (1989), pp. 334–352.

[9] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[10] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.

[11] H. Gazit and G. L. Miller, *A parallel algorithm for finding a separator in planar graphs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 238–248.

[12] ———, *An improved parallel algorithm that computes the* BFS *numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.

[13] H. Gazit, G. L. Miller, and S. H. Teng, *Optimal tree contraction in the* EREW *model*, in Concurrent Computations: Algorithms, Architecture, and Technology, S. T. Dickinson, B. W. Dickinson, and S. Schwartz, eds., Plenum Press, New York, 1988, pp. 139–156.

[14] A. M. Gibbons and W. Rytter, *An optimal parallel algorithm for dynamic expression evaluation and its applications*, in Proc. 6th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 241, Springer-Verlag, Berlin, New York, 1986, pp. 453–469.

[15] T. Hagerup, *Optimal parallel algorithms on planar graphs*, Inform. Comput., 84 (1990), pp. 71–96.

[16] Y. Han, *An optimal linked list prefix algorithm on a local memory computer*, IEEE Transactions on Computers, 40 (1991), pp. 1149–1153.

[17] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[18] M.-Y. Kao, *Linear-processor* NC *algorithms for planar directed graphs* I: *Strongly connected components*, SIAM J. Comput., 22 (1993), pp. 431–459.

[19] R. Karp and V. Ramachandran, *A survey of parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, New York, 1990, pp. 869–941.

[20] S. R. Kosaraju and A. L. Delcher, *Optimal parallel evaluation of tree-structured computations by raking*, in Proceedings of the 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures, J. H. Reif, ed., Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988, pp. 101–110.

[21] C. P. Kruskal, L. Rudolph, and M. Snir, *The power of parallel prefix*, IEEE Transactions on Computers, C-34 (1985), pp. 965–968.

[22] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980), pp. 831–838.

[23] L. Lovász, *Computing ears and branchings*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 464–467.

[24] G. L. Miller and J. H. Reif, *Parallel tree contractions and its applications*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 478–489.

[25] V. Pan and J. H. Reif, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.

[26] V. Ramachandran and J. H. Reif, *An optimal parallel algorithm for graph planarity*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 282–287.

[27] W. Tutte, *Graph Theory*, Vol. 21 of Encyclopedia of Mathematics and Its Applications, Addison-Wesley, Reading, MA, 1984.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 482–499, June 1993

BLOCKING SMALL CUTS IN A NETWORK, AND RELATED PROBLEMS*

DANIEL BIENSTOCK† AND NICOLE DIAZ†

Abstract. Let G be a graph with weights on the edges, S a subset of vertices, and k an integer. The problem of computing a minimum-weight subset of edges that meets all the cuts of cardinality $\leq k$ that separate pairs of vertices in S is considered. This problem is motivated by issues in network survivability. Assuming $|S| = 2$, it is shown that although this problem is NP-hard, it can be solved in linear time for each fixed value of k. Furthermore, if $|S| > 2$, the problem is NP-hard even for small values of k but can be solved in linear time for each fixed k and $|S|$.

Key words. graph algorithms, connectivity, complexity

AMS(MOS) subject classifications. 05, 68

Introduction. An increasingly important issue in communications networks design in the problem of building "survivability" into networks. Typically one wants to meet connectivity requirements between various pair of nodes. Such requirements are usually fairly small, as we are trying to guard against the unlikely (but not impossible) event of connection failures due to external factors. From a mathematical point of view, the problem can be stated as follows: Given a graph G, we want to add edges to G so that in the resulting graph, the edge connectivity between each pair i, j of vertices is at least a given number $r(i, j)$ (usually $r(i, j) \leq 3$, and possibly $r(i, j) = 0$). Moreover, each candidate edge has a certain cost associated with it, and we want to add to G a minimum-cost set of edges that is feasible in the above sense. This model has been considered by many authors, see [MMP], [BBM] for the cases $r(i, j) = 2$, and $r(i, j) = k$, respectively. Grötschel and Monma [GM] consider a cutting-plane algorithm for the case where $r(i, j)$ is of the form $\min\{r_i, r_j\}$. If in addition we have constant edge-costs, the problem is polynomially solvable (see [F]).

In order for the addition of an edge to be meaningful from a faulty-tolerance point of view, the geometrical layout of this edge must be physically diverse from the rest of the network. In some practical settings this requirement may make the addition of edges prohibitively expensive. On the other hand, the following may be possible: By paying a certain cost, we may be able to "reinforce" an existing edge, so as to make it immune to the type of damage under consideration. (See [C] for a different approach.)

In this new model (reinforcing existing edges instead of adding edges), the survivability problem can be stated as follows. We are given a graph G, a subset $S \subseteq V(G)$, and a weight function w on $E(G)$. We wish to find a minimum-weight subset of $E(G)$, whose contraction yields a graph with desired connectivity levels between vertices of S. Equivalently, we seek a minimum-weight subset $Z \subseteq E(G)$ such that $Z \cap C \neq \emptyset$ for every "small" cut C separating members of S. In the case where the $r(i, j)$ are equal to a constant k (for pairs i, j in S), we denote the problem by $P_w(S, G)$, and its value by $\tau_w(S, G)$. This paper begins the study of such problems, and we will primarily concentrate on the case $|S| = 2$. The case $|S| > 2$, which is substantially more intractable, will be considered in a future paper [BD]. However, the case $|S| = 2$ is already quite difficult and interesting from a theoretical point of view. Moreover, we expect that studying this special case will give insights useful for the general case.

*Received by the editors April 8, 1991; accepted for publication (in revised form) January 29, 1992. Partially supported by a National Science Foundation Presidential Young Investigator award, the Center for Telecommunications Research (Columbia), and a consulting agreement with Bellcore.

†Department of Industrial Engineering and Operations Research, Columbia University, New York, New York 10027.

Hence, given vertices s and t and a positive integer k, we want to block all s–t cuts of cardinality $\leq k$. We abbreviate, e.g., $P_w(\{s,t\},G)$ as $P_w(s,t,G)$. Not surprisingly, this class of problems is NP-hard (shown in §1) even when $w(e) = 1$ for all e. However, the proof produces values of k that grow with $|V(G)|$, and as stated above, we are mainly concerned in those cases where k is small. The main result of this paper, given in §2, is that for each fixed value of k there is a linear-time algorithm for the problem. The proportionality constant grows rapidly with k, of course, but for $k \leq 5$ the algorithm should be useful.

Sections 3 and 4 consider some related aspects of the problem. Viewed as an integer program, $P_w(s,t,G)$ is a set-covering problem. We expect that a cutting-plane algorithm will yield good computational results, especially when generalized for the case $|S| > 2$, and some initial computational experience suggests that this is the case [BD]. However, as shown in §3, there is a serious technical difficulty here in that solving the linear-programming relaxation of this set-covering problem turns out to be NP-hard. In §4 we return to the general case $|S| > 2$ and show that $P_w(S,G)$, is NP-hard already for $k = 8$ (although it is linear-time solvable for fixed k and fixed $|S|$).

The graphs we consider in this paper have no loops but may have parallel edges (all results are easily extended to simple graphs). If A is a set and f is a function from A to the reals, then for $X \subseteq A$, we write $f(X)$ or $\sum\{f(i) : i \in X\}$ to indicate the sum of values of f over X. We use the following notation, corresponding to a graph G. If $X, Y \subseteq V(G)$, then $\delta_G(X,Y)$ denotes the set of edges with one end in X the other in Y. We denote $\delta_G(X, V(G)\backslash X)$ by $\delta_G(X)$, and $d_G(X) = |\delta_G(X)|$ (if $X = \{u\}$ we write $\delta_G\{u\}$, etc.). For $X, Y \subseteq V(G)$, $\lambda_G(X,Y)$ denotes the maximum number of edge-disjoint paths from X to Y. A cut is a set of the form $\delta_G(X)$. A cut $\delta_G(X)$ *separates* vertex subsets A and B or is an A–B cut if (say) $A \subseteq X$ and (say) $B \subseteq V(G)\backslash X$. A *small* cut is one with at most k edges. If D is a digraph and $X \subseteq V(D)$, we write $\delta_D^+(X)$ for the set of edges with tail in X and head in $V(D)\backslash X$, and $d_D^+(X) = |\delta_D^+(X)|(\delta_D^-(X)$ and $d_D^-(X)$ are defined in a similar way). We say a set Z of edges is a *dicut* if for some $X, Z = \delta_D^+(X)$, and $\delta_D^-(X) = \emptyset$.

1. Blocking small cuts separating two vertices is NP-hard.

In this section we show that the problem $P_w(s,t,G)$ is NP-hard, even if $w(e) = 1$ for every edge e. More precisely, we show that the recognition problem: Given G, s, t, k, and $r \geq 0$, is $\tau_1(s,t,G) \leq r$?, is NP-complete.

The reduction is from the Steiner tree problem on graphs, defined as follows. Let H be a graph and $Z \subseteq V(H)$. A tree in H that spans Z (possibly together with other vertices) is called a *Steiner tree for* Z. Let n be an integer. Does H contain a Steiner tree for Z with at most n edges? This problem is NP-hard [GJ1]. We denote such an instance of the Steiner tree problem by (H, Z, n).

LEMMA 1.1. *The recognition version of the small cut blocking problem in* NP-*complete, even if all weights are polynomially bounded.*

Proof. Consider an instance (H, Z, n) of the Steiner tree problem. We next construct a problem $P_w(s,t,G)$. The graph G is defined by $V(G) = V(H) \cup \{t\}$, where t is a new vertex and $E(G)$ consists of $E(H)$, together with $u*$ copies of the edge $\{t, z\}$ (for each $z \in Z$), where $u* = |E(H)| + 1$.

Define w by

$$w(e) = 1 \text{ if } e \in E(H), \qquad w(e) = n + 1 \text{ otherwise.}$$

Set

$$k = u * |Z| - 1.$$

Finally, choose $s \in Z$ arbitrarily. Now consider the following three facts:

(i) $u * |Z| > k$. Consequently, if $\delta_G(X)$ is a cut separating s and t, with $d_G(X) \le k$, it must be the case that $(V(G) \backslash X) \cap Z \neq \emptyset$.

(ii) Suppose $Y \subseteq V(H)$ is such that $Y \cap Z \neq \emptyset$ and $(V(G) \backslash Y) \cap Z \neq \emptyset$. Without loss of generality, assume $s \in Y$. Then $\delta_G(Y)$ separates s from t and $d_G(Y) \le |E(H)| + |Y|u* \le |E(H)| + (|Z| - 1)u* = k$.

(iii) If $S \subseteq E(G)$ is such that $w(S) \le n$, then $S \subseteq E(H)$.

From (i) to (iii) we deduce that $\tau_w(s, t, G) \le n$ if and only if H contains a Steiner tree for Z with at most n edges. This concludes the proof. □

LEMMA 1.2. *The small cut blocking problem is* NP*-complete even with unit costs.*

Proof. Consider an instance $P_w(s, t, G)$, where $w(e) > 0$ for every $e \in E(G)$. Construct a new graph G' as follows. Every edge $e = \{x, y\} \in E(G)$ is replaced by a path $p(e)$ with ends x and y of length $w(e)$, with internal vertices not in $V(G)$. Further, for each pair $e \neq e'$, the set of internal vertices of $p(e)$ is disjoint from the corresponding set for $p(e')$. It is seen that $\tau_w(s, t, G) = \tau_1(s, t, G')$, which concludes the proof. □

We remark that the case $k = \lambda_G(s, t)$ can be solved in polynomial time. The algorithm makes use of a min–max relation and is sketched in §3.

2. The case of fixed k.

In this section we study the problems $P_w(s, t, G)$ in the case when k is fixed and show that there is a linear-time algorithm for such problems, i.e., an algorithm with complexity $O(|E(G)|)$. The constant in this $O(\,)$ is, in the worst case, a doubly exponential function of k. Hence, for $k \ge 10$ this algorithm may not be useful from a practical point of view. But, as stated before, in practice one would not be interested in such relatively high values of k. For $k \le 5$, we expect the algorithm to be computationally practicable.

2.1. Preliminaries.

We use the following terminology. If G is a graph, a *separation in* G is a triple $\{A, B, X\}$, where A and B are subgraphs of G such that $V(A) \cup V(B) = V(G), X = V(A) \cap V(B), E(A) \cap E(B) = \emptyset$, and $E(A) \cup E(B) = E(G)$. We say the separation is *at* X. If $|X| \le k$, we say the separation is *small.*

DEFINITION. Let G be a graph, and $\{A, B, X\}$ a separation in G. Let $Z \subseteq V(B)$ with $|Z| \le 2$, and let $R \subseteq E(G)$. The *B-consolidation of* G, *with respect to* R *and* Z, is the graph $G*$ obtained in the steps (i) and (ii) given next.

(i) For every $\emptyset \neq Y \subseteq X$ and each $z \in Z$, define a set $F(Y, z) \subseteq E(B) \backslash R$ as follows. If possible, choose W such that (1) $Y \subseteq W \subseteq V(B) \backslash ((X \backslash Y) \cup z)$, (2) $\delta_B(W) \cap R = \emptyset$, (3) $d_B(W)$ minimum subject to (1) and (2), and (4) W minimal subject to (1)–(3). If such a W exists *and* $d_B(W) \le k$, we set $F(Y, z) = \delta_B(W)$. Otherwise, we set $F(Y, z) = \emptyset$.

(ii) Let $F = \cup F(Y, z)$ (over all Y, z as in (i)). Obtain G^* by *contracting* (in G) all edges in $E(B) \backslash F$.

See Fig. 1 for an example.

Remark 2.1.1. Notice that G^* has a separation of the form $\{A^*, B^*, X^*\}$, where A^* is obtained from A by identifying elements of X (which maps X into X^*) and B^* is obtained from B by the edge contractions in the definition. For convenience, we regard $E(A^*)(E(B^*))$ as a subset of $E(A)$ (respectively, $E(B)$). Also notice that the number of edges in B^* is at most $k2^{|X|}$.

DEFINITION. A *type 1 small graph* is a connected graph H with a distinguished subset $X \subseteq V(H)$ and a distinguished vertex p, such that

(1) $|X| \le k$ and $|E(H)| \le k2^k$.

(2) $\lambda_H(p, X) > k$.

We say that X and p are the *boundary* and the *pole* of H, respectively.

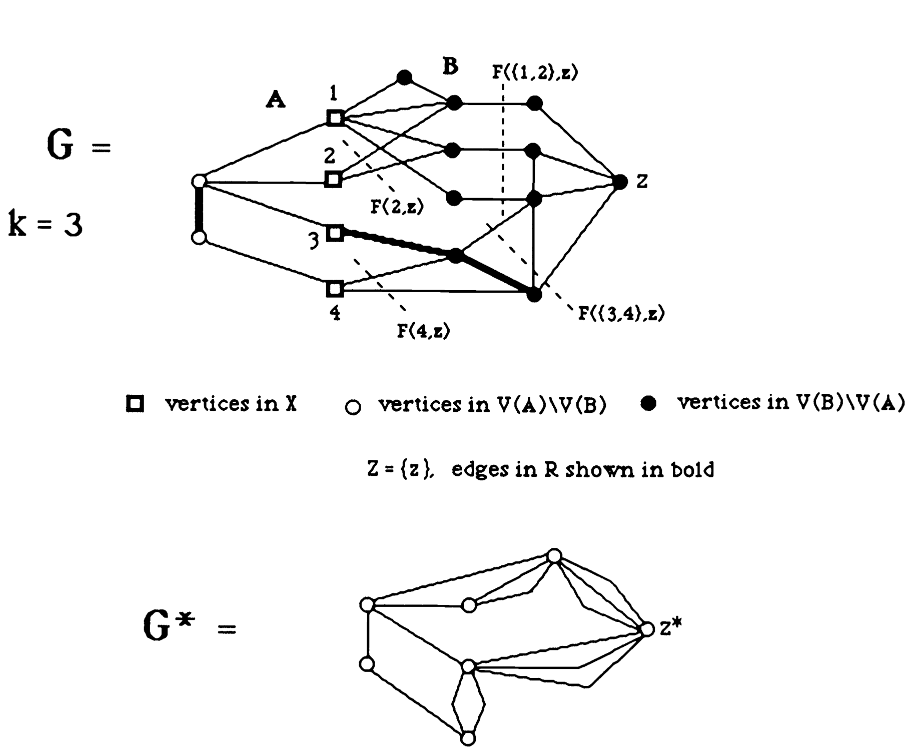


FIG. 1

Remark 2.1.2. Let $\{A, B, X\}$ be a small separation in a graph G, $R \subseteq E(G), p \in V(B)$. Let G^* be the B-consolidation of G (with respect to R and p) and consider the separation $\{A^*, B^*, X^*\}$ in G^*. Let $p^* \in V(B)$ be the image of p in G^*. If $\lambda_{B^*}(p^*, X^*) > k$, then B^* is a type 1 small graph.

In what follows, given a problem $P_w(s, t, G)$, a set $S \subseteq E(G)$ is called *feasible* if it intersects every small cut separating s and t.

PROPOSITION 2.1.1. *Let $\{A, B, X\}$ be a small separation in G such that A and B are connected, w is a weight function, $s \in V(A)$, and $t \in V(B)$. Let $S \subseteq E(G)$ be an optimal solution to the problem $P_w(s, t, G)$. Consider the problem $P_w(s^*, t^*, G^*)$, where G^* is the B-consolidation of G (with respect to S and t), t^* (respectively, s^*) is the image of t (respectively, s) in G^* and the weight function is extended by setting $w(e) = +\infty$ if $e \notin E(A)$. Finally, let $L^* \subseteq E(A)$ be an optimal solution to $P_w(s^*, t^*, G^*)$. Then*

(i) $\tau_w(s^*, t^*, G^*) = \sum\{w(e) : e \in S \cap E(A)\}$.

(ii) *$L^* \cup (S \cap E(B))$ is an optimal solution to $P_w(s, t, G)$.*

(iii) *B^* is a type 1 small graph, with boundary X^* and pole t^*.*

Proof. First we show (i) and (ii). We claim that

(a) $\tau_w(s^*, t^*, G^*) \leq \sum\{w(e) : e \in S \cap E(A)\}$.

Consider a small cut C^* separating s^* and t^* in G^*. Now in G, the edges of C^* form a small cut separating s and t. Consequently, since S is feasible for $P_w(s, t, G)$, $S \cap C^* \neq \emptyset$. Further, since $(S \cap E(B)) \cap C^* = \emptyset$ (or else at least one member of C would have been contracted in the construction of G^*) we must therefore have $(S \cap E(A)) \cap C^* \neq \emptyset$. Thus $S \cap E(A)$ is feasible for $P_w(s^*, t^*, G^*)$ and (a) follows.

Next we claim

(b) $\tau_w(s^*, t^*, G^*) \geq \sum\{w(e) : e \in S \cap E(A)\}$.

Write $R = S \cap E(B)$. To show (b) we prove that $L^* \cup R$ is a feasible solution to $P_w(s, t, G)$. Consider a small cut $C = \delta_G(D)$ separating s and t in G, with $s \in D$. If $C \subseteq E(A)$, then also $C \subseteq E(A^*)$ and it separates s^* and t^*; consequently $L^* \cap C \neq \emptyset$. Assume that $C \cap E(B) \neq \emptyset$. Write $D^X = D \cap X$ and $D^B = D \cap V(B)$. Then $\delta_B(D^B) = C \cap E(B)$ separates D^X from t in B. Suppose that $\emptyset = R \cap \delta_B(D^B) (= R \cap C)$. Since C is small, $d_B(D^B) \leq k$ and in the construction of G^*, we must have defined $F(D^X, t) = \delta_B(W)$, for some $W \subseteq V(B)$ with $d_B(W) \leq d_B(D^B)$ and $D^X \subseteq W$. Consequently, $(C \cap E(A)) \cup \delta_B(W)$ is the set of edges in a small cut in G^* separating s^* and t^*. By definition of L^* we then have that $L^* \cap (C \cap E(A)) \neq \emptyset$. We conclude, for any small cut C in G separating s and t, $(L^* \cup R) \cap C \neq \emptyset$. This settles (b). Hence (i) holds, and $L^* \cup R$ is an optimal solution to $P_w(s, t, G)$, i.e., (ii) holds as well.

In order to prove (iii) we have to show that $\lambda_{B^*}(t^*, X^*) > k$ (see Remark 2.1.2). If this does not hold, then there is a small cut $C^* \subseteq E(B^*)$ separating t^* and X^*. But then, in $G, C^* \subseteq E(B)$ separates t and X, and therefore it separates s and t, whereas $R \cap C^* = \emptyset$. This is impossible, since C^* is small. This concludes the proof. □

Remark 2.1.3. We stress that in Proposition 2.1.1 $\{B, A, X\}$ is also a small separation, and so a corresponding result holds with A and B interchanged.

Remark 2.1.4. Proposition 2.1.1 forms the basis of an algorithm for computing $\tau_w(s, t, G)$. Namely, construct a small separation $\{A, B, X\}$ of G as in the statement (and it is easy to see that such a separation exists if $\lambda_G(s, t) \leq k$). Then, recursively, solve the small cut blocking problem for all possible graphs obtained by "glueing" to A a type 1 small graph "at" X. Since $|X| \leq k$ the total number of such problems is bounded by a function of k. We keep track of an optimal solution to each such problem (call it an A-solution). Next, (recursively) we solve the small cut blocking problem for all possible graphs obtained by glueing to B a type 1 small graph, at X, and keep track of all such B-solutions. Finally, try all possible combinations of an A-solution and a B-solution, and from among all those combinations that are feasible for $P_w(s, t, G)$, select a minimum-weight one. Proposition 2.1.1 guarantees that this algorithm yields the correct answer. But in order to turn this into a polynomial-time algorithm, further tools are needed and will be given below. Nevertheless, Proposition 2.1.1 contains the main ideas in the algorithm we describe.

DEFINITION. A *type* 2 *small graph* is a graph H with a distinguished subset $Y \subseteq V(H)$ and a pair of distinguished vertices p_1, p_2, such that

(1) $|Y| \leq 2k$ and $|E(H)| \leq 2k2^{2k}$.

(2) If $i \neq j$, $\lambda_H(p_i, Y \cup p_j) > k$.

(3) Every vertex of H is in the same component as p_1 or p_2 (and hence H has at most two components).

We say that Y is the *boundary* and $\{p_1, p_2\}$ are the *poles* of H. Now we have the following analogue of Proposition 2.1.1.

PROPOSITION 2.1.2. *Consider a problem $P_w(s, t, G)$, where G has a separation $\{H, J, X\}$ with $s, t \in V(H)$ and $|X| \leq 2k$. Suppose J is connected and that every vertex of H is in the same component as s or t. Let S be an optimal solution to the problem $P_w(s, t, G)$. Let G^* be the H-consolidation of G with respect to S and $\{s, t\}$. Consider the problem $P_w(s^*, t^*, G^*)$, where s^* (respectively, t^*) is the image of s (respectively, t) in G^**

and the weight function w is as before extended by setting $w(e) = +\infty$ if $e \in E(H)$. Finally, let $R^* \subseteq E(J)$ be an optimal solution to $P_w(s^*, t^*, G^*)$. Then

(i) $\tau_w(s^*, t^*, G^*) = \sum\{w(e) : e \in S \cap E(J)\}$.

(ii) $(S \cap E(H)) \cup R^*$ *is an optimal solution to* $P_w(s, t, G)$.

(iii) H^* *is a type* 2 *small graph, with boundary* X^* *and poles* $\{s^*, t^*\}$.

Proof. The proof is similar to that of Proposition 2.1.1. □

2.2. The digraph of minimum cuts. Write $q = \lambda_G(s,t)$. Consider the graph Γ_q obtained from G by contracting every edge that is not in any s–t cut of size q (i.e., we contract the edges that do not appear in any min-cut). The following properties of Γ_q are well known. Suppose $\{u, v\} \in E(\Gamma_q)$. We orient $\{u, v\}$ as (u, v) if there is a minimum s–t cut $\delta_G(X)$ with $u \in X$ and $v \notin X$. This orientation is well defined (i.e., every minimum s–t cut containing $\{u, v\}$ yields the same orientation). The resulting digraph Δ has the properties (i)–(v) (where s' and t' are the images of s and t, respectively):

(i) Δ is acyclic.

(ii) Every dicut $\delta^+_\Delta(X)$ satisfies $|\delta^+_\Delta(X)| = q, s^* \in X$ and $t^* \notin X$.

(iii) The edges in every dicut $\delta^+_\Delta(X)$ form a minimum s–t cut in G and vice versa.

(iv) Every edge of Δ is contained in a directed path from s to t.

(v) $d^+_\Delta(x') = d^-_\Delta(t') = q, d^-_\Delta(s') = d^+_\Delta(t') = 0$.

We use the *topological ordering* of $V(\Delta)$. Here we number the vertices of Δ by $1, 2, \ldots, N$ (say), so that $s' = 1, t' = N$, and for $1 \leq i \leq N-1$, there are no edges (u, v) with $v \in \{1, \ldots, i\}$ and $u \in \{i+1, \ldots, N\}$. Although every acyclic digraph has such an ordering, notice that in our case it also implies that for $1 \leq i \leq N-1$, $d^+_\Delta(\{1, \ldots, i\}) = q$. In the algorithm given below, we use this numbering of the vertices, but apply it instead to the graph Γ_q.

2.3. Linear-time algorithm for computing $\tau_w(s, t, G)$. Given a subgraph A of G, we denote by $\nu_G(A)$ those vertices of A that are incident with edges not in A. For the case $k = q$, the problem $P_w(s, t, G)$ is easy. Here we want a minimum-weight set that blocks all dicuts in Δ, and the properties of this digraph imply that the elements of the blocker of the hypergraph of all s–t dicuts is $\{X$: there is an s–t path p where X = set of forward edges in $p\}$. The minimum-weight element of this blocker can be easily found as a shortest path problem in an auxiliary digraph.

In what follows, we assume $k > q$. Notice that each vertex i of Γ_q corresponds to a subset $V(i)$ of vertices of G (possibly a single vertex) contracted in the construction of Γ_q. We use the following notation (see Fig. 2 for an example).

For $1 \leq i \leq N$, $A(i)$ is the subgraph of G induced by $\cup\{V(j) : 1 \leq j \leq i\}$. For $1 \leq i \leq N$, $J(i)$ is the subgraph induced by $V(i)$. For $1 \leq i \leq N-1$, $B(i)$ is the union of the subgraph of G induced by $\cup\{V(j) : i < j \leq N\}$, together with all edges with only one end in $V(A(i))$.

Remark 2.3.1. For $1 \leq i \leq N-1$, G has a separation $\{A(i), B(i), X(i)\}$ (where $X(i) = \nu_G(A(i)) = \nu_G(B(i))$). This separation is small, since the number of edges of G with only one end in $V(A(i))$ equals $q \leq k$.

For $1 \leq i \leq N-1, D(i)$ is the union of $A(i)$, and all the edges with one end in $V(A(i))$ and the other in $V(i+1)$. For $2 \leq i \leq N-1$, $H(i)$ is $D(i-1) \cup B(i)$.

Remark 2.3.2. For $2 \leq i \leq N-1$, G has a separation $\{H(i), J(i), Y(i)\}$ (with $Y(i) = \nu_G(H(i)) = \nu_G(J(i))$). Further, $|Y(i)| \leq 2k$.

Remark 2.3.3. For $1 \leq i \leq N$, $J(i)$ is connected (as we obtained it by contracting edges of G into a single node of Γ_q). For $1 \leq i \leq N, A(i)$ is connected, and for $1 \leq i \leq N-1, B(i)$ and $D(i)$ are connected (this follows by property (v) of the topological

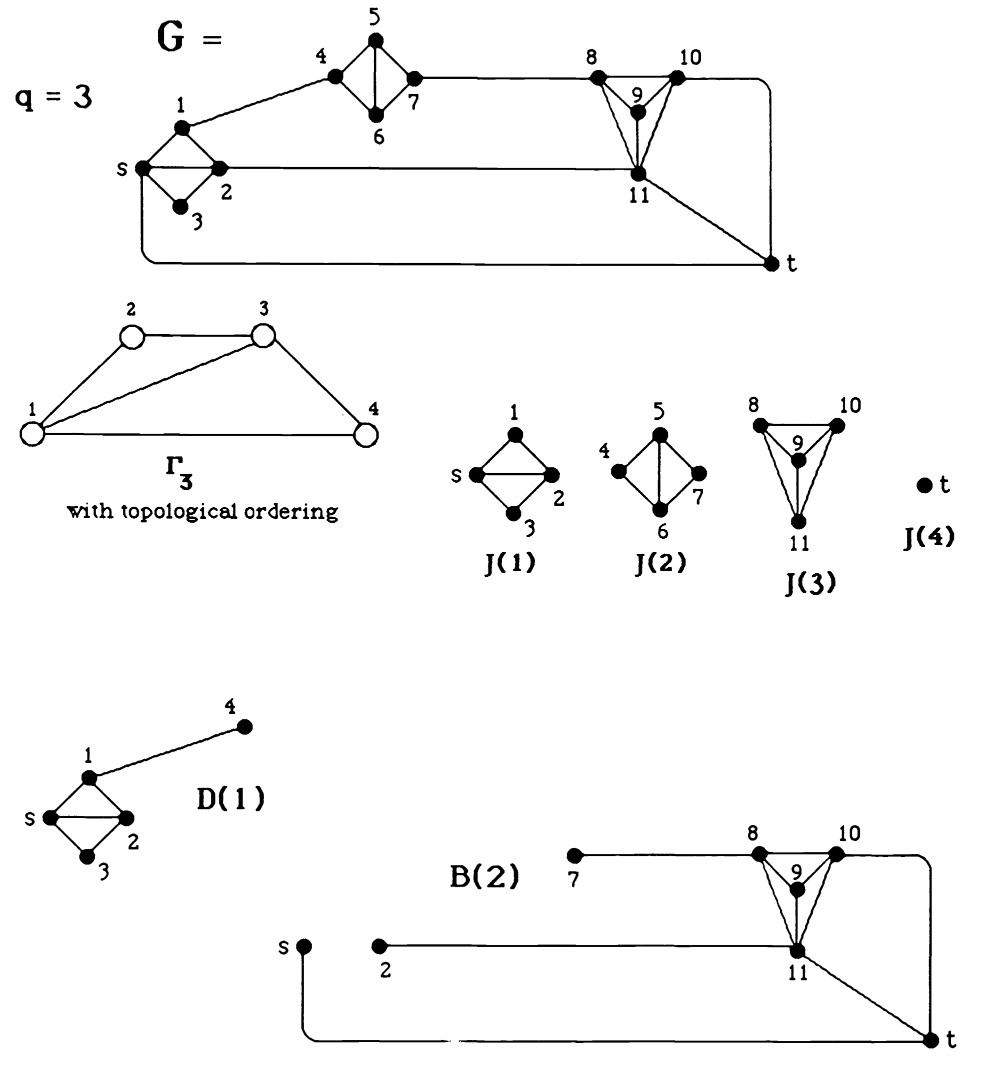


FIG. 2

ordering of $V(\Delta)$; see §2.2). Similarly, for $2 \leq i \leq N-1$, every vertex of $H(i)$ is in the same component of $H(i)$ as s or t.

Suppose S is a feasible solution to $P_w(s,t,G)$. For $1 \leq i \leq N-1$, consider the $B(i)$-consolidation of G with respect to S and t. Then by Proposition 2.1.1, its subgraph $B^*(i)$ is a type 1 small graph. Similarly, for $2 \leq i \leq N-1$, consider the $H(i)$-consolidation of G with respect to S and $\{s,t\}$. Then its subgraph $H^*(i)$ is a type 2 small graph. In fact, we can say more.

PROPOSITION 2.3.1. *Let S be a feasible solution to $P_w(s,t,G)$. Then*

(i) *Let I^* be the $B(1)$-consolidation of G with respect to S and t.*
Then $\lambda_{I^*}(s^*,t^*) \geq q+1$.

(ii) *For $2 \leq i \leq N-1$, let $M^*(i)$ be the $H(i)$-consolidation of G with respect to S and $\{s,t\}$. Then $\lambda_{M^*(i)}(s^*,t^*) \geq q+1$.*

Proof.

(i) Recall that I^* has a separation of the form $\{A^*(1), B^*(1), X^*(1)\}$. Consider any cut C separating s^* and t^* in I^*. If $C \subseteq E(B^*(1))$, then C separates t^* from $X^*(i)$ and thus $|C| \geq k+1 \geq q+1$ since $B^*(1)$ is a type 1 small graph. Suppose C intersects $E(A^*(1))$. Then C, now regarded as a cut in G separating s and t, intersects $E(A(1))$. But all edges in $A(1)$ were *contracted* in generating the graph Γ_q that contains all min-cuts. Thus again $|C| \geq q+1$.

(ii) This proof follows in a similar manner. □

Description of the algorithm. Consider the algorithm sketched in Remark 2.1.4. This algorithm works correctly, but is not necessarily polynomially bounded because the amount of recursive work may grow too fast. This obstacle is overcome by using the structure of Γ_q. Essentially, we solve all problems obtained by glueing to $A(i)$ each possible type 1 small graph at $X(i)$. Below we show that there is an easy inductive way to solve these problems for increasing i, so that the only recursive work we carry out involves graphs that are at least $q+1$ edge-connected between s and t. Since k is bounded, the total depth of recursions is bounded.

Our first step is to make formal the process of "glueing" two graphs. Let A be a graph and B be a type 1 small graph with boundary X. Let $Z \subseteq V(A)$, with $|Z| \geq |X| = r$, say. A *glueing of A and B, at Z*, is specified by a partition π of Z into r classes $\{C(1), \dots, C(r)\}$ and a labeling σ of B as $\{b(1), \dots, b(r)\}$. The glueing is the graph obtained by identifying (for $1 \leq j \leq r$) all vertices of $C(j)$ into a single vertex and then identifying this vertex with $b(j)$. We denote this graph by $A \otimes B$ (π and σ will be clear from the context). Suppose $s \in V(A)$, that t is the pole of B, and w is a weight function on $E(A)$. Let s^* (respectively, t^*) be the image of s (respectively, t) in the glueing $A \otimes B$. Then the *canonical problem* $P_w(s^*, t^*, A \otimes B)$ is that in which we extend w by setting $w(e) = +\infty$ if $e \in E(B)$. Similar definitions apply with regards to glueing a type 2 small graph and another graph.

Now we can describe our algorithm. The reader may already guess this algorithm, but we can formally describe it as follows.

```
Algorithm Block($s, t, G, w$)
Step 1  Test whether $q = \lambda_G(s,t) > k$. If so, set $\tau_w(s,t,G) = 0$ and stop.
Step 2  Construct the graph $\Gamma_q$.
Step 3  Execute algorithm Ext given below, which returns
        (a) an optimal solution to the canonical problem $P_w(s^*, u^*, D(N-1) \otimes B)$
        for every possible glueing of $D(N-1)$ and a type 1 small graph $B$ with pole
        $u$, at $\nu_G(D(N-1))$, such that $\lambda_{D(N-1)\otimes B}(s^*, u^*) \geq q+1$, and
        (b) an optimal solution to the canonical problem $P_w(t^*, v^*, J(N) \otimes A)$ for
        every possible glueing of $J(N)$ and a type 1 small graph $A$ with pole $v$, at
        $\nu_G(J(N))$, such that $\lambda_{J(N)\otimes A}(t^*, v^*) \geq q+1$.
Step 4  Set $\tau_w(s,t,G) =$
                $\min\{w(S_{N-1}) + w(R_N)$:
                $S_{N-1}$ is an optimal solution to one of the problems in (a),
                $R_N$ is an optimal solution to one of the problems in (b), and
                $S_{N-1} \cup R_N$ is feasible for $p_w(s,t,G)\}$
End
```

In the description of algorithm Ext, a *level i problem* $(1 \leq i \leq N-1)$ will be a canonical problem of the form $P_w(s^*, u^*, A(i) \otimes B)$, where B is a type 1 small graph with pole u.

```
Algorithm Ext
1 Recursively (i.e., using Block), solve every possible level 1 problem
  $P_w(s^*, u^*, A(1) \otimes B)$ such that $\lambda_{A(1)\otimes B}(s^*, u^*) \geq q+1$.
2 For $i = 1, 2, \ldots, N-1$, Do
  (a) Solve every problem $P_w(s^*, u^*, D(i) \otimes B)$, where $B$ is a type 1 small graph
  with pole $u$, and the glueing is at $\nu_G(D(i))$, as follows:
  $\tau_w(s^*, u^*, D(i) \otimes B) = \min\{w(L_i) + w(Z_i)$ :
        $L_i$ is an optimal solution to one of the level $i$ problems,
        $Z_i$ is a subset of the edges $\delta_G(V(A(i)), V(i+1))$, and
        $L_i \cup Z_i$ is feasible for $P_w(s^*, u^*, D(i) \otimes B)\}$.
[COMMENT: For $i = N-1$, (a) computes some the data required in step 3(a) of Block.]
  (b) If $i < N-1$, recursively solve every problem $P_w(u^*, v^*, J(i+1) \otimes H)$
  where $H$ is a type 2 small graph with poles $\{u, v\}$, such that $\lambda_{J(i+1)\otimes H}(u^*, v^*) \geq$
  $q+1$ and the glueing is at $\nu_G(J(i+1))$.
  (c) If $i < N-1$, solve each level $(i+1)$ problem $P_w(s^*, u^*, A(i+1) \otimes B)$ as
  follows:
  $\tau_w(s^*, u^*, A(i+1) \otimes B) = \min w(S_i) + w(J_{i+1})$
  such that
        $S_i$ is an optimal solution to one of the problems in (a),
        $J_{i+1}$ is an optimal solution to one of the problems in (b), and
        $S_i \cup J_{i+1}$ is feasible for $P_w(s^*, u^*, A(i+1) \otimes B)$.
  End For
3 Recursively, solve the problem $P_w(t^*, v^*, J(N) \otimes A)$ for every possible type 1
  small graph $A$ with pole $v$, such that $\lambda_{J(N)\otimes A}(t^*, v^*) \geq q+1$.
End
```

The correctness of algorithm Block follows from Proposition 2.1.1 (also recall Remarks 2.1.3 and 2.1.4, 2.1.2, and 2.3.1). Next we study its run time and show that it is linear in $|E(G)|$ for every fixed value of k. To this effect, we set $T(m, q)$ to be the maximum amount of work required by Block to solve a problem $P_w(s, t, G)$ when $\lambda_G(s, t) = q$. We will show (by induction on $k - q$) that

$$(*) \qquad T(m, q) \leq f(k-q)m + g(k-q),$$

for certain functions f and g. We have

(i) Step 1 of Block requires $O(km)$ time. This follows because it is not necessary to compute $\lambda_G(s, t)$; rather, we want to see if G has at least $k+1$ edge-disjoint paths from s to t, and any "augmenting path" algorithm does so in $O(km)$ time (see [T]). Consequently, $T(m, q) = O(km)$ for any m and $q > k$.

(ii) To construct Γ_q we must identify those edges of G that are contained in minimum s–t cuts, contract all others, and obtain the topological ordering. It is not difficult to see how to implement the last two tasks in $O(m)$ time. Further, the identification can be implemented in $O(qm^2)$ time; but as we show in Appendix A, with some care this step can be carried out in $O(q^3m)$ time.

(iii) In Step 4 of Block, each feasibility test requires $O(km)$ work, and if we write $n(k)$ = number of type 1 or type 2 small graphs, then the total number of tests is at most $(n(k))^2$.

(iv) Next we analyze the complexity of one call to Ext.

Consider first the recursive work. The recursive work appears in step 1 of Ext (involving graph $A(1) = J(1)$), in step 2(b) (involving $J(i)$, where $2 \leq i \leq N-1$) and in step 3, involving $J(N)$. Now any (type 1 or 2) small graph has at most $2k2^{2k}$ edges. So if $J(i)$ has $m(i)$ edges, then any recursive call to Block involving $J(i)$ is on a graph with at most $2k2^{2k}m(i)$ edges. Further, the different graphs $J(i)$ are edge-disjoint. Finally, the recursive calls are on graphs with edge-connectivity at least $q+1$ between the two relevant vertices. Hence, the total recursive work in one call to Ext is at most

$$\sum \left\{n(k)T(2k2^{2k}m(i), q+1) : 1 \le i \le N\right\},$$

which by induction (using (*)) is at most

$$n(k)2k2^{2k}f(k-q-1)m + n(k)2k2^{2k}g(k-q-1).$$

Now for the nonrecursive work in Ext. The major part of it is made up of the feasibility tests in steps 2(a) and 2(c). It is not difficult to see that *each* test can be carried out in $O(km)$ time, but this would lead to a bound $O(m^2)$ bound for all the tests. As we show in Appendix B, with some care we can attain an overall $O(n^2(k)m)$ bound for the feasibility tests.

(v) In summary, we obtain the following bound for $q \le k$:

$$\begin{aligned} T(m,q) \le O(q^3m) + O(n^2(k)m) + n(k)2k2^{2k}f(k-q-1)m + n(k)2k2^{2k}g(k-q-1) \\ + O(n^2(k)m) \le f(k-q)m + g(k-q), \end{aligned}$$

for appropriately defined f and g, as desired. This completes the proof of (*).

Remark 2.3.4. The function $n(k)$ is doubly exponential in k. From the analysis above, we conclude that the overall algorithm for $P_w(s,t,G)$ has complexity $O(h(k)m)$, where h is doubly exponential in k.

3. The linear programming relaxation. The small cut blocking problem $P_w(s,t,G)$ can be equivalently restated as the following set covering problem:

$$\tau_w(s,t,G) = \min \sum \{w(e)x(e) : e \in E(G)\}$$

such that

$$Ax \ge 1,$$

$$x \ge 0 \quad \text{integral},$$

where A is the incidence matrix of small s–t cuts versus edges of G, and 1 is the column of 1's. Even for k moderately small it may be convenient to view the problem $P_w(s,t,G)$ in this manner, since we may be able, for example, to obtain good approximations faster than the run time of the (theoretically efficient) exact algorithm described in §2. Indeed, cutting plane algorithms have been successfully used to effectively solve many types of combinatorial optimization problems, and one hopes that that would be the case here as well. Moreover, an understanding of the polyhedral structure of this set covering problem would probably be useful with regards to the general version $P_w(S,G)$ of the small cut blocking problem (which is NP-hard even for small k, as shown in the next section).

An important element in developing a cutting-plane algorithm for solving the integer program above would be the efficient solution of its linear-programming relaxation, which we denote $P_w(s,t,G)^*$, with value $\tau_w(s,t,G)^*$. The fact that the matrix A is potentially very large presents a difficulty: Even for k small, A could have many rows, and the practical computation of $\tau_w(s,t,G)^*$ could be costly (although theoretically efficient). For general k, it is not even clear how to solve the linear program in polynomial time. One wonders in that case if there is a *compact* reformulation of the linear program (one where the constraint matrix has polynomially many rows and columns). The main result in this section is negative: for general k, it is NP-hard just to compute the quantity $\tau_w(s,t,G)^*$ (and therefore, unless $P = \text{NP}$, there is no compact reformulation, even allowing for different variables).

We denote by $B(s,t,G)$ the polyhedron (where $d = |E(G)|)\{x \in \mathbb{R}^d : Ax \geq 1, x \geq 0\}$. Given a polyhedron $Q \subseteq \mathbb{R}^n$, the *separation problem* for Q is the following: Suppose $x \in \mathbb{R}^n$. Then either decide $x \in Q$ or provide a hyperplane that separates x and Q.

In the proof below, we make use of the *max-cut* problem. In this problem (whose recognition version is NP-complete [GJ1]), we are given a simple graph H and asked to produce a subset $x \subseteq V(H)$ such that $d_H(X)$ is maximum.

LEMMA 3.1. *The separation problem for polyhedra $B(s,t,G)$ is NP-hard.*

Proof. Consider an instance of the recognition version of the max-cut problem with graph H, where we seek cut of size $\geq L$. We assume $|E(H)| > L$ and H nonbipartite (otherwise the max-cut problem is trivial). We construct a graph G as follows. First, let $\bar{H}$ denote the complement graph of H, and set $m = |E(\bar{H})|$. Then $V(G) = V(H) \cup \{s,t\}$. $E(G)$ contains $E(\bar{H})$, as well as $m+2$ copies of the edge $\{s,v\}$ and one copy of the edge $\{t,v\}$, for every $v \in V(H)$.

We next define a vector $\mathbf{x}$. Write $n = |V(H)|$. Choose a *fixed* $u \in V(H)$. For $e \in E(G)$, set

$$x(e) = (n(n-1)/2 - L + 1)^{-1} \quad \text{if } e \in E(\bar{H}), \text{and}$$

$$x(e) = 0 \quad \text{if } e \text{ is incident with } s \text{ or } t, \text{ with one exception: } x(\{u,t\}) = 1.$$

Finally, set $k = (m+2)n - 1$.

Suppose $\delta_G(Y)$ is a cut separating s and t with $s \in Y$. Then $k < (m+2)n$ implies that if this cut is small, we must have $V(H) \cap Y \neq \emptyset$. Moreover, if $x(\delta_G(Y)) < 1$, then also $V(H) \cap V(G)\backslash Y \neq \emptyset$, and moreover, the number of edges of $\bar{H}$ with one end in $V(H) \cap Y$ and the other in $V(H) \cap (V(G)\backslash Y)$ must be at most $n(n-1)/2 - L$. In summary, if $\delta_G(Y)$ is a small cut with $x(\delta_G(Y)) < 1$, then $d_H(V(H) \cap Y) \geq L$, and the converse to this follows in a similar way. Hence, $x \notin B(s,t,G)$ if and only if there is a $W \subseteq V(H)$ with $d_H(W) \leq L$. This concludes the proof. □

Now consider a problem $P_w(s,t,G)$ with $d = |E(G)| \geq 1$. Let $\mathbf{e} \in \mathbb{R}^d$ be the vector of 1's. Clearly $\mathbf{e} \in B(s,t,G)$. Let $Q \subseteq \mathbb{R}^d$ be the polyhedron defined by $Q = \{z \in \mathbb{R}^d : z(x - \mathbf{e}) \leq 1 \text{ for all } x \in B(s,t,G)\}$. Q is called the *polar* of $B(s,t,G)$. It is not difficult to see that Q is well behaved: It contains a full-dimensional sphere with center $(-1/d)\mathbf{e}$ and polynomially bounded inverse radius. Using well-known results concerning the ellipsoid method [GLS], Lemma 3.1 implies that it is NP-hard to compute the value $\tau_w(s,t,G)^*$.

COROLLARY. *Unless $P = \text{NP}$, there is no compact reformulation of $P_w(s,t,G)^*$.*

How well does $\tau_w(s,t,G)^*$ approximate $P_w(s,t,G)$? We conjecture that on the average the approximation will be fairly good. However, in the worst case it is quite poor, as we show next.

LEMMA 3.2. *There exist examples with $\tau_1(s,t,G)/\tau_1(s,t,G)^*$ arbitrarily large.*

Proof. Let m be a positive integer. Consider the hypergraph of $(m+1)$ subsets of $\{1,2,\ldots,2m\}$ and let $S(m)$ denote the set covering problem whose corresponding matrix is the edge-vertex incidence matrix of this hypergraph. It is not difficult to see that the value of $S(m)$ is m, but that the value of the LP-relaxation is (only) $2m/(m+1)$ (*Remark*: This example is similar to one mentioned in [S].) Next we construct a graph G. We begin with vertices s, t and $\{1, 2, \dots, 2m\}$ and for $1 \leq i \leq 2m$, the edge $\{i, t\}$ and two copies of $\{s, i\}$. Next, we subdivide each edge incident with s, $2m-1$ times (i.e., each such edge is replaced by a path of length $2m$). The resulting graph is G. Color all edges of G incident with t blue, and all other edges, red. Set $k = 3m-1$, and consider $P_1(s, t, G)$.

We observe: (1) Suppose $\delta_G(Z)$ is a minimal s–t cut with $s \in Z$. Write $z = |Z \cap \{1, 2, \dots, 2m\}|$. Then $d_G(Z) = 4m - z$, and consequently, $\delta_G(Z)$ is small if and only if $z \geq m+1$. Moreover, (2) Suppose x is an optimal solution to either $P_1(s, t, G)$ or $P_1(s, t, G)^*$, chosen so that in addition $x(R)$ is minimized, where R is the set of red edges. It is not difficult to see that in this case $x(e) = 0$ for every red edge e (x must be constant on each "subdivided edge" between s and some i). (1) and (2) imply that $P_1(s, t, G)$ and $P_1(s, t, G^*)$ are isomorphic to $S(m)$ and its LP-relaxation, respectively. Thus $\tau_1(s, t, G)/\tau_1(s, t, G)^* = (m+1)/2$, as desired. □

We note that the lower bound achieved in Lemma 3.2 grows linearly in k. This is best possible since the rows of the matrix A have at most k 1's (see [HH]).

In spite of the negative results given in this section, one hopes that an effective cutting plane algorithm can be developed for problems $P_w(s, t, G)$, and this is an interesting topic which we plan to study shortly. We remark that there is a different formulation of the problem $P_w(s, t, G)$, as a mixed-integer program. Here we have, besides the 0–1 variables $x(e)$, additional *flow* variables for the edges. Then we want to select a minimum-weight subset of edges, each of which will be assigned *capacity* $k+1$ (and all other edges are assigned capacity 1), so that in the resulting network there is a flow of $k+1$ units from s to t. This program is compact, but its LP-relaxation can be far weaker than the one we considered above (and it is never better).

There is one special case where the linear program described above has integer solutions and can be solved in polynomial time. This is the case where $k = \lambda_G(s, t)$. Recall the digraph of minimum cuts Δ described in §2.2. This digraph has the property that every dicut $\delta +_\Delta (X)$ satisfies $|\delta +_\Delta (X)| = k, s^* \in X$ and $t^* \notin X$ and conversely. We use this fact as follows: For a given undirected $s-t$ path p in Δ, define $F(p) = \{e : e \text{ is a forward edge in } p\}$. Then, as stated in §2.3, it can be verified that the blocker of the clutter of dicuts in Δ (in this case) is the set of all sets $F(p)$. Using this fact, one can reduce the problem $P_w(s, t, G)$ to a shortest path problem in a related digraph. Further, this fact yields a min–max relation: $\tau_w(s, t, G)$ is equal to the maximum number of s–t cuts of size k (in G), such that no edge e is contained in more than $w(e)$ of the cuts.

4. Blocking small cuts between many vertices.

In this section we consider the small cut blocking problem, in the more general case $P_w(S, G)$ where we are interested in the cuts or cardinality $\leq k$ separating pairs of vertices in some set $S \subseteq V(G)$. One might expect that for k fixed this problem would be polynomially solvable. Indeed, at first glance it may appear that the algorithm in §2 could be generalized, so that instead of using a linear ordering of the graph (that decomposes the graph into small separations) one would use a tree-decomposition of small width (see [BLW] for an early reference) and then proceed with dynamic programming as in §2.

However, the problem is intractable even for small values of k. In fact, we show that it is NP-hard for $k = 8$ (and, thus, for any $k \geq 8$). On the other hand, using arguments very similar to those in §2, one obtains a linear-time algorithm for the case ≤ 2. We remark that this particular algorithm may be practical and also useful (since with $k = 2$

we effectively achieve 3-edge connectivity between vertices of S). Similarly, there is a linear-time algorithm for every fixed k *and* fixed $|S|$ (the complexity is singly exponential in $|S|$). We omit the description of this algorithm, which is essentially a more complicated version of that given in §2 (we use small graphs with many poles).

In the rest of this section we show that the class of instances $P_1(S, G)$ is NP-hard for $k = 8$. We show that the general weighted version is NP-hard; the rest of the proof (unit weights) proceeds as in §1.

The reduction is from the vertex cover problem in cubic graphs. Given a cubic graph H (i.e., H 3-regular) we seek a subset $X \subseteq V(H)$, with $|X|$ minimum, such that $X \cap e \neq \emptyset$ for every edge e. The fact that this problem is NP-hard can be deduced from the results in [GJ2].

Thus, consider an instance of the vertex cover problem in a cubic graph H on at least three vertices, where we seek a cover with at most N vertices. We next construct a problem $P_w(S, G)$. The graph G is obtained as follows.

First, for every vertex x of H (incident, say, with edges a, b, and c), G has three vertices $x(a)$, $x(b)$, and $x(C)$, which induce a triangle. We write $T(x) = \{x(a), x(b), x(c)\}$.

Next, for every edge $a = \{x, y\}$ of H, G has a vertex s_a, and the edges $\{s_a, x(a)\}$ and $\{s_a, y(a)\}$.

All edges f described so far have $w(f) = 1$.

Finally, there is a vertex s^* and two copies of $\{s^*, x(a)\}$ for every $x \in V(H)$ and $a \in E(H)$ such that x and a are incident. We also have six copies of the edge $\{s^*, s_a\}$ for every $a \in E(H)$. All edges incident with s^* have weight $2|V(H)| + N + 1$. We set $S = V(G)$, and as we said above, $k = 8$.

The following is easily obtained by counting:

CLAIM 4.1. *Consider an inclusion minimal cut $\delta_G(Z)$ with $d_G(Z) \leq 8$, where $s^* \notin Z$. Then either* (i) $Z = \{s_a\}$, *for some* $a = \{x, y\} \in E(H)$, *or* (ii) *for some* $x \in V(H)$, $Z \subseteq T(x)$ *and* $|Z| \leq 2$. *Conversely, all Z as in* (i) *or* (ii) *have* $d_G(Z) \leq 8$.

Let us color all edges of G contained in triangles $T(x), x \in V(H)$, red; the edges incident with s^*, blue; and all other edges, green.

CLAIM 4.2. *Suppose we can choose an optimal solution to $P_w(S, G)$ with no blue edges. Let B be such an optimal solution, such that in addition B has a minimum number of red edges. Then*

(i) *Let $x \in V(H)$, and suppose that B includes at least one green edge with one end in $T(x)$. Then B includes all green edges with one end in $T(x)$, and $B \cap E(T(x)) = \emptyset$. Conversely, if $B \cap E(T(x)) = \emptyset$, then B includes all green edges with one end in $T(x)$.*

(ii) *If $B \cap E(T(x)) \neq \emptyset$, then B includes at least two edges with both ends in $T(x)$.*

(iii) *The set $\{x \in V(H) : B$ includes all green edges with one end in $T(x)\}$ is a vertex cover in H.*

Proof. (i) Let a, b, and c be the three edges incident with x in H. Suppose first that (say) B includes $\{s_a, x(a)\}$ but not $\{s_b, x(b)\}$. Consider the cuts $\delta_G(X)$, where $X = \{x(b)\}$ or $X = \{x(b), x(c)\}$. In order for B to meet these two cuts, it must be the case that either B includes $\{s_c, x(c)\}$ and at least one edge from $T(x)$ or that B includes at least two edges from $T(x)$. In either case, $B' = (B \backslash E(T(x))) \cup \{\{s_b, x(b)\}, \{s_c, s(c)\}\}$, then B' is an optimal solution to $P_w(S, G)$ with no blue edges and fewer red edges than B, a contradiction. Consequently B includes $\{s_b, x(b)\}$ and $\{s_c, x(c)\}$, and the fact that $B \cap E(T(x)) = \emptyset$ now follows immediately. The converse statement follows from the first. (ii) (respectively, (iii)) follows from (i), and the fact that for all $x(a), d_G(x(a)) \leq 8$ (respectively, for all $s_a, d_G(s_a) \leq 8$). □

CLAIM 4.3. *Suppose S is a vertex cover for H. Define $B \subseteq E(G)$ as follows: For each $x \in V(H)$, if $x \in S$, then B includes all green edges with one end in $T(x)$, otherwise B includes two edges with both ends in T_x (arbitrarily chosen). Then B is a feasible solution to $P_w(S, G)$, and $w(B) = 2|V(H)| + |S|$.*

Proof. The proof follows from Claim 4.1. □

Putting together the above claims, we obtain the following theorem:

THEOREM 4.4. *H has a vertex cover of cardinality at most N, if and only if $\tau_w(S, G) \leq 2|V(H)| + N$.*

Proof. The only observation needed is that the bound on $\tau_w(S, G)$ excludes the edges incident with s^* from being used. □

5. Conclusion and related problems. One important future task is to better understand the problem $P_w(S, G)$ for small values of k. We conjecture that it is NP-hard already for $k = 3$. On the other hand, does there exist a polynomial-time algorithm for this problem, with fixed performance bound? Because of the similarity with the set covering problem, we again conjecture that the answer is negative. However, there is hope that a cutting-plane algorithm could be effective here from a practical point of view, as discussed in §3. Some early computational experience is promising [BD]. A polyhedral study of the problem would be the first step in this direction, and this will be our next area of work.

A problem that is seemingly related to blocking small cuts is that of blocking *short paths* between two vertices s and t in a graph. Here an adversary has to pay a price for removing any edge from the graph, and the adversary wants to remove edges (at minimum cost) so as to make the distance from s to t larger than some given amount h. For $h \leq 3$, this problem can be solved in polynomial time by reducing it to a matching problem. However, for $h \geq 5$, it is NP-hard (the proof of this fact uses essentially the same paradigm as the proof in §4). Here, again, one expects a cutting-plane algorithm to be effective.

Appendix A. In this appendix we provide a linear-time algorithm for finding those edges that are contained in min-cuts. More precisely, given G, selected vertices s and t, $q = \lambda_G(s, t)$ and $m = |E(G)|$, in time $O(q^3 m)$ the algorithm below finds which edges of G are contained in s–t cuts of size q (for short we will refer to these cuts as min-cuts). We call such edges *critical.*

In what follows, we use the following convention: Any time that we speak of a cut $\delta_A(X)$ separating s and t (in some graph A), we assume that $s \in X$. First, we make some observations.

If $\delta_G(X)$ and $\delta_G(Y)$ are min-cuts, then so are $\delta_G(X \cap Y)$ and $\delta_G(X \cup Y)$ (this follows using the submodular inequality). Hence, if $Z \subseteq V(G)\backslash t$ is such that $\lambda_G(Z \cup \{s\}, t) = q$, then there is a *unique minimal* subset $W \subseteq V(G)$, with $\{s\} \cup Z \subseteq W$, such that $\delta_G(W)$ is a min-cut. We call this cut the *Z-minimal min-cut* (and a similar definition is possible with maximal instead of minimal cuts). It is also seen that

(A.1) if $\delta_G(X)$ and $\delta_G(X')$ are, respectively, the Z- and Z'-minimal min-cuts, then $\delta_G(X \cup X')$ is the $(Z \cup Z')$-minimal min-cut.

Consequently, let $\delta_G(S)$ be an s-minimal min-cut (in other words, a min-cut closest to s), and $\delta_G(V(G)\backslash T)$ an s-maximal min-cut. By contracting S and T, respectively, into single vertices s' and t', we have that in the resulting graph s' and t' have degree q. For convenience, we retain the G, s, t notation for the new graph. Hence, we have the following:

(A.2) Without loss of generality, $\delta_G(s)$ and $\delta_G(X(G)\backslash t)$ are min-cuts.

Next, consider a family $\{p_1, p_2, \ldots, p_q\}$ of q pairwise edge-disjoint s–t paths in G. Clearly every critical edge is contained in one of these paths, and each min-cut C and each path p_i intersect at precisely one edge. Now suppose that for some vertex $w \neq s, t, r > 1$ of these paths are incident with w. For simplicity assume these are paths p_i, $1 \leq i \leq r$, and that the edges of p_i incident with w are $\{u(i), w\}$ and $\{w, v(i)\}$. We modify our graph as follows. First, we remove w, and then we add r new vertices $w(i), 1 \leq i \leq r$, which induce a tree (say, a star). Next, for each i we add the edges $\{u(i), w(i)\}$ and $\{w(i), v(i)\}$. It is clear that in the new graph there is a system of q edge-disjoint paths each of which is incident with a different vertex $w(i)$ and that the s–t edge connectivity has not changed. Furthermore, the set of critical edges in the new graph is isomorphic to the set of critical edges in the old graph (this follows because the $w(i)$ induce a connected graph). Hence, we have the following:

(A.3) We may assume that the paths $p_i, 1 \leq i \leq q$, are pairwise internally vertex disjoint.

We point out that the transformation above increases the number of edges by at most a factor of 2. Denote by $|p_i|$ the number of edges in $p_i, 1 \leq i \leq q$. Let us color the edges in the paths p_i green and all other edges, red. Next, we label each vertex in p_i by $p_i[j]$, where $0 \leq j \leq |p_i|$ is the distance from s to the vertex along p_i.

Our approach is as follows: For each i and j, we compute the $p_i[j]$-minimal min-cut $C_j[j]$. This is useful since the green edge $\{p_ij, p_i[j+1]\}$ is critical if and only if it belongs to $C_i[j]$. Recall that any min-cut intersects any path p_i on at least one edge and therefore on precisely one edge. Hence, we can describe any min-cut by specifying on which edge it intersects each path p_i. Thus we can describe each min-cut with $O(q)$ data.

We use the following notation. Let $1 \leq a, b \leq q$ (possibly $a = b$). For each vertex $p_a[j]$, we define a number $f(p_a[j], b)$. First, let

$$R(p_a[j], b) = \{g : \text{ there is a } \textit{red} \text{ edge } \{p_a[h], p_b[g]\} \text{ with } h \leq j\}.$$

Next, set

$$g(p_a[j], b) = \max\{g \in R(p_a[j], b)\}, \quad \text{if } R(p_a[j], b) \neq \emptyset \quad \text{ and } g(p_a[j], b) = 0 \text{ otherwise.}$$

Finally,

$$f(p_a[j], b) = g(p_a[j], b) \quad \text{if } a \neq b, \qquad \text{and } f(p_a[j], a) = \max\{j, g(p_a[j], a)\}.$$

Later we shall show how to compute all numbers $f(p_a[j], b)$ in $O(q^2m)$ time.

The algorithm we use to compute the cuts $C_i[j]$ is inductive. For $1 \leq h \leq q$, let $G[h]$ denote the subgraph of G induced by vertices in the paths $p_i, 1 \leq i \leq h$. It is seen that $\lambda_{G[h]}(s, t) = h$. For each vertex v in $G[h]$, let $A_h(v)$ denote the v-minimal min-cut in $G[h]$. Inductively (that is, in increasing values of h), our algorithm computes all the cuts $A_h(p_i[j])$, where $1 \leq i \leq h$ and $0 \leq j \leq |p_i|$. For $h = q$, this yields the cuts we desire $(A_q(p_i[j]) = C_i[j])$.

An algorithm for computing the cuts $A_h(p_i[j])$. Consider first the case $h = 1$. Then $\lambda_{G[1]}(s, t) = 1$, and it is seen that for each j, the cut $A_1(p_1[j])$ consists of the edge $\{p_1[z], p_q[z+1]\}$, where $z = f(p_1[j], 1)$.

Now for the general inductive step. Suppose we have computed all the cuts $A_{h-1}(p_i[j])$. Next we compute all cuts $A_h(p_i[j])$, beginning with the cuts $A_h(p_h[j])$. Fix j. For $1 \le i \le h-1$, write $\Phi(j,i) = f(p_h[j], i)$ and let $B(i,j) \subseteq V(G(h-1))$ be such that $\delta_{G[h-1]}(B(i,j))$ is the $p_i[\Phi(j,i)]$-minimal min-cut in $G[h-1]$. Define $B_j = \cup\{B(i,j) : 1 \le i \le h-1\}$. Then (refer to A.1) we have the following:

$$\text{(A.4)}\qquad \begin{aligned}&\delta_{G[h-1]}(B_j) \text{ is the } \{p_1[\Phi(j,1)],\\ &p_2[\Phi(j,2)], \dots, p_{h-1}[\Phi(j,h-1)]\} - \text{minimal min-cut in } G[h-1].\end{aligned}$$

We compute the cuts $A_h(p_h[j])$ in *decreasing* order of j. Thus, let j^* be such that $p_h[j^*+1] = t$. We have the following claims:

CLAIM 1. $A_h(p_h[j^*]) = \delta_{G[h-1]}(B_{j^*}) \cup \{p_h[j^*], p_h[j^*+1]\}$.

Proof. It is clear that $\delta_{G[h-1]}(B_{j^*}) \cup \{p_h[j^*], p_h[j^*+1]\}$ is a min-cut in $G[h]$ (recall (A.2)). Now let $X \subseteq V(G[h])$ be such that $\delta_{G[h]}(X) = A_h(p_h[j^*])$, and let $X' = X \cap V(G[h-1])$. Now $\delta_{G[h-1]}(X') = A_h(p_h[j^*]) \cap E(G[h-1])$ is a min-cut in $G[h-1]$. Further, $p_i[\Phi(j^*, i)] \in X$ for $1 \le i \le h-1$. Therefore $B_{j^*} \subseteq X'$, and then the minimality of $A_h(p_h[j^*])$ and (A.4) imply that $B_{j^*} = X'$, which proves the claim. □

Next, assume we have computed all the cuts $A_h(p_h[j'])$ for each $j' > j$. Define $F_j = \max\{f(p_i[\Phi(j,i)], h) : 1 \le i \le h-1\}$. We have the following claim:

CLAIM 2. $A_h(p_h[j]) = A_h(p_h[F_j])$ *if* $F_j > j$, $A_h(p_h[j]) = \delta_{G[h-1]}(B_j) \cup \{p_h[j], p_h[j+1]\}$ *otherwise.*

Proof. The proof follows by an argument similar to that in the proof of Claim 1. □

Finally, we compute all cuts $A_h(p_i[j]), 1 \le i < h$, as follows. Fix $i < h$, and consider any vertex $p_i[j], 0 \le j \le |p_i|$. Let $D \subseteq V(G[h-1])$ be such that $\delta_{G[h-1]}(D) = A_{h-1}(p_i[j])$. Set $H = \max\{f(v,h) : v \in D\}$ and let $X \subseteq V(G[h])$ be such that $\delta_{G[h]}(X) = A_h(p_h[H])$. Then we have the following claim:

CLAIM 3. $A_h(p_i[j]) + \delta_{G[h]}(D \cup X)$.

Proof. First, $\delta_{G[h]}(D \cup X)$ is a min-cut in $G[h]$. This follows because this cut contains no *red* edges. Moreover, if X' is such that $\delta_{G[h]}(X') = A_h(p_i[j])$. Then clearly $D \subseteq X' \cap V(G[h-1])$. Thus $D \cup X \subseteq X'$, which proves the claim. □

Complexity of the algorithm for computing the cuts $A_h(p_i[j])$. We next show that the total workload in computing all the cuts $A_h(p_i[j]), 1 \le h \le q$, is $O(q^3 m)$. First we show how to compute all numbers $g(p_a[j], b)$ in $O(q^2 m)$ time (from which we immediately have the numbers $f(p_a[j], b)$). For fixed a and b, this is easily done for *increasing* values of j. Suppose we know $g(p_a[j], b)$. Then

$$g(p_a[j+1], b) = \max\{g(p_a[j], b), \max\{d : \{p_a[j], p_b[d]\} \text{ is a red edge}\}\},$$

and a similar formula yields $g(p_1[0], b)$. Notice that for given a and b, the total amount of work involved in computing all numbers $g(p_a[j], b)$ is (at most) proportional to the number of edges with one end in p_a. Hence the computation of all the $g(p_a[j], b)$ (over all a, b, and $0 \le j \le |p_a|$) takes time $O(q^2 m)$ (with more care it can be done in $O(m)$, but that is not crucial here).

Next, recall that if a min-cut $\delta_{G[h]}(X)$ in $G[h]$ contains an edge $\{p_i[j], p_i[j+1]\}$, then X contains all vertices $p_i[r], 1 \le r \le j$, and consequently, we can compactly describe any min-cut by specifying just what edge of each path p_i it contains (i.e., using $O(h)$ data in $G[h]$). As a result, suppose, for example, that $\delta_{G[h]}(X)$ and $\delta_{G[h]}(Y)$ are min-cuts in $G[h]$, described as just stated. Then we can compute the min-cut $\delta_{G[h]}(X \cup Y)$ in $O(h^2)$ time (by computing the maxima of h sets of h numbers). Using this fact, it is not difficult to see that the operations leading to Claims 1–3 can be implemented in time $O(h^2m)$, assuming we already have the $A_{h-1}(p_i[j])$ data.

Consequently, in $O(q^3m)$ time we can compute all cuts $A_q(p_i[j])$, i.e., the $p_i[j]$-minimal min-cuts in G, as desired.

Appendix B. In this appendix we sketch how to efficiently carry out the "feasibility" tests in algorithm Ext.

The main thrust is as follows. Consider the level i problems. Let us write $S(i, B)$ for the optimal solution to $P_w(s^*, u^*, A(i) \otimes B)$ that is computed by Ext. Then, besides storing the solutions $S(i, B)$, we also compute their *compatibility*: For every pair B, B' of small graphs that we consider, we record where $S(i, B)$ is *feasible* for $P_w(s^*, u^*, A(i) \otimes B')$. We also compute similar compatibility data for problems of the form $P_w(s^*, u^*, D(i) \otimes B)$.

Suppose we have computed these data for the level i problems. Consider now one of the problems in step (a) of Ext, i.e., a problem $P_w(s^*, u^*, D(i) \otimes B)$. Here we have to test for each subset Z_i of the edges $\delta_G(V(A(i)), V(i+1))$ and each optimal solution $S(i, B')$ whether $S(i, B') \cup Z_i$ is feasible for $P_w(s^*, u^*, D(i) \otimes B)$. We do this as follows: Let $K(i)$ be the graph made up by the union of B and $\delta_G(V(A(i)), V(i+1))$. Then we compute the $K(i)$-consolidation of $D(i) \otimes B$ (with respect to Z_i and u^*, with a slight abuse of notation here: We really mean the *image* of $K(i)$ in $D(i) \otimes B$). Clearly, this graph is of the form $A(i) \otimes F$, for some small graph F, and therefore, using the compatibility data, we can look up the answer to the feasibility test. Moreover, this consolidation operation can be carried out in $O(2k2^{4k})$ time (a bounded number of tests for small min-cuts in a graph with a bounded number of edges). We also compute compatibility data for the optimal solutions to problems involving $D(i)$ in a similar way.

Finally, consider a level $(i + 1)$ problem, i.e., a problem $P_w(s^*, u^*, A(i + 1) \otimes B)$. In the feasibility tests here we consider pairs of the form S_i, J_i, where S_i is an optimal solution to a problem of the form $P_w(s^*, u^*, D(i) \otimes B')$, and J_i is an optimal solution to a problem of the form $P_w(u^*, v^*, J(i + 1) \otimes H)$ considered in step (b) of Ext. Here, as in the previous paragraph, we proceed with the feasibility test by first computing the B-feasibility of $A(i + 1) \otimes B$ with respect to $S_i \cup J_i$ and u^*. Having done so, once more the answer to the feasibility test is available from the compatibility data for the problems involving $D(i)$. Moreover, it is not hard to see that the consolidation operation can be carried out in time $O(2k2^{2k}m(i + 1) + k2^{4k})$, where $m(i + 1)$ denotes the number of edges in $J(i + 1)$: We perform, at most, 2^{2k} tests for small min-cuts in a graph with at most $m(i + 1) + k2^{4k}$ edges. The compatibility data for the level $(i + 1)$ problems is computed in a similar way.

Since the $J(i + 1)$ are pairwise edge-disjoint subgraphs of G, the overall complexity of the feasibility tests is thus $O(2k2^{2k}m)$, as desired.

REFERENCES

[BBM] D. Bienstock, E. F. Brickell, and C. L. Monma, *On the structure of minimum-weight k-connected spanning networks*, SIAM J. Discrete Math., 3 (1990), pp. 320–329.

[BD] D. Bienstock and N. Diaz, *A cutting-plane algorithm for blocking small cuts in a network*, in preparation.

[BLW] M. W. Bern, E. L. Lawler, and A. L. Wong, *Why certain subgraph computations require only linear time*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 117–125.

[C] W. H. Cunningham, *Optimal attack and reinforcement of a network*, J. Assoc. Comput. Mach., 32 (1985), pp. 549–561.

[F] A. FRANK, *Augmenting graphs to meet edge-connectivity requirements*, unpublished manuscript, 1990.

[GJ1] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[GJ2] ———, *Some simplified* NP-*complete graph problems*, Theoret. Comput. Sci., 1 (1979), pp. 237–267.

[GLS] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1980), pp. 169–197.

[GM] M. GRÖTSCHEL AND C. L. MONMA, *Integer polyhedra associated with certain network design problems with connectivity constraints*, SIAM J. Discrete Math., 3 (1990), pp. 502–523.

[HH] N. G. HALL AND D. S. HOCHBAUM, *A fast approximation algorithm for the multicovering problem*, Discrete Math., 15 (1986), pp. 41–54.

[MMP] C. L. MONMA, B. S. MUNSON, AND W. R. PULLEYBLANK, *Minimum-weight two-connected spanning networks*, Math. Programming, 46 (1990), pp. 153–171.

[S] P. D. SEYMOUR, *The matroids with the max-flow min-cut property*, J. Combin. Theory Ser. B, 23 (1977), pp. 189–222.

[T] R. E. TARJAN, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 500–526, June 1993

FLOOR-PLANNING BY GRAPH DUALIZATION: 2-CONCAVE RECTILINEAR MODULES*

KOK-HOO YEAP† AND MAJID SARRAFZADEH†

Abstract. Given a planar triangulated graph (PTG) G, the problem of constructing a floor-plan F such that G is the dual of F and the boundary of F is rectangular is studied. It is shown that if only zero-concave rectilinear modules (CRM) (or rectangular modules) and 1-CRM (i.e., L-shaped) are allowed, there are PTGs that do not admit any floor-plan. However, if 2-bend modules (e.g., T-shaped and Z-shaped) are also allowed, then every biconnected PTG admits a floor-plan. Thus, the employment of 2-bend modules is necessary and sufficient for graph dualization floor-planning. A linear-time algorithm for constructing a 2-CRM floor-plan of an arbitrary PTG is proposed.

Key words. floor-planning, planar graphs, graph dualization, CRM

AMS(MOS) subject classifications. 05C85, 68Q20, 68Q35, 68R10

1. Introduction.

Floor-planning is an early step in VLSI chip design where one decides the relative location of functional entities in a chip. The most immediate representation of a floor-plan is the partition of a rectangular chip area into modules (usually rectilinear polygons) where each module represents a functional entity. Such a partition can be represented by a partition graph where faces of the graph correspond to modules, edges represent the sides of modules, and vertices are junctions. Figure 1(a) shows a floor-plan represented in partition graph.

The dual graph of a partition is a graph where each vertex represents a module and each edge (i, j) represents adjacency of module i and module j. Given a partition, its dual graph is easily and uniquely determined. However, given a dual graph specifying adjacency requirements, it is not readily converted into a partition. In this formulation of floor-planning problem, we are given a graph specifying the connection requirements of circuit modules and wish to find a rectilinear partition on a rectangular chip area.

For simplicity, most floor-planning systems are restricted to 0-concave rectilinear modules (0-CRM), i.e., rectangles. The dual graph of a rectangular floor-plan is a planar triangulated graph (PTG). However, there exist planar triangulated graphs that do not have any dual floor-plans. Kozminski and Kinnen developed the necessary and sufficient conditions for the existence of a 0-CRM floor-plan [2]. They also showed a technique to transform one floor-plan to another where the adjacency requirements were preserved [3]. A linear time algorithm for constructing a rectangular floor-plan, if one exists, was reported in [1]. An algorithm to enumerate all rectangular floor-plans was reported in [13]. An algorithm for sliceable floor-planning was proposed in [14].

The construction of a floor-plan is complicated by the existence of complex triangles (CTs) (a cycle of three edges that is not a face) because the dual of a rectangular floor-plan does not contain any CT. One approach eliminates all CTs to obtain a rectangular floor-plan [12]. This technique introduces new vertices and edges in the original PTG, producing empty spaces in the floor-plan. If the empty spaces are considered as part of some adjacent modules, this approach produces general rectilinear-shaped modules. The weighted CT elimination problem has been shown to be NP-complete [11].

*Received by the editors February 11, 1991; accepted for publication (in revised form) March 24, 1992. This work was supported in part by the National Science Foundation under grant MIP-8921540.

†Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208.

Adjacency requirements of CTs can be achieved by introducing more complicated shapes instead of rectangular modules. A 1-concave rectilinear module (1-CRM) with 1-bend is required to satisfy the adjacency requirements of a CT. Similarly a 2-concave rectilinear module (2-CRM) can satisfy adjacency requirements of two CTs simultaneously. In some sense, the introduction of more complicated shapes is analogous to eliminating CTs. However, this approach maintains the adjacency requirements without arbitrarily adding new vertices and edges to the input PTG.

In this paper we examine 0-CRM (rectangle), 1-CRM, and 2-CRM. A 1-CRM refers to a rectilinear polygon that consists of six sides, five convex and one concave corners. Similarly, a 2-CRM refers to a rectilinear polygon with eight sides, six convex and two concave corners. A 2-CRM can be viewed as two adjacent 1-CRMs. In actual implementation, a 1- or 2-CRM can be represented by a set of adjacent rectangles.

The necessary and sufficient conditions for the existence of a 1-CRM floor-plan are given in [10]. However, there exist PTGs, as we will show, where 1-CRM floor-plan is not sufficient. Obviously, an arbitrary PTG admits a floor-plan if we allow general rectilinear-shaped modules, i.e., k-CRM for arbitrary large k. It was an open problem to find the least k in which some floor-plans exist for any PTG. We show that any biconnected PTG admits a 2-CRM floor-plan. A linear-time algorithm for constructing one is presented. The biconnected requirement is not a restrictive condition. By adding edges, a 1-connected planar graph can be transformed into a biconnected graph without loosing planarity and adjacency. We thus establish that employment of 2-bend modules is both necessary and sufficient.

The restriction of the dual graphs to PTG is a natural constraint. The dual of a floor-plan is planar since a floor-plan is planar by definition. Also, PTG represents the most dense planar graph. Any planar graph can be triangulated by adding edges. In the floor-planning process, an adjacency graph is first planarized and triangulated to yield a PTG. The PTG then serves as the input to the floor-planning problem.

This paper is organized as follows: Section 2 discusses the major issues arising in the floor-plan construction of a PTG, especially the local adjacency requirements. Section 3 presents a key theorem for the establishment of the sufficient conditions for dualization. An algorithm for floor-plan construction is proposed in §4.

2. Graph dualization. In this section, we discuss major issues arising in graph dualization of a PTG. We examine the local adjacency requirements of a PTG and show that a 1-CRM floor-plan is not sufficient. The notion of sharing, which is crucial to dual construction, is also discussed.

2.1. Floor-plan and its dual. A *floor-plan* F is a plane graph where

1. Each edge is either a horizontal or vertical line segment.
2. The boundary of F is rectangular.
3. Each vertex has either degree 2 or 3.

A degree 4 vertex can be represented by two vertices of degree 3 and an edge e_0 of zero length, as shown in Fig. 1(b). Thus we can assume that each vertex of F has degree 2 or 3 without loss of generality. Also we assume that there is no degree 2 vertex as shown in Fig. 1(c), because it can be represented by a single edge.

A face of a floor-plan is also called a *module*. An angle formed by two neighboring edges e_1 and e_2 incident to a common vertex v is called a *corner*. A corner is denoted by an ordered pair of edges (e_1, e_2) with e_1 preceding e_2 in a clockwise manner centered at v. There are three types of corners in F: *convex* ($90°$), *aligned* ($180°$), and *concave* ($270°$). Because modules are confined by horizontal an vertical line segments, the num-

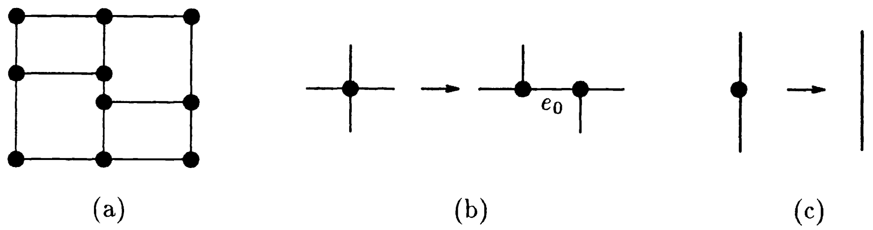


FIG. 1. *A floor-plan and its restrictions.*

ber of convex corners minus that of concave corners is always four. For the purpose of discussion, we always assume that F is confined by four infinite regions r, u, l, and b as shown in Fig. 2.

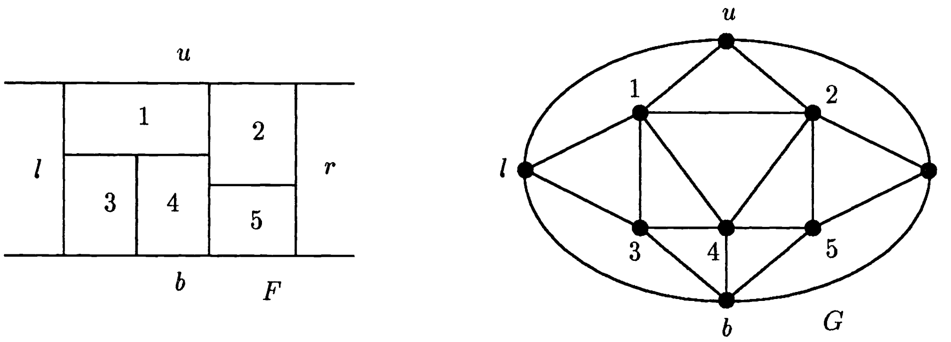


FIG. 2. *A floor-plan F and its dual G.*

The *dual* of a floor-plan F is a plane graph $G = (V, E)$, where each vertex in V corresponds to a face in F and each edge of G corresponds to an edge of F. A face bounded by three edges is also called a *triangle*. A face bounded by two parallel edges is called a *biface*. There is a one to one correspondence between the edges of F and that of G. Given F, the dual G is easily and uniquely determined. Figure 2 shows a floor-plan F and its dual G. If the vertices r, u, l, b (and incident edges) are deleted from G, the remaining subgraph G_c is called the *core* of G and G is called an *extended graph* of G_c. The vertices of G_c are called *core vertices*. Because corners are defined by two edges, there is also a one to one correspondence between corners of F and that of G. The bounded faces of F correspond to the core vertices of G.

In the graph dualization approach to floor-planning, we are given a core G_c and wish to find a floor-plan F where the dual G has core G_c. The very first step in the dualization process is to append the vertices r, u, l, b (and edges) to G_c to obtain G that matches the configuration shown in Fig. 3. This is not a trivial process. An excellent discussion on the details is contained in [2] and we shall not address this problem here. From now on, we will assume that G is given and conforms to the configuration as shown in Fig. 3. If F is restricted to 0-CRM modules, the dual G is a PTG. For a 1-CRM or 2-CRM floor-plan, the dual is also a PTG if we eliminate one parallel edge of each biface. Thus, it is reasonable to restrict our input G to PTGs. A PTG is also the most dense planar graph. Given a core G_c that specifies adjacency requirements of modules, we assume that some preprocessing has been performed to planarize and triangulate G_c to obtain G.

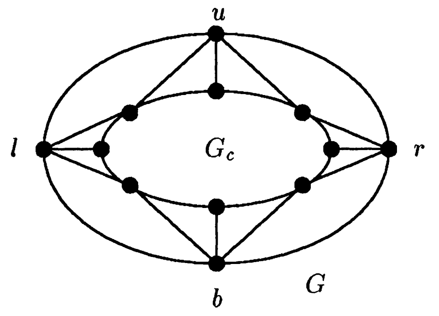


FIG. 3. *Configuration of a dual G.*

2.2. Alternate representation of a floor-plan with its dual. Let G be the dual of a floor-plan F. An *F-labeling* [3] of G is a labeling of edges and corners of G where

1. Dual edges of horizontal line segments are labeled H.
2. Dual edges of vertical line segments are labeled V.
3. Dual corners of G are labeled **convex, aligned,** or **concave.**

An edge label is also called *orientation*. The corners of the infinite face of G (e.g., $((l,u),(u,r))$) are not labeled because they are not interesting in our discussion. Figure 4 shows an example of F-labeling. Vertical edges are denoted by thick lines, whereas horizontal edges are shown with thin lines. A corner formed by two edges of identical orientations is an **aligned** corner. A corner formed by two edges of different orientations is **concave** if there is an arrow pointing to the vertex, otherwise it is **convex.**

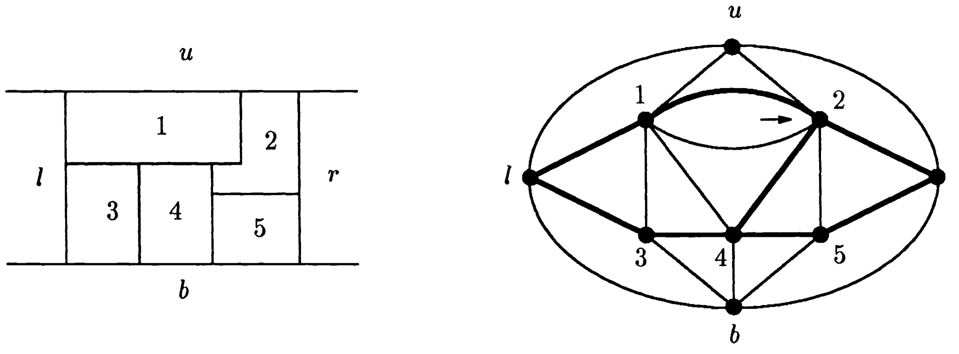


FIG. 4. *An example of F-labeling.*

An arbitrary F-labeling of G may not correspond to a floor-plan. We call an F-labeling *legal* if a corresponding floor-plan exists. There is a one-to-one correspondence between a legal F-labeling of G and a floor-plan F. Given G, the task of finding F can be viewed as finding a legal F-labeling in G.

Consider a legal F-labeling in G. By the definition of floor-plan, a legal F-labeling exhibits the following properties:

P1. All incident edges of u and b have horizontal orientation.

P2. All incident edges of l and r have vertical orientation except $(u,l),(u,r),(b,l)$, and (b,r).

P3. Let the convex, aligned, and concave corners be assigned cost 1, 0, and -1, respectively. The sum of all corner costs incident to a core vertex is exactly 4. The sum costs of u, b is 0, and l, r is 2. Also, the total corner cost of a triangle of G is 2, and the total cost of a biface is 0. This property is a consequence of rectilinear faces in F.

P4. Each triangle of G has at least one horizontal edge and at least one vertical edge. Each biface has exactly one horizontal and one vertical edge.

P5. No biface is incident to vertices r, u, l, b. All incident corners of r, u, l, b are not concave.

It is also evident that if an F-labeling of G satisfies the above conditions, it is a legal F-labeling. We can reconstruct the horizontal and vertical line segments of floor-plan F with the proper corners specified by the F-labeling. Thus a dualization process from G to F corresponds to finding an F-labeling satisfying the above conditions.

2.3. Complex triangles. A *complex triangle* (CT) is a nonface cycle containing exactly three edges. For our discussion, CTs are only defined on dual G. Figure 5 shows two examples of complex triangles.

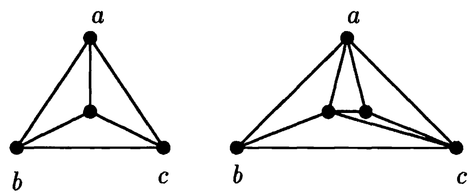


FIG. 5. *CTs.*

Consider the dualization of a CT $\triangle_{abc}$. If only 0-CRM is allowed in the floor-plan, it is impossible to satisfy the adjacency requirements of the edges (a, b), (b, c), and (a, c) simultaneously. However, if 1-CRM is allowed, a floor-plan exists as shown in Fig. 6. A 0-CRM floor-plan does not exist if and only if G contains a CT [2]. Note that the concave corner of module 3 contributes to the existence of a floor-plan. A biface is formed in the dual due to the concave corner of vertex 3. For clarity, some edges of the floor-plans are not shown in the duals.

As far as the CT is concerned, any of the vertices $1, 2, 3$ may be chosen to bear a concave corner so that the adjacency requirements are satisfied. We call such choice the *assignment* of a CT to a vertex. Vertices 1 and 2 are also suitable for assignment.

Intuitively, the assignment of a CT to a vertex v can be seen as adding a concave corner to module v, thus bending it to satisfy the adjacency of the CT. To construct a floor-plan, all CTs in G must be assigned. The number of assignments on a vertex is related to the number of concave corners in the final floor-plan. Since we only allow 2-CRM, the number of assignments on any vertex shall not exceed 2.

A *perfect assignment* of a dual G is a set of assignments of all CTs of G where

1. Every CT is assigned to a core vertex.
2. No vertex carries more than two assignments.
3. Noncore vertices of G (i.e., r, u, l, and b) are not assigned.

In a perfect assignment, a core vertex may be assigned zero, one, or two times.

In our algorithm, if a vertex is not assigned, its corresponding module in F is a rectangle. If a vertex is assigned once, it is a 1-CRM. There is only one type of 1-CRM if we disregard module orientations. A 1-CRM is also called an L-shaped module. If a vertex is assigned twice, a 2-CRM is formed. There are four classes of 2-CRM as shown in Fig. 7.

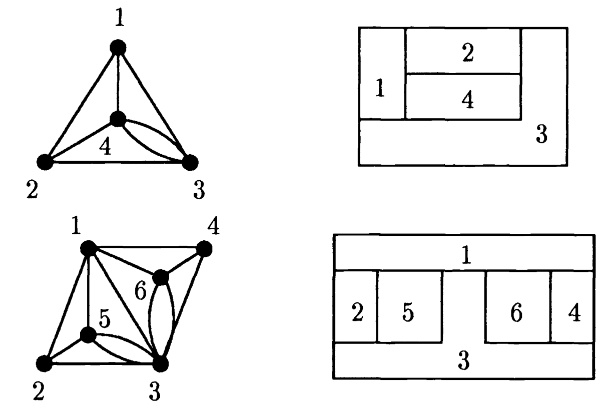


FIG. 6. *Floor-plans of CTs.*

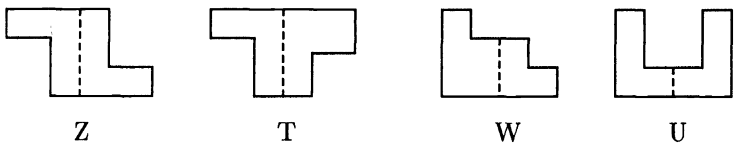


FIG. 7. *Four classes of 2-CRM: Z, T, W, and U.*

2.4. Insufficiency of 1-CRM duals. Given a PTG G, some vertices of a CT must be at least 1-CRM to satisfy the local adjacency requirements. Kozminski and Kinnen proved the necessary and sufficient conditions for a graph to admit a 0-CRM (rectangular) floor-plan [2]. It was further generalized by [10] to prove the necessary and sufficient conditions for the existence of a 1-CRM floor-plan. The primary condition for the existence of a 1-CRM floor-plan is that all nonoverlapping (i.e., one is not contained in the other) CTs can be assigned to some vertices and each vertex is not assigned more than once. If a vertex is assigned, it becomes a 1-CRM in the corresponding floor-plan; otherwise it is a rectangle. However, there exist PTGs that do not admit 1-CRM floor-plans. In particular, consider the graph shown in Fig. 8: The graph contains seven nonoverlapping CTs and only six vertices are available for assignment. There is no assignment that can satisfy the necessary conditions for the existence of a 1-CRM floor-plan. This suggests that the dualization of a PTG requires more complicated shapes. In this paper, we prove that generalization to 2-CRM is necessary and sufficient for the existence of a floor-plan for any biconnected PTG.

2.5. Complex triangle graph and containment tree. A *complex triangle graph* (CTG) of $G = (V, E)$ is $CTG(G) = (V_c, E_c)$, $V_c \subseteq V$ and $E_c \subseteq E$ where

$$V_c = \{\, v \mid v \text{ belongs to some CTs of } G \,\},$$
$$E_c = \{\, e \mid e \text{ belongs to some CTs of } G \,\}.$$

Given G, it is easy to construct the CTG thereof. A CT is always assigned to a vertex in V_c. Therefore, it suffices to consider $CTG(G)$ in search of a perfect assignment. In general, $CTG(G)$ may not be connected. However, it is easy for our assignment algorithm to be

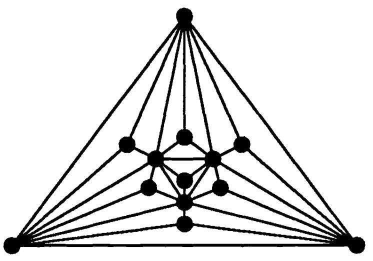


FIG. 8. *A PTG that does not admit any 1-CRM floor-plan.*

repeated for each connected component of $CTG(G)$. Therefore, we will assume that $CTG(G)$ is connected.

There is a natural hierarchy among the CTs of G. Formally, we say that CT $\triangle_{abc}$ *contains* CT $\triangle_{def}$ if the vertices d, e, and f lie in the region bounded by the edges (a,b), (a,c), and (b,c). The area of $\triangle_{def}$ must be strictly less than that of $\triangle_{abc}$. We say that $\triangle_{abc}$ *immediately contains* $\triangle_{def}$ if there exist no triangles contained in $\triangle_{abc}$, which contain $\triangle_{def}$. If two complex triangles have no containment relation, we say that they are *independent*. Figure 9 illustrates the containment relationship.

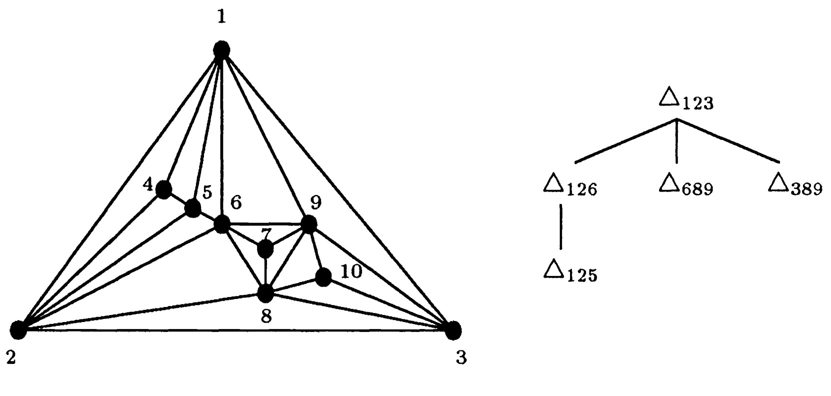


FIG. 9. *A PTG and its corresponding containment tree.*

Given G, we can construct the *containment tree* representing the hierarchy of the CTs. Each tree node represents a CT, and a parent node immediately contains its child nodes. A parent and a child can share at most two vertices. A pseudo-root node is added as the parent of top-level CTs, if needed. Traversing the tree top-down allows us to process all CTs hierarchically.

2.6. Sharing assignments. Consider two CTs $\triangle_{abc}$ and $\triangle_{ade}$ where $\triangle_{abc}$ immediately contains $\triangle_{ade}$. Suppose $\triangle_{abc}$ is assigned to vertex a. To satisfy the adjacency requirements of $\triangle_{ade}$, we can naturally assign it to vertex a to reduce the number of assigned vertices (see Fig. 10(b), where $\triangle_{abc} = \triangle_{123}$, $\triangle_{ade} = \triangle_{124}$, $a = 1$). Such assignment of the child $\triangle_{ade}$ is called *sharing assignment* because $\triangle_{ade}$ shares the same concave corner of a to satisfy its adjacency requirements. Certainly, $\triangle_{ade}$ has the freedom of being assigned to d or e, but the final floor-plan will have more nonrectangular modules, an undesirable feature (see Fig. 10(c), modules 1 and 4).

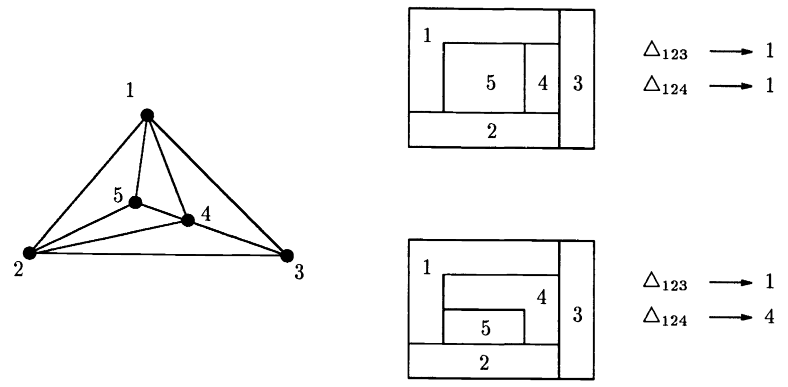

1
5
4
2
3
1
5
4
3
2
$\triangle_{123} \longrightarrow 1$
$\triangle_{124} \longrightarrow 1$
1
4
5
3
2
$\triangle_{123} \longrightarrow 1$
$\triangle_{124} \longrightarrow 4$

FIG. 10. *Sharing assignment:* (a) *adjacency graph,* (b) $\triangle_{123}$ *and* $\triangle_{124}$ *share vertex* 1; *and* (c) $\triangle_{123}$ *and* $\triangle_{124}$ *have independent assignments.*

When the child triangle $\triangle_{ade}$ shares an assignment with its parent, it does not add more bend to module a. Therefore, we do not increase the assignment count on vertex a when sharing occurs. All descendants of a CT can share its assignment as long as they have a common vertex. We will later see that sharing is necessary for the existence of 2-CRM floor-plans.

3. 2-CRM floor-plans. In this section, we first develop the properties of dual decomposition under a cut (to be defined). We then show that given a graph G with perfect assignment, we can construct a floor-plan where modules are 0-CRM, 1-CRM, or 2-CRM. The floor-plan construction is based on top-down recursive decomposition of G and a case analysis. The problem of floor-planning is thus reduced to finding a perfect assignment on G.

A *path* is an ordered set of vertices $(v_1, \dots, v_n)$ in $G = (V, E)$, where $v_i \in V$, $i = 1, \dots, n$, and $(v_i, v_{i+1}) \in E$, $i = 1, \dots, n-1$. $v_1, \dots, v_n$ are distinct. We consider the decomposition of G into two subgraphs G_1 and G_2. Let $P = (v_1 = u, \dots, v_n = b)$ be a path in G that begins on vertex u and ends on vertex b. The path P *decomposes* G into left and right subgraphs G_l and G_r as shown in Fig. 11(a). Note that the vertices of P are duplicated in both subgraphs and vertices r' and l' are appended to G_l and G_r, respectively. G_l and G_r both conform to the configuration of Fig. 3. For a nontrivial decomposition, the path should be chosen such that G_l and G_r each contains at least one core vertex (of G) not in P. An edge (v_i, v_j) where $\mid i - j \mid \geq 2$ is called a *chord* (with respect to P). The existence of a chord will create a new CT in a subgraph when G is decomposed along P. If P does not contain chords and generates nontrivial decomposition of G to left and right subgraphs, we call P a *vertical cut*. A *horizontal cut* is defined similarly where upper and lower subgraphs are decomposed. A *cut* is either a vertical or horizontal cut.

In our algorithm, a floor-plan is constructed by recursive subdivision based on cuts that decompose G into G_1 and G_2. Such a decomposition has the properties stated in Lemma 1.

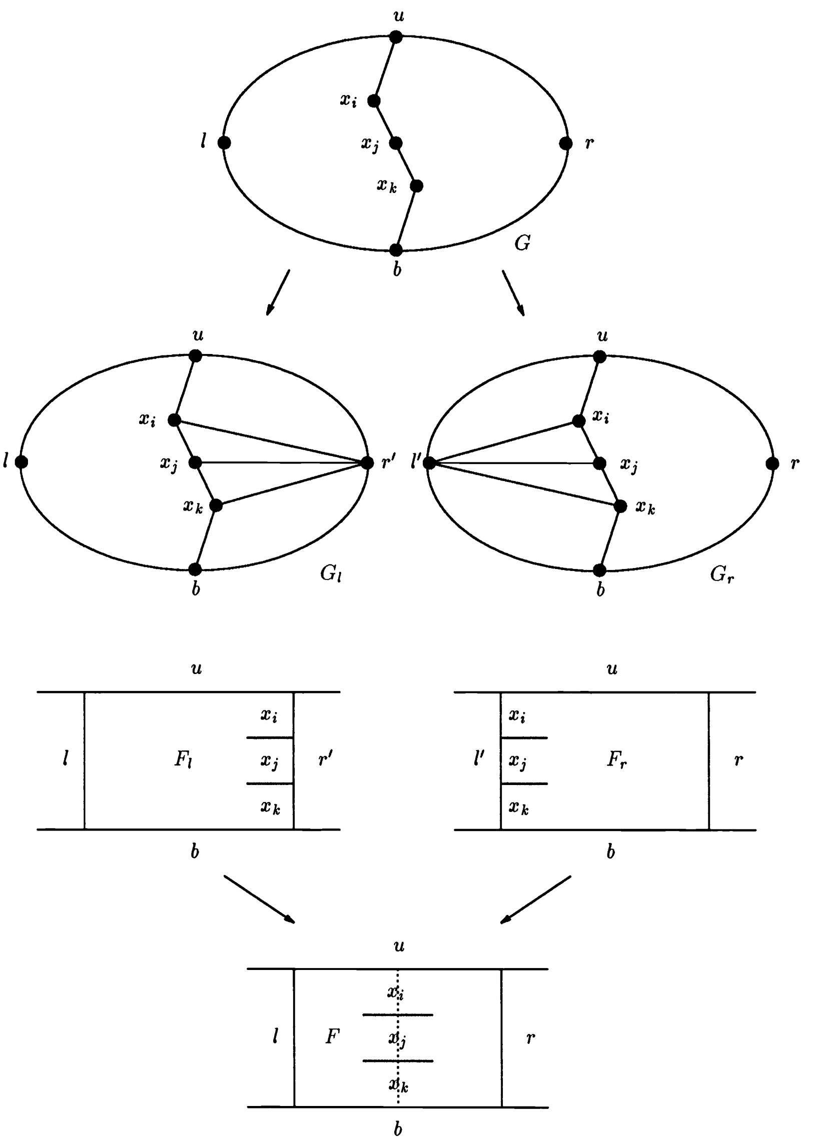


FIG. 11. *Decomposition by a cut:* (a) *decomposition of G to G_l and G_r*; *and* (b) *merging floor-plans of G_l and G_r to obtain the floor-plan of G.*

LEMMA 1. *Let $P^* = (u = v_1, \ldots, v_n = b)$ be a vertical cut in G where no edges (chords) (v_i, v_j), $| i - j | \geq 2$ exist. Let G_l and G_r be two subgraphs decomposed by P^*, then*

1. $CT(G) = CT(G_l) \cup CT(G_r)$ *and*
2. $CT(G_l) \cap CT(G_r) = \phi$,

where $CT(G)$ denotes the set of CTs of G.

Proof. There is no chord among vertices of P^*. Thus the introduction of vertices r' and l' will not create new CTs since r' and l' are only adjacent to vertices of P^*.

1. Since $G_l - r'$ and $G_r - l'$ are subgraphs of G, $CT(G) \supseteq CT(G_l) \cup CT(G_l)$ trivially holds. To show that $CT(G) \subseteq CT(G_l) \cup CT(G_r)$, we consider the converse: If it is true, there must be some CT $\triangle_{abc} \notin CT(G_l) \cup CT(G_l)$ but $\triangle_{abc} \in CT(G)$. The edges (a, b), (b, c), and (c, a) cannot appear simultaneously in G_l or G_r. Assume, without loss of generality, that (a, b) is in G_l and (b, c), (c, a) are in G_r. Since a, b appears in both subgraphs, $a, b \in P^*$. Let $a = v_i$ and $b = v_j$ for some integer i, j. If $\mid i - j \mid = 1$, $\triangle_{abc}$ will be in G_r, a contradiction. Thus $\mid i - j \mid \geq 2$ since a and b are distinct. But this would imply that (a, b) is a chord, another contradiction.

2. If $CT(G_l) \cap CT(G_r)$ is not empty, let $\triangle_{abc}$ be a CT of $CT(G_l) \cap CT(G_r)$. Since the common vertices of G_l and G_r are only the vertices of P^*, a, b, and c must be in P^*. Because P^* contains no chords, edges (a, b), (b, c), and (c, a) cannot form a CT. We reach a contradiction. □

A similar lemma applies to horizontal cuts. Intuitively, Lemma 1 says that the set of CTs in G can be decomposed into two disjoint sets by a cut. Such a decomposition preserves the "identity" of each CT in G, i.e., a complex triangle must be found in either G_1 or G_2, exclusively. Furthermore, no new CT is introduced by the decomposition nor does it destroy existing CTs. If (a, b) is an edge of some CT and $a = v_i$, $b = v_j$ are vertices of the cut, $\mid i - j \mid = 1$ must hold true. The lemma also suggests a method for recursive construction of floor-plans, which leads to our main result.

THEOREM 1. *If G has a perfect assignment, a 2-CRM floor-plan F exists with dual G_a and G is obtained from G_a by eliminating one edge from each biface of G_a.*

Proof. We prove the theorem by constructing an *augmented graph* G_a from the perfect assignment of G. By induction on the number of vertices of G_a, we generate a legal F-labeling on G_a. A 2-CRM floor-plan can thus be constructed.

For each assignment of CT $\triangle_{abc}$ to a, we chose an edge (a, x), $x \notin \{b, c\}$ in the CT and duplicate (a, x) to create a biface with an arrow pointing to vertex a. The resulting graph is G_a. The procedure is demonstrated in Fig. 12. There is a one-to-one correspondence between an assignment and a biface.

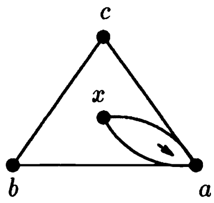


FIG. 12. *Transforming G to augmented graph G_a.*

We now show that there exists a legal F-labeling of G_a, thus implying the existence of a 2-CRM floor-plan.

Induction Basis: This is trivially true for G_a with five vertices.

Induction Hypothesis: There exists a legal F-labeling of G_a with no more than n vertices.

Induction Step: The initial labeling procedure is as follows:

1. All incident edges of u and b are labeled H.
2. All other unlabeled edges incident to l and r are labeled V.
3. If (x, y) is an edge where (u, x, y) or (b, x, y) is a face, label (x, y) as V.
4. If (x, y) is an edge where (l, x, y) or (r, x, y) is a face, label (x, y) as H.

In the labeling process, we only label the edges. The corners are implicitly labeled when their edges are labeled. The implicit rules for corner labeling are as follows:

1. If the two edges of a corner have identical labels, the corner is labeled **aligned** (180°).

2. If the two edges of a corner have distinct labels and there is an arrow pointing to the corner, the corner is labeled **concave** (270°); otherwise the corner is labeled **convex** (90°).

We first try to find a vertical cut P in G. The procedure is rather straightforward:

1. Path search. Find a path $P = (u = v_1, \dots, v_n = b)$ where there is at least one core vertex on the left and right of P.

2. Chord elimination. If a chord (v_i, v_j), $(i < j)$ exists in P, remove vertices v_k from P, where $i < k < j$. Repeat this procedure until all chords in P are eliminated.

If the resulting path P^* meets the criteria of a vertical cut, we decompose the graph as shown in Fig. 11. Vertices of P^* are duplicated and the vertices r', l' and corresponding edges are added. When the vertices of P^* are duplicated, the assignments are carried along with corresponding CTs that contributed to the assignments. Thus if $\triangle$ was assigned to v and $\triangle$ appears on the left subgraph after decomposition, vertex v on the left subgraph will carry the assignment of $\triangle$. The duplicated vertex v on the right subgraph will not carry the assignment of $\triangle$. However, the vertex v on the right subgraph may carry assignment of other CTs. It can be easily seen that if G has a perfect assignment, the subgraphs also have perfect assignments. Thus the induction hypothesis applies since each subgraph contains one less vertex. A similar procedure is applied to a horizontal cut.

By Lemma 1, CTs are not decomposed by a cut. Thus an edge of a cut cannot be an edge of a biface because a biface only exists in a CT. The decomposition process is depicted in Fig. 11. The resulting floor-plan F is constructed by merging the left and right floor-plans F_l and F_r as shown. Modules (of F_l and F_r) corresponding to vertices in the vertical cut P^* are coalesced when F_l and F_r are merged.

If vertical or horizontal cuts do not exist, we proceed with the following case analysis on G_a:

Case A. At least one of the noncore vertices has degree 3.

Case B. All noncore vertices have degree at least 4.

CASE A. *At least one of the noncore vertices has degree* 3.

Without loss of generality, we assume r has degree 3. The configuration of G_a with its initial labeling is shown in Fig. 13. Consider the distinguished vertex v adjacent to r. Vertex v has at least three edges (v, r), (v, u), (v, b). There are three possibilities for v:

A.1. v is not assigned.

A.2. v is assigned once.

A.3. v is assigned twice.

CASE A.1. *v is not assigned.*

We delete vertex r and its associated edges. We then relabel vertex v as r'. There is no biface incident to v since v is not assigned. All unlabeled incident edges of r' are labeled V except (u, r') and (b, r'), which have been labeled H. The resulting graph has one less vertex and thus the induction hypothesis applies. The F-labeling and corresponding floor-plans are demonstrated in Fig. 14. As before, thick edges have vertical orientation and thin edges have horizontal orientation. The orientations of dotted edges are not determined yet. One can easily verify that the F-labeling of vertex v does not violate the properties stated in §2.2.

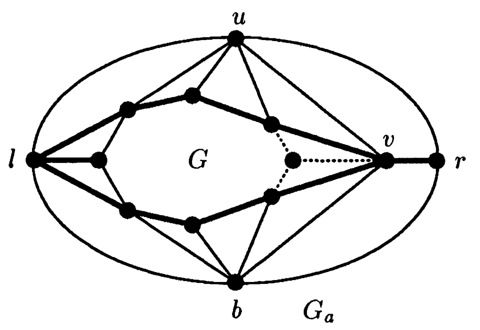


FIG. 13. *Graph of Case* A.

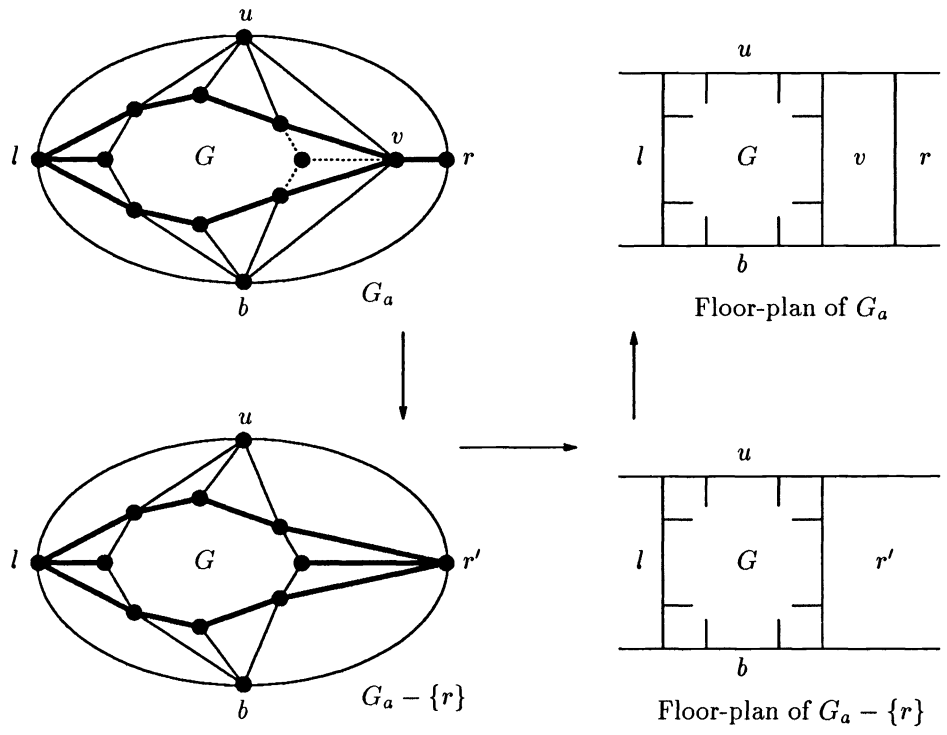


FIG. 14. *Floor-plans and duals in Case* A.1.

CASE A.2. *v is assigned once.*

Let (x_s, v) be the biface of the assignment of v. Refer to the configuration of G_a in Fig. 13. Let $P = \{u = x_1, \ldots, x_s, \ldots, x_p = b\}$ be the path from u to b in counterclockwise order where all vertices of P are adjacent to v. The general configuration of the graph is shown in Fig. 15.

The biface represents a concave corner of module v. Let $\{e_1 = (u, v), \ldots, e_s = (x_s, v), e_{s+1} = (x_s, v), \ldots, e_{p+1} = (b, v)\}$ be the set of edges in counterclockwise order with (e_s, e_{s+1}) as the biface. We claim that the other edges cannot form bifaces. Suppose

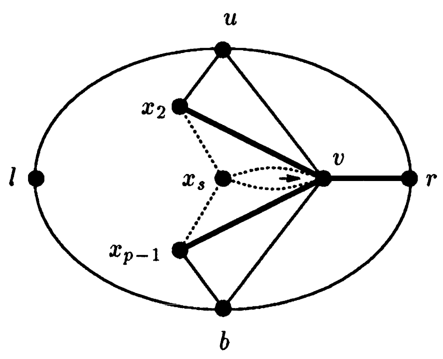


FIG. 15. *Graph of Case* A.2.

for contradiction that e_m, e_{m+1} is also a biface. If the convex corner of the biface is incident to v, v will have an assignment count of more than one. If the convex corner is incident to some other vertex, v must lie strictly inside some CT. From the configuration of Fig. 15, it is impossible for v to lie in any CT.

We perform a local transformation to the graph G_a so that the resulting graph has one less vertex. The transformation procedure is illustrated in Fig. 16. The initial labeling of G_a is shown on Fig. 16(a). We delete vertices r, b and their incident edges. We then split vertex v into two vertices and label them r' and b'. (See Fig. 16(c).) Edges $\{e_1, \ldots, e_s\}$ are now incident to r' and $\{e_{s+1}, \ldots, e_p\}$ are incident to b'. Edges (l, b') and (r', b') are also added. The unlabeled edges $\{e_2, \ldots, e_s\}$ are labeled V, and $\{e_{s+1}, \ldots, e_p\}$ are labeled H, as shown in Fig. 16(d).

Let G'_a be the resulting graph after the transformation. All CTs assigned to v, along with their bifaces, are disintegrated by the transformation process. From the floor-plan of Fig. 16(e), we see that the concave corner (biface) is eliminated by the introduction of edge (r', b'). The merging of the floor-plan of G' with 1-CRM module of v, is illustrated in Fig. 16(b). A degenerate case where $x_{p-1} = l$ is illustrated in Fig. 17. We can verify that the labeling of vertex v and P conforms to the properties stated in §2.2.

CASE A.3. *v is assigned twice.*

Refer to the general configuration of Fig. 13 again. Since v is assigned twice, there must be two bifaces incident to v. Let (x_{s1}, v) and (x_{s2}, v) be the bifaces. As before, we consider the vertices adjacent to v. Let $P = \{u = x_1, \ldots, x_{s1}, \ldots, x_{s2}, \ldots, x_p = b\}$ be the path from u to b in counterclockwise order where all vertices of P are adjacent to v. Also let $\{e_1, \ldots, e_{s1}, e_{s1+1}, \ldots, e_{s2+1}, e_{s2+2}, \ldots, e_{p+2}\}$ be the incident edges with bifaces (e_{s1}, e_{s1+1}) and (e_{s2+1}, e_{s2+2}). Since x_{s1} and x_{s2} are located in two independent CTs, $s2 \geq s1 + 2$ must hold true. Thus there is a vertex $x_s (s1 < s < s2)$ not located in any of the two CTs.

We claim that there are no vertices to the left of P. Otherwise, we can apply the chord elimination procedure to P to obtain the vertical cut P^* since vertices of P are adjacent to v. Because of the absence of such vertices, vertex x_{p-1} must be adjacent to l (by triangulation) or $x_{p-1} = l$. (If not, we would have a vertex z with edge (z, b) on the left of (x_{p-1}, b), where z is on the left of P.) We consider the general case where $x_{p-1} \neq l$ (Fig. 18) and treat the case where $x_{p-1} = l$ as a degenerate case (Fig. 19).

Consider the path $Q = (r, v, x_s, x_{s+1}, \ldots, x_{s2}, \ldots, x_{p-1}, l)$. Applying the chord elimination procedure (as described in the beginning of this proof) to the subpath Q'

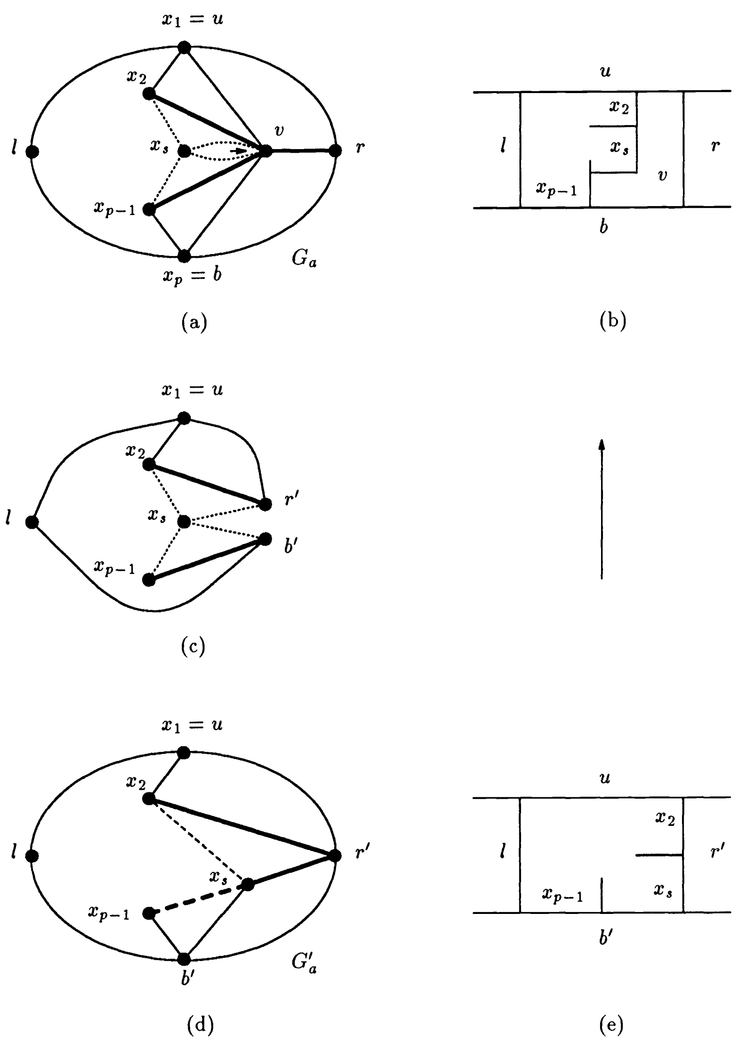


FIG. 16. *Transformation of G_a in Case* A.2.1.

$= (x_s, x_{s+1}, \ldots, x_{s2}, \ldots, x_{p-1}, l)$, we obtain a path $Q^* = (y_1 = r, y_2 = v, y_3 = x_s, \ldots, y_{q-1} = x_t, y_q = l)$. We have assumed that no horizontal cut existed in our case analysis implying Q^* is not a cut. However, all chords of Q^* must be incident to vertex v; otherwise they would have been eliminated by the chord elimination of Q' when constructing Q^*. Thus, the only condition that disqualify Q^* as a cut are the chords (y_j, v) for some j.

Despite the fact that Q^* is not a cut, we will decompose G_a along the horizontal path Q^* and give special treatment to the chords (y_j, v). Let the upper and lower subgraphs be G_u and G_b. The decomposition process is depicted in Fig. 18. Edges (y_i, b') are appended to G_u and edges (y_i, u') are appended to G_b, for $i = 1, \ldots, q$. Because of the chords (y_j, v), the addition of edges (y_j, u') in G_b results in new CTs $\triangle_{y_j v u'}$, which may not exist in G_a. However, from the figures, we can assign the concave corner of these

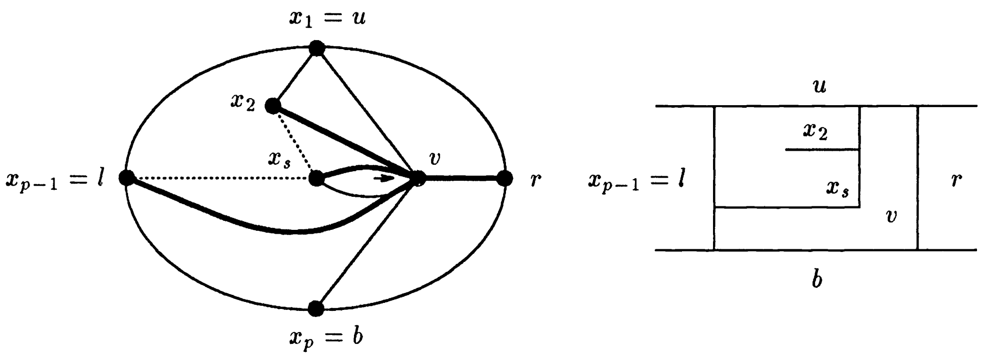


FIG. 17. *A degenerate graph of Case* A.2.

CTs to v. The assignment is a sharing assignment (see §2.6) using the concave corner of the biface (x_{s2}, v). Other CTs of G_a are handled as if Q^* is a cut. We can verify that the vertices along Q^* do not violate the properties given in §2.2.

Note that in some cases we may have $x_t = x_{p-1}$ or $x_t = x_s$ (see Fig. 18 for x_t), however, x_s and x_{p-1} are distinct by our construction. A degenerate example where $x_{p-1} = l$ is illustrated in Fig. 19.

CASE B. *All noncore vertices have degree at least* 4.

This is impossible as we shall show that we can construct a cut on G_a. The general configuration of the graph is shown in Fig. 20. Note that, there are at least four more distinct vertices v_1, v_2, v_3, v_4 in G_a as shown in the figure. We consider two possibilities:

B.1. Some of the edges (v_3, u), (v_3, r), and (v_1, b) exist.

B.2. None of the edges (v_3, u), (v_3, r), and (v_1, b) exist.

CASE B.1. *Some of the edges* (v_3, u), (v_3, r), *and* (v_1, b) *exist.*

Since the edges are topologically identical, we assume that (v_3, r) exists. The path (l, v_3, r) is a horizontal cut. Thus we have a contradiction.

CASE B.2. *None of the edges* (v_3, u), (v_3, r), *and* (v_1, b) *exists.*

We scan adjacent vertices of v_3 clockwise to obtain the path $(l, x_1, \ldots, x_m, b)$. Similarly, we scan adjacent vertices of b clockwise to obtain the path $(v_3, y_1, \ldots, y_n, r)$. Note that $x_m = y_1$ and $y_n = v_2$. We construct the path $P = (l, x_1, \ldots, x_m = y_1, \ldots, y_n, r)$ (see Fig. 21). Vertex u cannot appear in P due to the nonexistence of (v_3, u). Also, v_1 cannot appear in P, otherwise we have a horizontal cut (l, v_3, v_1, r). We apply the chord elimination procedure to P and obtain a horizontal cut P^*. This is another contradiction. □

From Theorem 1, the problem of the existence of a 2-CRM floor-plan is reduced to finding a perfect assignment in G. For a given G, there may be more than one perfect assignment. Different assignments correspond to different floor-plans. In the next section, we present an algorithm to obtain a perfect assignment in G.

4. An algorithm for perfect assignment.

In this section, we first demonstrate that assignment sharing is necessary for the existence of a 2-CRM floor-plan. We then propose a hierarchical assignment algorithm to obtain a perfect assignment in a PTG. The algorithm can be easily adapted to the extended dual G, as described in §4.3. Finally, we present an example.

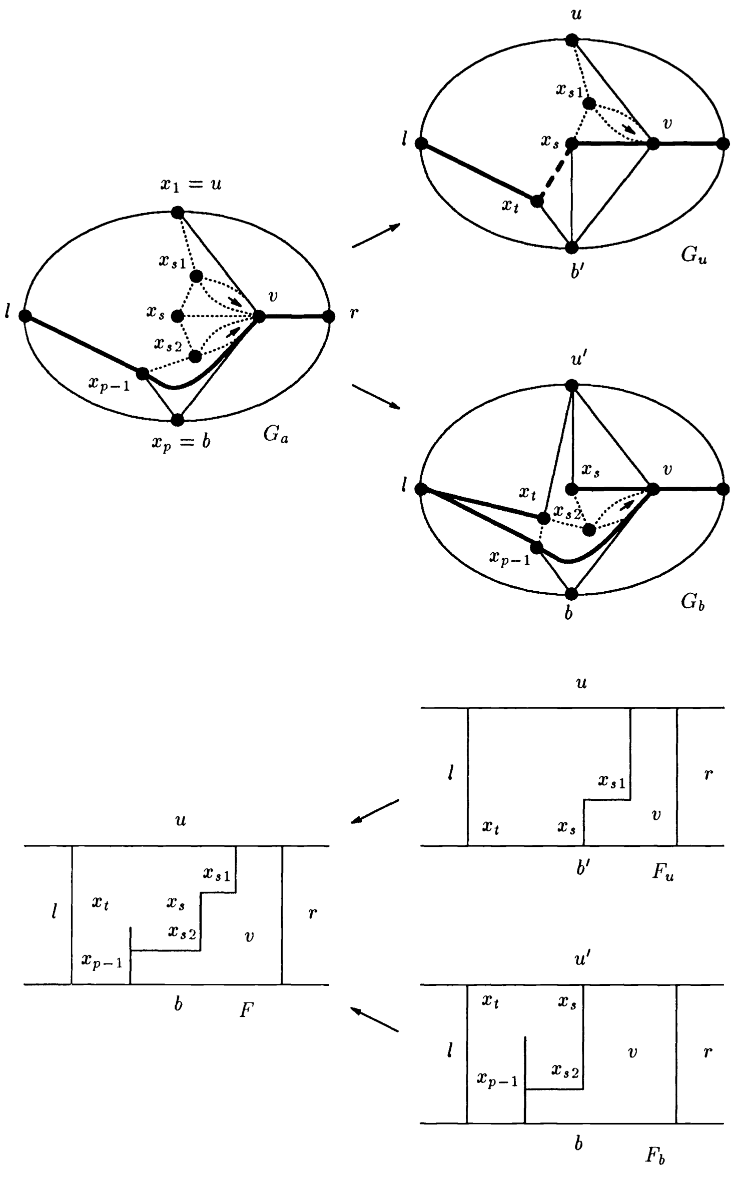


FIG. 18. *Dual decomposition of Case* A.3.

4.1. Necessity of sharing assignments. Sharing assignment refers to the assignment of a child CT that uses the concave corner of its parent CT to satisfy its adjacency requirements. The notion of sharing has been discussed in §2.6, and Fig. 10 illustrates an example of a sharing assignment. An assignment can be shared by more than one generation of CTs. Sharing reduces the number of nonrectangular modules in a floor-plan. In some graphs, sharing is even necessary to achieve perfect assignments.

Consider a series of PTGs G_i as shown in Fig. 22. G_n is obtained by adding a vertex and three edges to each bounded face of G_{n-1}. The containment tree of G_n is a full

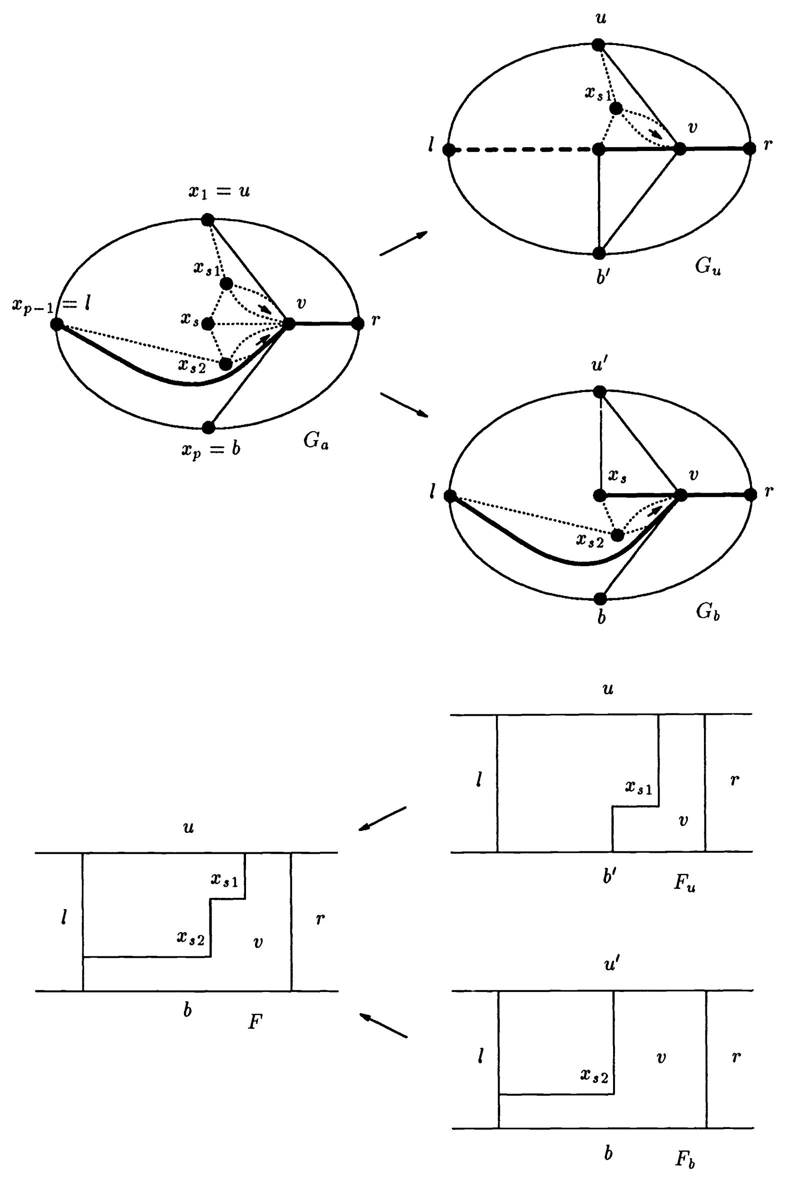


FIG. 19. *A degenerate example of Case* A.3.

ternary tree. Let $f(n)$ denote the number of bounded faces in G_n and $CT(G_n)$ denote the number of CTs in G_n. From G_n to G_{n+1}, each bounded face of G_n becomes a CT of G_{n+1}. Because $f(n) = 3^n$, we have

$$CT(G_{n+1}) = CT(G_n) + 3^n, \qquad n \geq 0,$$
$$CT(G_0) = 0.$$

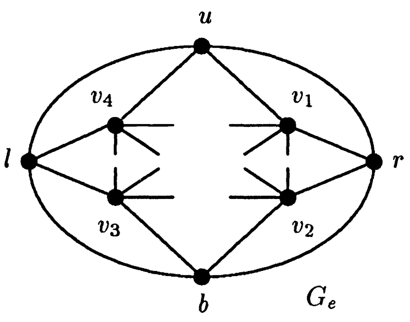


FIG. 20. *Graph of Case* B.

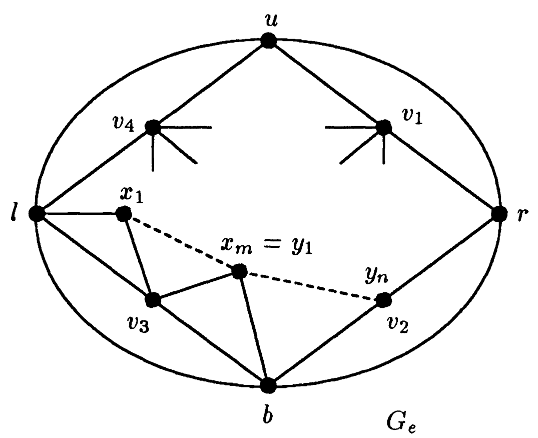


FIG. 21. *Graph of Case* B.2.

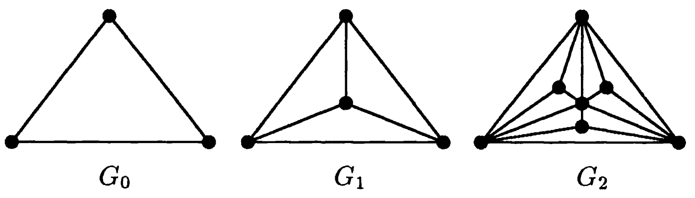


FIG. 22. *Family of graphs to show the necessity of sharing assignment.*

Solving the recurrence equations gives

$$CT(G_n) = (3^n - 1)/2,$$

which is also the number of nodes in the containment tree of G_n. Let $V(G_n)$ denote the number of vertices available for assignment in G_n (this number is not equal to the number of vertices in G_n). For $n \geq 1$, we have

$$V(G_{n+1}) = V(G_n) + 3^{n-1}, \qquad n \geq 1,$$
$$V(G_1) = 3.$$

Solving the recurrence equations gives

$$V(G_n) = (3^{n-1} + 5)/2.$$

Since each vertex can carry at most two assignments, the maximum number of assignments $A(G_n)$ in G_n is

$$A(G_n) = 2V(G_n) = 3^{n-1} + 5.$$

For $n \geq 4$, $CT(G_n) > A(G_n)$. If each assignment of a CT is counted once, it is impossible to obtain a perfect assignment since the number of CTs outnumber that of available assignments. This is also intuitively evident from the fact that each new vertex introduces three CTs but only two new assignments.

We thus see that sharing is necessary to achieve a perfect assignment. When a CT shares its assignment with its parent, we do not increase the assignment count on the vertex because sharing does not result in more complicated shapes. The assignment of a vertex is counted twice only if it is assigned by two independent CTs (i.e., one does not contain the other and vice versa). From the practical point of view, sharing should be encouraged whenever possible since we prefer simpler shaped modules.

4.2. Hierarchical assignment algorithm for PTG. The hierarchy of CTs suggests a recursive assignment procedure. We proceed in breadth-first manner in the containment tree. At each step of the recursion, we only consider the CTs that have identical parent. Let G_i be the graph induced by the CTs in step i, i.e., the CTG of siblings having identical parent $\triangle$. We also call $\triangle$ the parent of G_i. We assume that G_i is connected. If not, our algorithm will be applied to each connected component of G_i. After G_i is assigned, the algorithm is recursively applied to the children of each CT of G_i in the containment tree.

We define the following terms on G_i: An edge (a, b) is called a *boundary edge* if it is an edge of the infinite face of G_i; otherwise it is called an *internal edge*. A vertex is called a *boundary vertex* if there are some boundary edges incident to it; otherwise, it is called an *internal vertex*. From the definition, the following properties hold on graph G_i:

P1. A boundary edge has exactly one triangle on one side.

P2. A boundary vertex has at least two boundary edges incident to it.

P3. A boundary vertex is adjacent to at least two other boundary vertices.

We call a vertex *saturated* if it has already been assigned twice. Consider vertices of G_i. Let $\triangle_{abc}$ be the the parent of G_i. Vertices a, b, c may appear in G_i. In the worst case, all of them may be saturated due to previous recursive assignment steps. Suppose $\triangle_{abc}$ is assigned to vertex a in the previous recursive step. Because of the sharing scheme, vertex a is allowed to carry one more assignment in G_i. Therefore, even if the vertex is already saturated from its previous assignments, we can *force* (artificially set the assignment count) it to take one more assignment in G_i. Thus at most two vertices (b and c) are saturated in G_i, and both of them are boundary vertices. In fact, the vertex a is also allowed to carry assignments of any descendant of $\triangle_{abc}$ as long as the assignments are shared.

As the algorithm proceeds, we delete edges and vertices of G_i that do not have unassigned triangles incident to them. We always maintain the following invariant properties:

Q1. At most two boundary vertices are saturated. The other boundary vertices may have zero or one assignment.

Q2. Internal vertices have zero assignment.

We choose an unsaturated vertex v from the set of boundary vertices. Such a vertex always exists since a nontrivial G_i has at least three boundary vertices, of which at most two are saturated. We scan the vertices adjacent to v counterclockwise. Let $v_0, \dots, v_n$, $n \geq 1$ be the vertices, where (v, v_0) and (v, v_n) are boundary edges. (Note that v_0 and v_n are boundary vertices.) There are at most n triangles $\triangle_1, \dots, \triangle_n$ incident to v, where $\triangle_i$ is $\triangle_{v_{i-1}\, v_i\, v}$. There are two possible cases:

Case 1. $v_1, \dots, v_{n-1}$ are all internal vertices.

Case 2. Some of $v_1, \dots, v_{n-1}$ are boundary vertices.

CASE 1. $v_1, \dots, v_{n-1}$ *are all internal vertices.*

If $v_1, \dots, v_{n-1}$ are all internal vertices, we will assign $\triangle_i$ to v_i for $i = 1, \dots, n-1$ and assign $\triangle_n$ to v. If a $\triangle_i$ does not exist, we simply ignore its assignment. We delete vertex v and edges $(v, v_0), \dots, (v, v_n)$ since all incident triangles have been assigned. Each of the edges (v_i, v_{i+1}), $i = 0, \dots, n-1$ and vertices v_i, $i = 0, \dots, n$ is deleted if it has no more incident triangles. The remaining vertices in $\{v_1, \dots, v_{n-1}\}$ are added to the set of boundary vertices. The resulting graph is smaller and the invariant properties Q1, Q2 are maintained. If G_i is not connected, the assignment algorithm is applied to each connected component. The vertex v, which has been deleted, is assigned at most twice. The procedure is shown in Fig. 23.

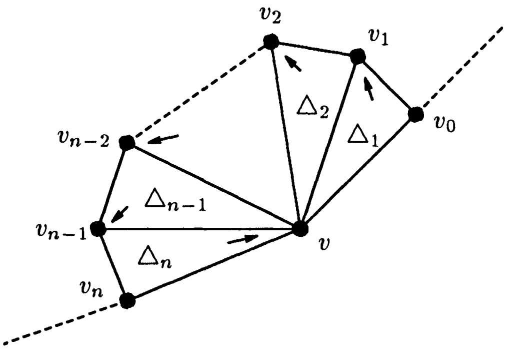


FIG. 23. *Assignment when* $v_1, \dots, v_{n-1}$ *are internal vertices.*

CASE 2. *Some of* $v_1, \dots, v_{n-1}$ *are boundary vertices.*

In this case, we will not make any assignment. Instead we will decompose G_i into two smaller subgraphs where the invariant properties Q1, Q2 are maintained in both subgraphs. Let v_m $(1 \leq m \leq n-1)$ be the boundary vertex with the smallest index m. Consider the edge (v, v_m) where v and v_m are boundary vertices. The edge (v, v_m) decomposes G_i into two subgraphs G_{ir} and G_{il} with (v, v_m) appearing in both subgraphs. There are two cases:

Case 2.1. v_m is saturated.

Case 2.2. v_m is not saturated.

CASE 2.1. v_m *is saturated.*

There are only two saturated vertices in G_i and by our selection, v is not saturated. Without loss of generality, we can assume that the other saturated vertex v_s is in G_{il}. We *force* vertex v in G_{ir} to be saturated, thus not allowing triangles in G_{ir} to be assigned to v. G_{ir} has exactly two saturated vertices: v_m and v (forced saturation), so does G_{il}: v_m and v_s. Vertex v in G_{il} will keep whatever assignment it carries in G_i but it will not be assigned by triangles in G_{ir}. Thus vertex v in G_i will not be oversaturated due to the forced saturation of v in G_{ir}. The decomposition is depicted in Fig. 24.

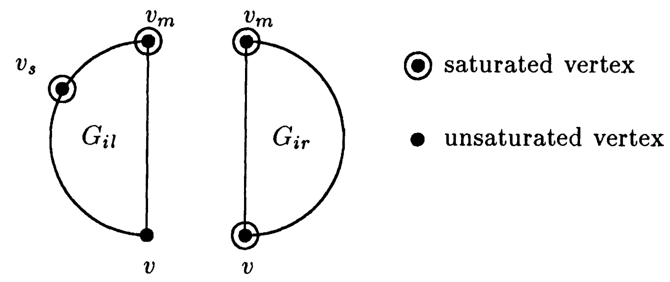


FIG. 24. *Decomposition when v_m is saturated.*

CASE 2.2. *v_m is not saturated.*

Let the two saturated vertices be v_{s1} and v_{s2}. v_{s1} (v_{s2}) appears in exactly one of the subgraphs after decomposition. If $v_{s1} \in G_{ir}$ and $v_{s2} \in G_{ir}$, we force v and v_m of G_{il} to be saturated. If $v_{s1} \in G_{ir}$ and $v_{s2} \in G_{il}$, we force v of G_{il} and v_m of G_{ir} to be saturated. If $v_{s1} \notin G_{ir}$ and $v_{s2} \notin G_{ir}$, we force v and v_m of G_{ir} to be saturated. In any case, G_{ir} and G_{il} have exactly two saturated vertices. v (v_m) of G_{il} is saturated if and only if v (v_m) of G_{ir} is not saturated. Thus v and v_m will not be oversaturated in G_i. The decomposition is shown in Fig. 25.

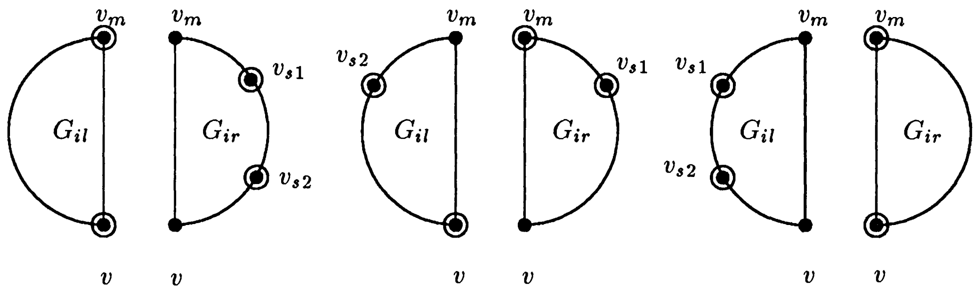


FIG. 25. *Decomposition when v_m is not saturated.* (a) $v_{s1}, v_{s2} \in G_{ir}$, (b) $v_{s1} \in G_{ir}$, $v_{s2} \in G_{il}$, (c) $v_{s1}, v_{s2} \notin G_{ir}$.

From the above discussion, we either eliminate some triangles or decompose G_i into smaller subgraphs. Thus, we have an assignment algorithm on G_i with the invariant properties as induction hypothesis. If G_i has only one triangle, the assignment problem is trivial.

A formal description of the algorithm is as follows:

```
ALGORITHM ASSIGN($CTG, TREE, root$)
INPUT: $CTG$ is a complex triangle graph of a planar triangulated graph and $TREE$
       is the corresponding containment tree. $root$ is the parent node of the set of
       complex triangles to be assigned in the current recursive step.
OUTPUT: A perfect assignment $ASSIGN = \{(\triangle_1, v_1), \ldots, (\triangle_n, v_n)\}$, where $\triangle_i =$
       $\triangle_{a_i b_i c_i}$ is a descendant complex triangle of $root$ and $v_i \in \{a_i, b_i, c_i\}$.

BEGIN
    Let $root$ be $\triangle_{abc}$ and assume it was assigned to $a$.
    Find the subgraph $G_i = (V_i, E_i)$ induced by the children of $root$.
    Force $v_{s1} = b$ and $v_{s2} = c$ to be saturated.
    If $a$ is saturated, force $a$ to have an assignment count of 1.
    $ASSIGN = \phi$.
    For each connected component $G_{ic}$ of $G_i$ do
        Let $G_{ic} = (V_{ic}, E_{ic})$.
        $B$ = CONNECTED_ASSIGN($G_{ic}, \{b, c\} \cap V_{ic}$).
        $ASSIGN = ASSIGN \cup B$.
    End For.
    For each $\triangle_{xyz} \in G_i$ do
        Let $G_{is}$ be the subgraphs induced by children of $\triangle_{xyz}$.
        $C$ = ASSIGN($G_{is}, TREE, \triangle_{xyz}$).
        $ASSIGN = ASSIGN \cup C$.
    End For.
    Return ($ASSIGN$).
END algorithm.
```

```
PROCEDURE CONNECTED_ASSIGN($G$, $SATURATED$)
INPUT: $G = (V, E)$ is a connected planar graph induced by children of a node in the
       containment tree. $SATURATED \subset V$ is the set of saturated boundary vertices
       of $G$, $|SATURATED| \leq 2$.
OUTPUT: A perfect assignment $A = \{(\triangle_1, v_1), \ldots, (\triangle_n, v_n)\}$, where $\triangle_i = \triangle_{a_i b_i c_i}$
       is a complex triangle of $G$ and $v_i \in \{a_i, b_i, c_i\}$.

BEGIN
    Find the set of boundary edges $E_b \subseteq E$.
    Find the set of boundary vertices $V_b \subseteq V$.
    $A = \phi$.
    Find a vertex $v \in (V_b - SATURATED)$.
    Let $v_0, \ldots, v_n$, $n \geq 1$ be the vertices where $(v, v_0), (v, v_n) \in E_b$ and $v_0, v_n \in V_b$.
    If $\{v_1, \ldots, v_{n-1}\}$ are all internal vertices then
        /* CASE 1, assign $\triangle_i$ to $v_i$ and $\triangle_n$ to $v$ */
        Let $\triangle_i = \triangle_{v_{i-1}\, v_i\, v}$, $i = 1, \ldots, n$.
        For $i = 1$ to $n - 1$ do
            $A = A \cup \{(\triangle_i, v_i)\}$ if $\triangle_i$ exists.
        $A = A \cup \{(\triangle_n, v)\}$ if $\triangle_n$ exists. /* last triangle */
        Delete vertex $v$ and incident edges.
        Delete vertex $v_i$, $i = 0, \ldots, n$ from $G$ if it has no more incident triangle.
        Delete edge $(v_i, v_{i+1})$, $i = 0, \ldots, n - 1$ from $G$ if it has no more incident triangle.
        For each connected component $G_c = (V_c, E_c)$ of the remaining graph do
```

```
            $A = A \cup$ CONNECTED_ASSIGN($G_c$, $SATURATED \cap V_c$).
    Else
        /* CASE 2, decompose $G$ into $G_l$ and $G_r$ */
        Let $v_{s1}$ and $v_{s2}$ be two vertices of $SATURATED$.
        Let $v_m \in \{v_1, \ldots, v_{n-1}\} \cap V_b$ where $v_1, \ldots, v_{m-1} \notin V_b$.
        $SATURATED\_L = \phi$.
        $SATURATED\_R = \phi$.
        Decompose $G$ into $G_l$ and $G_r$ with $(v, v_m) = G_l \cap G_r$.
        If $v_m \in SATURATED$ then
            /* CASE 2.1 */
            Let $v_{s1} = v_m$, without loss of generality.
            If $v_{s2}$ in $G_l$ then
                $SATURATED\_L = \{v_{s2}, v_m\}$.
                $SATURATED\_R = \{v, v_m\}$.
            Else
                $SATURATED\_L = \{v, v_m\}$.
                $SATURATED\_R = \{v_{s2}, v_m\}$.
            End If.
        Else /* $v_m \notin SATURATED$ */
            /* CASE 2.2 */
            If $v_{s1} \in G_r$ then
                $SATURATED\_R = SATURATED\_R \cup \{v_{s1}\}$.
                $SATURATED\_L = SATURATED\_L \cup \{v\}$.
            Else
                $SATURATED\_R = SATURATED\_R \cup \{v\}$.
                $SATURATED\_L = SATURATED\_L \cup \{v_{s1}\}$.
            End If.
            If $v_{s2} \in G_l$ then
                $SATURATED\_R = SATURATED\_R \cup \{v_m\}$.
                $SATURATED\_L = SATURATED\_L \cup \{v_{s2}\}$.
            Else
                $SATURATED\_R = SATURATED\_R \cup \{v_m\}$.
                $SATURATED\_L = SATURATED\_L \cup \{v_{s2}\}$.
            End If.
        End If.
        $B$ = CONNECTED_ASSIGN($G_l$, $SATURATED\_L$).
        $C$ = CONNECTED_ASSIGN($G_r$, $SATURATED\_R$).
        $A = A \cup B \cup C$.
    End if.
    Return ($A$).
END procedure.
```

Notice that the sharing assignment scheme is implicitly incorporated by forcing vertex a of $\triangle_{abc}$ in **ASSIGN()** to reduce its assignment count. Data structures that support **ASSIGN()** and **CONNECTED_ASSIGN()** are relatively simple. A doubly connected edge list [8] can be used to find CTs incident to a vertex. The algorithm visits each CT without backtracking. Thus the complexity of the assignment algorithm is $O(c)$, where c is the number of CTs. c is $O(|V|)$ in a planar graph. Therefore the overall time complexity is $O(|V|)$ and we have the following theorem:

THEOREM 2. *Given a PTG, a perfect assignment can be constructed in $O(|V|)$ time, where $|V|$ is the number of vertices of the PTG.*

4.3. Perfect assignment for extended graph. In general the algorithm will make an assignment to any nonsaturated vertex of G_i. We have pointed out that in a perfect assignment of G, the four noncore vertices r, u, l, b are not allowed to carry assignment. This may violate the invariant properties because in the worst case, we may have four saturated vertices to begin with. However, we know that all core vertices have an assignment count of zero initially. We can exploit this fact to decompose G into two subgraphs satisfying the invariant properties.

Given a biconnected PTG G_c, we construct an extended graph G by adding the vertices r, u, l, b. We force the vertices r, u, l, b to be saturated. Consider the graph $G' = G - \{(r,u),(u,l),(l,b),(b,r)\}$. Since G_c is biconnected, the edges (r,u), (u,l), (l,b), and (b,r) do not appear in any CT of G. Therefore the CTG of G' and that of G are identical, and we will only consider decomposition of G'.

If each connected component of CTG $CTG(G')$ contains no more than two saturated vertices, we are done. Suppose a connected component of $CTG(G')$ contains three or more saturated vertices. We identify the first boundary vertex v_r of G', which is clockwise adjacent to r. Similarly we find v_l. (See Fig. 26.) Since G_c is biconnected, v_r and v_l must be distinct. We find a path $P = (v_r, \ldots, v_l)$ in G' and construct a path P^* by applying the chord elimination procedure. By our selection, v_l and v_r are located at the infinite face of G'. We cut G' into G'_1 and G'_2 along the path P^*, as shown in Fig. 26. $CTG(G'_1)$ ($CTG(G'_2)$) has at most two saturated vertices since G'_1 (G'_2) has at most two saturated vertices. Vertices of P^* in $CTG(G'_1)$ and $CTG(G'_2)$ are forced to have assignment count of one. Therefore vertices of P^* in $CTG(G')$ will not be oversaturated after we apply the assignment algorithm to $CTG(G'_1)$ and $CTG(G'_2)$.

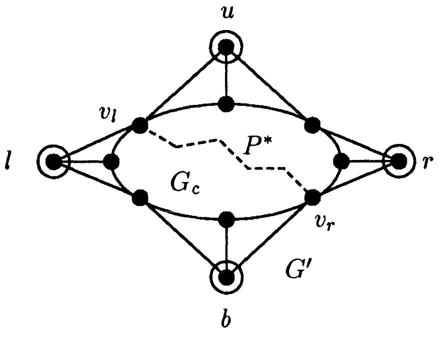


FIG. 26. *Initial decomposition of G.*

THEOREM 3. *Given any extended graph G constructed from a biconnected PTG G_c, a perfect assignment can be constructed in $O(|V|)$ time, where $|V|$ is the number of vertices of G_c.*

Note that the biconnected condition of G is needed only for the initial decomposition of G'. The assignment algorithm is still valid for some classes of weakly connected core graphs as long as the initial decomposition exists. However, not all extended graphs of planar graphs can be decomposed in this manner. Figure 27 is an example of such a graph.

As a direct result of Theorems 1 and 3, we have the follwing theorem:

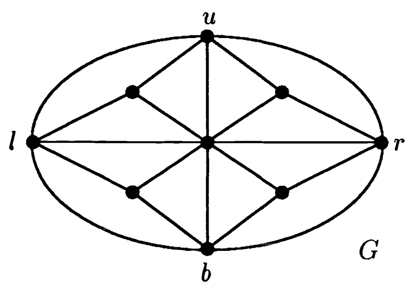


FIG. 27. *An example of G that does not have a perfect assignment.*

THEOREM 4. *Given any biconnected PTG G_c with $|V|$ vertices, a 2-CRM floor-plan F can be constructed in $O(|V|)$ time, where the dual of F has core G_c when parallel edges are eliminated.*

Example. We demonstrate an example of the construction process using the graph of Fig. 8, which does not admit any 1-CRM floor-plan. The results are shown in Fig. 28: (a) is the core graph and (b) shows an extended graph. (c) is a perfect assignment and (d) gives the 2-CRM dual floor-plan constructed with the perfect assignment in (c).

5. Conclusion. We have presented a linear time algorithm for floor-plan construction using a graph dualization technique. By allowing more general shapes, 1-CRM and 2-CRM, we have shown that all adjacency requirements of a PTG can be satisfied on a rectangular chip. The construction ensures that no unnecessary adjacency is added in the final floor-plan, thus paves the way for a more compact layout. If the input graph is planar but not triangulated, a floor-plan is still possible by introducing trivial adjacency. The most complicated shape created is 2-CRM but most modules are rectangular. The 1-CRM and 2-CRM can be easily incorporated in the widely used rectilinear systems. The expected number of CTs for a randomly generated PTG is approximately 16% of the number of vertices [14].

The algorithm presented serves as a theoretical basis for floor-planning based on the graph dualization approach. For practical applications, other requirements such as area, aspect ratio, and perimeter constraints should be considered. We are currently looking at the applications of the results obtained from this study. Sizing and aspect ratio is one of our major focuses. Only when such issues are resolved could one propose dualization-based floor-planning as a practical alternative. The floor-plan sizing problem has been shown to be NP-complete for general rectangular floor-plans [9]. Future research is focused on approximate solutions or restriction to special classes of floor-plans, for example, sliceable floor-plans, which are more promising in practice. More information can be found in [7] and in Chapter 7 of [5].

One interesting problem is to enumerate all floor-plans with identical duals. For rectangular floor-plans, the problem have been studied [3], [13]. For 1- and 2-CRM floor-plans, the problem has yet to be studied. Planarization of the circuit is especially not well understood. Current solutions are based on heuristic approaches [6]. Many problems related to graph dualization need to be investigated even for rectangular floor-plans.

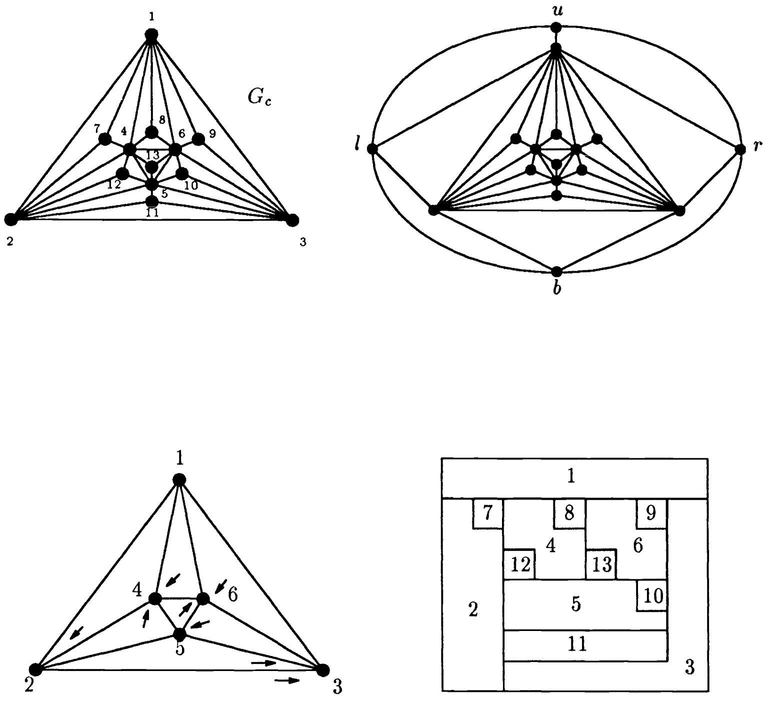


FIG. 28. *An example of dual construction:* (a) *core graph* G_C; (b) *an extended graph constructed from* G_C; (c) *a perfect assignment; and* (d) *a floor-plan constructed from the perfect assignment in* (c).

REFERENCES

[1] J. BHASKER AND S. SAHNI, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica, 3 (1988), pp. 247–278.

[2] K. KOZMINSKI AND E. KINNEN, *Rectangular dual of planar graphs*, Networks, 15 (1985), pp. 145–157.

[3] ———, *Rectangular dualization and rectangular dissection*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 1401–1416.

[4] Y. T. LAI AND S. M. LEINWAND, *Algorithms for floor-plan design via rectangular dualization*, IEEE Trans. Computer-Aided Design, 7 (1988), pp. 1278–1289.

[5] T. LENGAUER, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, New York, 1990.

[6] B. LOKANATHAN AND E. KINNEN, *Performance optimized floor planning by graph planarization*, Proc. of 26th Design Automation Conference, 1989, pp. 116–121.

[7] R. H. J. M. OTTEN, *Automatic floorplan design*, Proc. of 19th Design Automation Conference, 1982, pp. 261–267.

[8] F. PREPARATA AND M. SHAMOS, *Computational Geometry—An Introduction*, Springer-Verlag, Berlin, New York, 1985.

[9] L. STOCKMAYER, *Optimal orientation of cells in slicing floorplan designs*, Inform. and Control, 57 (1983), pp. 91–101.

[10] Y. SUN AND M. SARRAFZADEH, *Floorplanning by graph dualization: L-shaped models*, Proc. of IEEE International Symposium on Circuits and Systems, 1990, pp. 2845–2848; also to appear in Algorithmica.

[11] Y. SUN AND K. H. YEAP, *Edge covering of complex triangles in rectangular dual floorplanning*, J. Circuits Systems Comput., to appear.

[12] S. TSUKIYAMA, K. KOIKE, AND I. SHIRAKAWA, *An algorithm to eliminate all complex triangles in a maximal planar graph for use in VLSI floor-plan*, Proc. of IEEE International Symposium on Circuits and Systems, 1986, pp. 321–324.

[13] S. TSUKIYAMA, K. TANI, AND T. MARUYAMA, *A condition for a maximal planar graph to have a unique rectangular dual and its application to VLSI floor-plan*, Proc. of IEEE International Symposium on Circuits and Systems, 1989, pp. 931–934.

[14] K. H. YEAP AND M. SARRAFZADEH, *A theorem of sliceability*, 2nd Great Lakes Computer Science Conference, Western Michigan University, Kalamazoo, MI, 1991.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 527–551, June 1993

A QUADRATIC TIME ALGORITHM FOR THE MINMAX LENGTH TRIANGULATION*

HERBERT EDELSBRUNNER† AND TIOW SENG TAN‡

Abstract. It is shown that a triangulation of a set of n points in the plane that minimizes the maximum edge length can be computed in time $O(n^2)$. The algorithm is reasonably easy to implement and is based on the theorem that there is a triangulation with minmax edge length that contains the relative neighborhood graph of the points as a subgraph. With minor modifications the algorithm works for arbitrary normed metrics.

Key words. computational geometry, point sets, triangulations, two dimensions, minmax edge length, normed metrics

AMS(MOS) subject classifications. 68Q25, 68U05, 65D05

1. Introduction.

A *triangulation* of a (finite) point set S in $\Re^2$ is a maximal connected straight-line plane graph whose vertices are the points of S. Maximality implies that, with the exception of the unbounded face, each face of the graph is a triangle. The number of different triangulations of S depends on $n = |S|$ and on the relative location of the points. As implied by a result in [ACNS82], 10^{13n} is an upper bound on the number of triangulations of any set of n points in $\Re^2$. Furthermore, if S is in convex position, then it admits $\frac{1}{n-1}\binom{2n-4}{n-2} \geq 2^{n-3}$ different triangulations. In order to choose an optimal triangulation, under some criterion, it is thus not feasible to exhaustively search the set of all triangulations.

Indeed, except for a handful of particular optimality criteria, the problem of finding an optimal triangulation for a given point set is hard, that is, no polynomial-time algorithms are known. Among these exceptions are the maxmin angle criterion [Sibs78], the minmax angle criterion [ETW92], the minmax smallest enclosing circle criterion [Raja91], and the minmax circumscribed circle criterion. The optimum under the first, third, and fourth criteria is achieved by the Delaunay triangulation which can be constructed in time $O(n \log n)$ [Del34], [PrSh85], [Edel87].

In this paper we study the complexity of minimizing the maximum edge length. A triangulation that minimizes the length of its longest edge is called a *minmax length triangulation*. It is related to the so-called *minimum length* (or *minimum weight*) *triangulation* that minimizes the sum of the edge lengths. The latter problem has been studied by Plaisted and Hong [PlHo87], Lingas [Ling87], and others. In spite of the lack of a proof that the problem is NP-hard, no polynomial time algorithm for constructing a minimum length triangulation is currently known. Even more annoying is the lack of a constant approximation scheme, that is, an algorithm that in polynomial time constructs a triangulation guaranteed to have total edge length of at most some constant times the optimum. The current best approximation scheme, described in [PlHo87], guarantees a factor of $O(\log n)$.

In view of the apparent difficulty of computing minimum length triangulations, it is somewhat surprising that we are able to provide a polynomial, in fact, a quadratic time algorithm, for constructing a minmax length triangulation. To our knowledge it is

*Received by the editors March 20, 1991; accepted for publication (in revised form) March 4, 1992.

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The work of this author was supported by the National Science Foundation grants CCR-8921421.

‡Department of Information Systems and Computer Science, National University of Singapore, Republic of Singapore. The work of this author was done during his study leave at the University of Illinois at Urbana-Champaign.

the first polynomial-time algorithm for this problem. Although there is evidence for the potential usefulness of such a triangulation (see [BrZl70], [WGS90]), we consider the additional insight into optimum triangulations under edge length criteria to be the main contribution of this paper.

The reader might find it instructive to rule out seemingly promising approaches to computing minmax length triangulations before diving into the occasionally involved developments of the forthcoming sections. Note first that the Delaunay triangulation does not minimize the maximum edge length (see also §2). Second, the incremental greedy method, which repeatedly adds the shortest edge that does not intersect any previously added edge, also fails to minimize the maximum edge length. Third, let us take a brief look at the decremental greedy method that throws away edges in the order of decreasing length. It stops the deletion process if another deletion would render the set of edges so that it does not contain any triangulating subset (see Wismath [Wism80, p. 81]). The trouble with this approach is that it is not clear how to efficiently decide whether the evolving edge set is still sufficient to triangulate the point set. Indeed, Lloyd [Lloy77] proves that the general version of this problem (to decide whether a given edge set contains a triangulation) is NP-complete. Finally, the iterative methods that use the edge-flip [Laws77] or the more general edge-insertion operation [ETW92] can get caught in local optima. The approach taken in this paper is entirely different from the above paradigms.

The organization of this paper is as follows. Section 2 reviews a few results on relative neighborhood graphs and other subgraphs of the Delaunay triangulation. Section 3 formulates the global algorithm; its straightforward implementation using dynamic programming takes time $O(n^3)$. The only intricate part of this algorithm is the proof of correctness provided in §4. Sections 5 and 6 present a specialized polygon triangulation algorithm that can be used to speed up the general algorithm to time $O(n^2)$. Whereas §§2–6 assume that the Euclidean metric is used to measure length, §7 demonstrates that all results extend to general normed metrics. Indeed, the arguments in §§2–6 are axiomatically derived from a few basic lemmas in order to minimize the number of changes necessary to generalize the results. Finally, §8 briefly discusses the contributions of this paper and states some related open problems.

2. Subgraphs of the Delaunay triangulation. Our approach to constructing a minmax length triangulation first adds enough edges to decompose the plane into simple polygonal regions and then (optimally) triangulates these regions. Both Plaisted and Hong [PlHo87] and Lingas [Ling87] used this approach to compute approximations of the minimum length triangulation. In our case the initial set of edges is provided by the (boundary of the) convex hull and the relative neighborhood graph of the point set S. The remainder of this section formally introduces these graphs, along with the Delaunay triangulation and the minimum spanning tree of S, and reviews some basic facts about their relationships. If x, y, z are three points in $\Re^2$, then xy denotes the relatively open line segment with endpoints x and y, $|xy|$ denotes its length, and xyz denotes the open triangle with vertices x, y, z.

The *Delaunay triangulation* of S, denoted by $dt(S)$, contains an edge ab, $a, b \in S$, if there is a circle through a and b so that all other points lie outside the circle. If the points are in general position, then $dt(S)$ is indeed a triangulation.

As mentioned in §1, the Delaunay triangulation does not minimize the length of the longest edge. Take, for example, the points $a = (-2, 0)$, $b = (1, \sqrt{3})$, $c = (1, -\sqrt{3})$, $d = (2 - \epsilon, 0)$, with $0 < \epsilon < 1$. They form a convex quadrilateral $abdc$, and the Delaunay triangulation uses ad as the fifth edge. As ϵ approaches 0 the length of ad approaches

$2/\sqrt{3}$ times the length of the longest edge in the alternative triangulation. Indeed, $2/\sqrt{3}$ is the worst possible ratio, as can be shown by using the result of [Raja91] that the Delaunay triangulation minimizes the radius of the maximum smallest enclosing circle, where the maximum is taken over all triangles. If the radius of this circle is 1, then the longest edge of the Delaunay triangulation has length of at most 2. By the optimality result every minmax length triangulation has a smallest enclosing circle of radius at least 1 and therefore an edge of length at least $\sqrt{3}$ (see also [WGS90]).

The *convex hull* of S is the smallest convex polygon that contains S. We define $ch(S)$ as the graph defined by the edges of this polygon. In the (degenerate) case in which three or more collinear points lie on the boundary of this polygon we think of each such point as a vertex of the polygon. Thus edges are taken only between adjacent collinear points. Each convex hull edge is an edge of *every* triangulation of S and therefore also of every minmax length triangulation.

An edge ab belongs to the *relative neighborhood graph* of S, denoted by $rng(S)$, if

$$|ab| \leq \min_{x \in S - \{a,b\}} \max\{|xa|, |xb|\}.$$

This definition goes back to Toussaint [Tous80], who modified a similar definition by Lankford [Lank69] for use in pattern recognition. Alternatively, we can define the *lune* of ab as the set $\{x \in \Re^2 : \max\{|xa|, |xb|\} < |ab|\}$, and then define $rng(S)$ as the set of edges ab whose lunes have empty intersection with S.

A *minimum spanning tree* of S, denoted by $mst(S)$, is a spanning tree of S that minimizes the total edge length; it also minimizes the maximum edge length.

All four graphs, $dt(S), ch(S), rng(S), mst(S)$, are plane and connected, and, with the exception of $ch(S)$, they span S. Where convenient we will interpret these graphs as edge sets. Plainly, $ch(S) \subseteq dt(S)$, and, as observed by Toussaint [Tous80], we also have $mst(S) \subseteq rng(S) \subseteq dt(S)$. Obviously, $ch(S) \subseteq mlt(S)$ for every minmax length triangulation $mlt(S)$, and we will show in §4 that there exists an $mlt(S)$ so that $rng(S) \subseteq mlt(S)$.

3. The global algorithm. As mentioned above, there exists a minmax length triangulation $mlt(S)$ that contains all edges of $ch(S)$ and $rng(S)$. Because $ch(S) \cup rng(S)$ is a connected graph, it decomposes the convex hull of S into simple *polygonal regions*, which we define as open sets, that contain no points of S. It is thus natural to construct $mlt(S)$ by computing $ch(S) \cup rng(S)$ and then (optimally) triangulating each polygonal region.

Strictly speaking, however, the polygonal regions are not necessarily simple polygons in the usual sense of the term, although their interiors *are* simply connected. The difference is that the interior of the closure of a polygonal region is not necessarily the same as the region itself; it may contain edges of the region, and it may be nonsimply connected. The most effective way to deal computationally with this minor difficulty is to represent each edge by a pair of oppositely directed edges and to represent the boundary of each region by the collection of directed edges for which the region lies on their left-hand side. In effect, this means that we interpret each polygonal region as a genuine simple polygon simply by pretending that its zero-width cracks are opened up a tiny amount. In most cases this is a convenient interpretation, and the notation will be adjusted accordingly. Only occasionally will the difference between a simple polygonal region and a simple polygon be uncovered.

Let us now formally specify the algorithm and give a preliminary analysis.

Input. A set S of n points in $\Re^2$.

Output. A minmax length triangulation of S.

Algorithm.
1. Construct $ch(S)$ and $rng(S)$.
2. Determine the polygonal regions defined by $ch(S) \cup rng(S)$.
3. Find a minmax length triangulation for each such polygonal region.

Step 1 can be carried out in time $O(n \log n)$ by using results documented in [PrSh85] and [Supo83] (see also [JKY90]). By using the standard quad-edge data structure of [GuSt85] for storing the plane graph $ch(S) \cup rng(S)$, step 2 can be accomplished in time $O(n)$. Finally, we can use dynamic programming to compute an optimal triangulation for each polygon in time cubic and storage quadratic in the number of its vertices (see [Klin80], [Gilb79]). This adds up to time $O(n^3)$ and storage $O(n^2)$. The correctness of the algorithm will be established in §4. Sections 5 and 6 will show how to speed up the algorithm to time $O(n^2)$ by using a specialized polygon triangulation algorithm.

4. The Subgraph Theorem. The main result of this section is what we call the Subgraph Theorem, which was announced earlier. We begin with two elementary geometry lemmas about distances between four points in convex and in nonconvex position.

□-LEMMA. *For a convex quadrilateral abcd we have* $|ab| + |cd| < |ac| + |bd|$.

Proof. Let x be the intersection point of the two diagonals ac and bd. Clearly, $|ab| + |cd| < (|ax| + |xb|) + (|cx| + |xd|) = |ac| + |bd|$. □

In words, the total length of the two diagonals of a convex quadrilateral always exceeds the total length of two opposite sides. This is true even if three of the four vertices are collinear. It implies that if one diagonal is no longer than one of the edges, then the other diagonal is longer than the opposite edge.

Δ-LEMMA. *Let a, b, c, d be four distinct points so that the closure of the triangle abc contains d. Then* $|ad| < \max\{|ab|, |ac|\}$.

Proof. If a, b, c, d are collinear, the result is obvious. Otherwise, let d' be the intersection of the edge bc with the line passing through a and d, and note that $|ad| \leq |ad'|$. Of all points on bc only the endpoints can possibly maximize the distance to a. The assertion follows because if d' is an endpoint of bc, then $d \neq d'$ and therefore ad is strictly shorter than ad'. □

Note that the length of the longest edge of any minimum spanning tree is no longer than the longest edge of any triangulation of S. This follows trivially from the fact that every triangulation contains a spanning tree. It is not very difficult to prove that the same is true for the relative neighborhood graph of S. First we need some notation. The circle with center x and radius ρ is denoted by (x, ρ), and the *bisector* of two points p and q is the set of points equidistant to both.

LENGTH LEMMA. *Every triangulation of S contains an edge that is at least as long as the longest edge of $rng(S)$.*

Proof. Let pq be the longest edge of $rng(S)$, and let $t(S)$ be an arbitrary triangulation of S. If $pq \in t(S)$, there is nothing to prove. Otherwise, pq intersects edges $r_1s_1, r_2s_2, \ldots, r_ks_k$ of $t(S)$, sorted from p to q, with all r_i on one side of the line through p and q and all s_i on the other. If pq is longer than all edges in $t(S)$, then r_1 and s_1 are both inside the circle $C_p = (p, |pq|)$ because pr_1 and ps_1 are both edges of $t(S)$. By the definition of $rng(S)$, r_1 and s_1 are thus outside or on the circle $C_q = (q, |pq|)$. Therefore, r_1 and s_1 lie in the half-plane of points closer to p than to q. Symmetrically, r_k and s_k lie inside C_q and outside or on C_p and therefore in the half-plane of points closer to q than to p. For each $1 \leq i \leq k-1$ we have either $r_i = r_{i+1}$ or $s_i = s_{i+1}$, which implies that there is an index j so that r_j and s_j do not lie on the same side of the bisector of pq. But then the □-Lemma implies that $|r_j s_j| > |pq|$ because $|pq|$ is no longer than each of two opposite edges of the convex quadrilateral $pr_j qs_j$, a contradiction. □

The proof of the Subgraph Theorem is similar to that of the Length Lemma, although it is considerably more involved. The basic idea is to assume an extremal counterexample and to contradict its existence by retriangulating parts of it by using no long edges. In the following we first develop three facts that show the possibilities of retriangulations, and then we prove the theorem.

Let $t(S)$ be a minmax length triangulation of S that does not contain some edge pq of $rng(S)$. Suppose pq intersects the triangles $t_1, t_2, \ldots, t_k$ of $t(S)$ sorted from p to q (see Fig. 1 left).

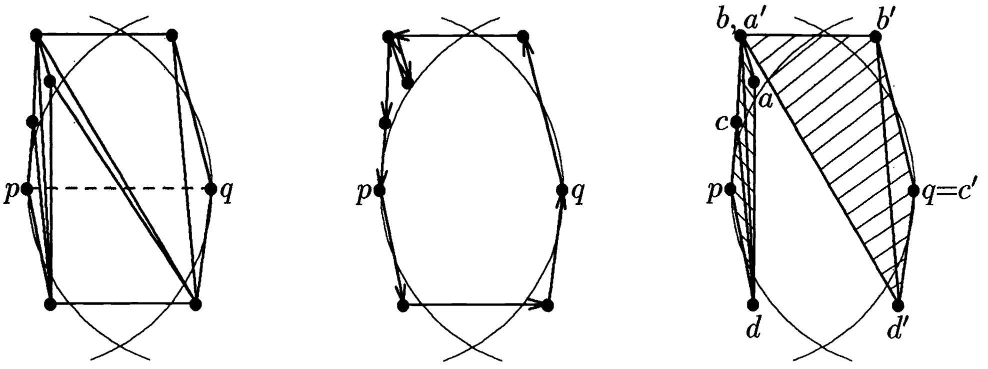


FIG. 1. *To the left are the triangles of $t(S)$ that intersect pq. If we remove the edges that intersect pq we get a polygon whose boundary is oriented in a counterclockwise order. The prefix P and the suffix Q defined for this configuration are illustrated to the right. Although b and a' are the same point, they refer to different angles of this point.*

The deletion of the edges that intersect pq would result in a simply connected region, which can be interpreted (as in §3) as a polygonal region—we treat each edge in its boundary as a pair of edges with opposite direction, and we trace the boundary of the region by traversing all directed edges that have the region on their left side. Any two consecutive (directed) edges define an *angle* (see Fig. 1 middle). Note that a vertex can correspond to many angles, although the common situation is that it corresponds only to one. We will therefore sometimes ignore the difference between vertices and corresponding angles. Points p and q correspond to only one angle each. An angle is *convex* if the two defining edges form a left turn. Call the sequence of edges from p to q the *lower chain*, and the sequence from q to p the *upper chain*. Each chain contains at least one convex angle different from p and q.

A *prefix* is an initial subsequence of $t_1, t_2, \ldots, t_k$, and a *suffix* is a terminal subsequence of $t_1, t_2, \ldots, t_k$. We say that a prefix (suffix) *covers* an angle of the polygon if it contains all triangles incident to this angle. Let i be minimal so that the prefix $P = t_1, t_2, \ldots, t_i$ covers a convex angle other than p, and let j be maximal so that the suffix $Q = t_j, t_{j+1}, \ldots, t_k$ covers a convex angle other than q. P and Q consist of at least two triangles each. We let b be the convex angle (vertex) covered by P—it is incident to both t_i and t_{i-1}—and we let d be the other vertex common to t_i and t_{i-1}. Furthermore, c is the third vertex of t_{i-1} and a is the third vertex of t_i (see Fig. 1 right). Symmetrically, define vertices b', d', c', a' of Q. We say that P (respectively, Q) is *type* 1 if the last (respectively, first) two triangles of P (respectively, Q) are the only ones incident to b

(respectively, b'), and we say that it is *type* 2 otherwise (see Fig. 2). If P is type 1, then a, b, c belong to the same chain and d belongs to the other chain (this includes the case that $c = p$), and if P is type 2, then a, b belong to one chain and c, d belong to the other.

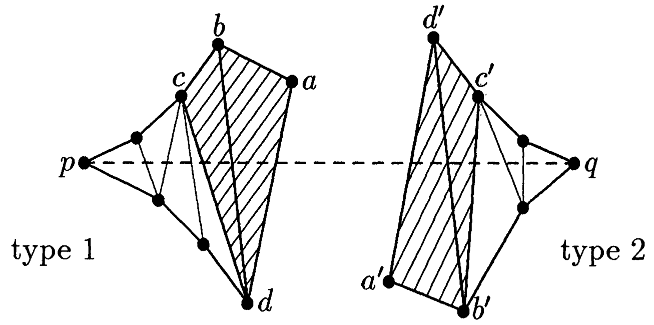


FIG. 2. *The definitions prefix P with vertices a, b, c, d and the suffix Q with vertices a', b', c', d' depend on pq. P is type 1, and Q is type 2. For illustration purposes the constraint that all vertices must lie outside the lune of pq has been ignored.*

FACT 1. *$P = t_1, t_2, \ldots, t_i$ and $Q = t_j, t_{j+1}, \ldots, t_k$ share at most two triangles, that is, $i - 1 \leq j$.*

Proof. We show that since the suffix $R = t_{i-1}, t_i, \ldots, t_k$ covers at least one convex angle other than q, Q cannot be bigger than R. If P is type 1, then R covers b, which is convex. Otherwise, R covers all angles between d and q, d included. Since all angles between p and d, p and d excluded, are nonconvex, at least one angle between d and q must be convex, and this angle is covered by R. □

It should be clear that $abcd$ and $a'b'c'd'$ are both convex quadrilaterals by the choice of their vertices. The next two facts imply that either $abcd$ or $a'b'c'd'$ or both have alternate triangulations that use ac or $a'c'$ while the maximum edge length of $t(S)$ is maintained. In other words, bd, or $b'd'$, or both can be switched. Formally, we call bd (respectively, $b'd'$) *switchable* if ac (respectively, $a'c'$) is no longer than the longest edge of $t(S)$. Fact 2 shows strong locality constraints for a and d (respectively, a' and d') if bd (respectively, $b'd'$) is not switchable. Define

$$A = \{x \in \Re^2 : |xp| > |pq| \text{ and } |xp| > |xq|\} \text{ and}$$

$$D = \{x \in \Re^2 : |xp| \geq |pq| \text{ and } |xq| < |pq|\},$$

with the understanding that A and a belong to one half-plane defined by the line passing through p and q and that D and d belong to the other (see Fig. 3).

FACT 2. *If bd is not switchable then $a \in A$ and $d \in D$.*

Proof. Since bd is not switchable, ac must be longer than the other five edges defined by a, b, c, d, and by the Length Lemma it must be longer than pq. We first show that $|ac| \leq |ap|$ and then derive the four inequalities needed to establish the claim.

(1) $|ac| \leq |ap|$. We can assume that $c \neq p$. Note that c is contained in the closure of triangle bdp. Since the line passing through b and d separates a from p, the closures of the two triangles abp and adp cover bdp completely, and therefore one of them contains

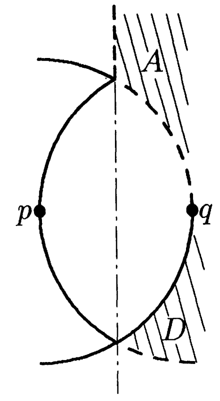


FIG. 3. *Regions A and D as defined for the case in which a is on the upper chain.*

c. If c lies in the closure of abp, the claim follows from $|ab| < |ac|$ and the Δ-Lemma for abp, and if c lies in adp, it follows from $|ad| < |ac|$ and the Δ-Lemma for adp.

(2) $|ap| > |pq|$. From the Length Lemma we get $|pq| < |ac|$, and from (1) we get $|ac| \leq |ap|$.

(3) $|dq| < |pq|$. Assume $|dq| \geq |pq|$. The $\Box$-Lemma for $paqd$ implies $|ad| > |ap|$ and thus $|ad| > |ac|$ because of (1), a contradiction.

(4) $|dp| \geq |pq|$. This is immediate from (3) because pq is an edge of $rng(S)$.

(5) $|ap| > |aq|$. Assume $|ap| \leq |aq|$, and recall $|dp| \geq |pq|$ from (4). By the $\Box$-Lemma for $paqd$ we get $|ad| > |aq|$, which implies $|ad| > |ap|$ by assumption and $|ad| > |ac|$ by (1), a contradiction.

The proof of Fact 2 is now complete because (2) and (5) are equivalent to $a \in A$ and because (3) and (4) are equivalent to $d \in D$. $\Box$

Symmetrically, we define regions A' and D', which are where a' and d' must lie if $b'd'$ is not switchable. Using Facts 1 and 2, we can now show that there is always an edge that can be switched.

FACT 3. *It is not possible that both bd and b'd' are nonswitchable.*

Proof. If bd and $b'd'$ are both nonswitchable, then ad lies on q's side of the bisector of pq and $a'd'$ lies on p's side by Fact 2. Because of Fact 1 and because ad is the last edge of P and $a'd'$ is the first edge of Q, we have $\{a, d, a', d'\} = \{a, b, c, d\} = \{a', b', c', d'\}$. Furthermore, the fact that bd and $b'd'$ are both edges of $t(S)$ implies that they are the same and thus that $b = d'$, $d = b'$, $a = c'$, $c = a'$ (see Fig. 4). It follows that the polygonal region has the shape of a diamond, with p, b, q, d as the only convex angles. This contradicts the locality constraints for a, b, c, d stated in Fact 2. In particular, the chain from p to $d \in D$ (as indicated by the dotted chain in Fig. 4) is concave or straight and therefore enclosed by the circle $(q, |pq|)$. It follows that this chain is disjoint from A', which is where $c = a'$, the predecessor of d in this chain, is supposed to lie. $\Box$

With the above results and notations we now choose an extremal counterexample to prove the main result of this section.

SUBGRAPH THEOREM. *Every finite point set S in $\Re^2$ has a minmax length triangulation $mlt(S)$ so that $rng(S) \subseteq mlt(S)$.*

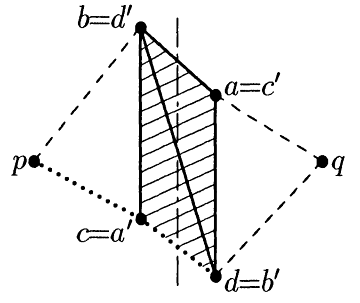


FIG. 4. *If bd and b′d′ are both nonswitchable, then b and d are the only convex angles besides p and q.*

Proof. We assume there is a set S so that no minmax length triangulation contains $rng(S)$. Let $t(S)$ be a minmax length triangulation of S that satisfies the following extremal properties, where later properties are contingent on earlier ones.

(i) $t(S)$ minimizes the number of edges that intersect pq.

(ii) $t(S)$ minimizes the number of edges incident to b that intersect pq.

(iii) $t(S)$ minimizes the number of edges incident to b' that intersect pq.

It is conceivable that $t(S)$ is not unique, but it will be sufficient to assume that $t(S)$ is any one of the remaining triangulations.

By Fact 3 either bd or $b'd'$, or both are switchable. If bd is switchable and P is type 1, then the number of edges that intersect pq decreases when bd is switched. This contradicts property (i). Thus P must be type 2 if bd is switchable, and, similarly, Q must be type 2 if $b'd'$ is switchable. When we switch bd, the degree of b decreases, which contradicts property (ii). Thus it must be that bd is not switchable and that $b'd'$ is. But switching $b'd'$ decreases the degree of b', which would contradict property (iii) unless the degree of b increases at the same time. Remember that because (iii) is contingent on (ii), so if (ii) is not satisfied any more, then we cannot draw any conclusion. Thus the configuration left for analysis is as shown in Fig. 5.

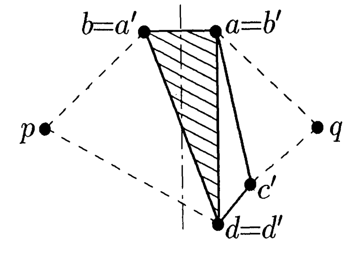


FIG. 5. *In the final configuration, because bd is nonswitchable,* $a \in A$ *and* $d \in D$, *and because b′d′ is switchable, so Q is type 2. Furthermore, because switching b′d′ to a′c′ increases the degree of b, so* $a' = b$, *and therefore P and Q overlap in exactly one triangle. The figure ignores the fact that by rights all points should lie outside the lune of pq.*

To reach the final contradiction we switch $b'd'$ and redefine Q on the basis of the new configuration. Since all angles from (the old) d' to q are nonconvex, the new points b' and a' are the same as before and the new d' is the old c'. Thus we can again switch

$b'd'$, and so on, until Q is type 1 or $c' = q$, at which point the next switch decreases the number of edges intersecting pq. This finally contradicts property (i). □

Remark. A natural extension of minimizing the length of the longest edge in a triangulation is to also minimize the length of the second-longest edge, and so on. Let $mvt(S)$ be a triangulation that minimizes the entire vector of edge lengths in this fashion. If the points of S are in general position, then $mvt(S)$ is unique. Curiously, it is not always true that (there is an) $mvt(S)$ (that) contains $rng(S)$ as a subgraph. The smallest example that illustrates this observation consists of four points a, b, c, d so that c and d lie fairly close to b, ab and cd intersect, and c and d both lie outside the circle $(a, |ab|)$.

5. Triangulating rng-polygons. The goal of this section and §6 is to improve the cubic-time algorithm of §3 to quadratic time. This is done by using a specialized polygon triangulation algorithm. The main part of the algorithm and the structural properties of minmax length triangulations that guarantee its correctness are developed in this section.

Recall that the first two steps of the algorithm in §3 decompose the convex hull of S into polygonal regions by drawing all edges of $ch(S)$ and $rng(S)$; these steps remain unaltered. Each region is represented by a cyclic chain of directed edges that trace its boundary in a counterclockwise order around the region. Because $rng(S)$ is a connected graph that spans S, any polygonal region is bounded by at most one edge not in $rng(S)$; this edge is in $ch(S) - rng(S)$. We call a polygonal region a *complete rng-polygon* if all its edges belong to $rng(S)$, and we call it an *incomplete rng-polygon* otherwise.

Obviously, rng-polygons are not as general as arbitrary polygonal regions because for each edge ab, except possibly for one, the lune of ab, $\lambda_{ab} = \{x \in \Re^2 : \max\{|ax|, |bx|\} < |ab|\}$, is free of points of S. We call pq a *diagonal* of a polygonal region if it lies entirely in the region. For each diagonal pq of an rng-polygon it must be that λ_{pq} contains at least one point of S. We further distinguish between the cases for which λ_{pq} contains points of S on both sides of pq and those for which it does not.

For a directed edge $\vec{pq}$ let $h_{\vec{pq}}$ be the set of points to the left of or on the directed line that passes through p and q in this order. Define the *half-lune* of $\vec{pq}$ as

$$\eta_{\vec{pq}} = \lambda_{pq} \cap h_{\vec{pq}}.$$

By definition, $\lambda_{pq} = \eta_{\vec{pq}} \cup \eta_{\vec{qp}}$, and we have $pq \in rng(S)$ if and only if $\eta_{\vec{pq}} \cap S = \eta_{\vec{qp}} \cap S = \emptyset$. We call pq a *2-edge* if both half-lunes contain points of S, and we call it a *1-edge* if only one half-lune contains points of S. For a 1-edge pq we say that the side where the half-lune contains points of S is *beyond* pq and that the other side is *beneath* pq. Note, for example, that if pq is a 1-edge bounding an incomplete rng-polygon R, then $pq \in ch(S)$ and therefore R is beyond pq. We will see later that 1-edges are useful in triangulating rng-polygons.

The first lemma of this section shows that when we triangulate an rng-polygon R, whether complete or incomplete, we can ignore all points outside R. More specifically, it shows that the type of any diagonal or edge of R remains unchanged when we remove all points of S that are not vertices of R.

REDUCTION LEMMA. *Let pq be a diagonal or edge of an rng-polygon R. If $\eta_{\vec{pq}}$ contains points of S, then it also contains vertices of R.*

Proof. If we assume $\eta_{\vec{pq}}$ contains points of S but no vertices of R, then it must intersect edges of R without containing their endpoints. Let yy' be the edge closest to p and q, and let x be a point in $\eta_{\vec{pq}} \cap S$. Since x is not a vertex of R, it must lie on the other side of yy' as seen from p and q. So $yy' \in rng(S) - ch(S)$, and therefore $\max\{|xy|, |xy'|\} \geq |yy'|$. Assume without loss of generality that $|xy| \geq |yy'|$. If y' lies outside or on the circle $(p, |pq|)$, we consider the convex quadrilateral $pyxy'$. Otherwise, y' lies outside or on $(q, |pq|)$, in which case we consider the convex quadrilateral $qyxy'$. But now we have $|xy| \geq |yy'|$ and either $|py'| > |px|$ or $|qy'| > |qx|$, a contradiction to the $\square$-Lemma in both cases. $\square$

Using the Reduction Lemma, we now address vertices visible from both endpoints of an edge. We need some notation. Two points x, y inside or on the boundary of a polygonal region are *visible* from each other if xy is contained in the region. The *distance* of a point x to an edge pq is defined as the infimum, over all points $z \in pq$, of $|xz|$. If $|pq| > \max\{|px|, |qx|\}$, then this distance is referred to as the *height* of the triangle pqx.

VISIBILITY LEMMA. *Let pq be a diagonal or edge of an rng-polygon R, and let x be a vertex of R that lies in $\eta_{\vec{pq}}$ and minimizes the distance from pq. Then x is visible from p and also from q.*

Proof. Consider the triangle pqx, let $x' \in pq$ be the point with minimum distance from x, and assume without loss of generality that x is not visible from q. Let yy' be an edge of R that intersects qx. The proof of the Reduction Lemma implies that at least one endpoint of yy' lies in $\eta_{\vec{pq}}$, say, $y \in \eta_{\vec{pq}}$. In addition, y and y' lie outside the triangle pqx because x is closest to pq (see Fig. 6). Hence yy' intersects xp, xq, and all edges

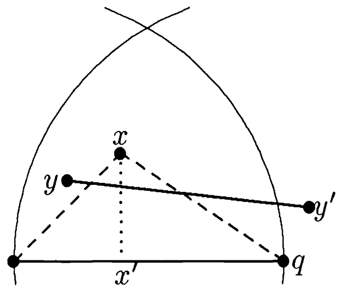


FIG. 6. *Quadrilateral $xyx'y'$ is convex because $x' \in pq$ and $y, y' \notin pqx$.*

xz with $z \in pq$. Thus $xyx'y'$ is a convex quadrilateral, and because of $|yx'| \geq |xx'|$ by the choice of x, we have $|yy'| > |y'x|$ from the $\square$-Lemma. By symmetry, if y' lies in $\eta_{\vec{pq}}$, we have $|yy'| > |xy|$, which implies $yy' \notin rng(S)$. This is a contradiction because $yy' \notin ch(S)$. Thus y' must lie outside $\eta_{\vec{pq}}$. If y' lies outside or on the circle $(p, |pq|)$, then $|py'| > |px|$ and therefore $|xy| < |yy'|$ by the $\square$-Lemma for $py'xy$. Symmetrically, we get $|xy| < |yy'|$ from the $\square$-Lemma for $qy'xy$ if y' lies outside or on the circle $(q, |pq|)$. Together with $|xy'| < |yy'|$ this contradicts $yy' \in rng(S)$. $\square$

We need one more elementary lemma.

CONTAINMENT LEMMA. *If $x \in \eta_{\vec{pq}}$, then $\eta_{\vec{xp}} \subseteq \lambda_{pq}$.*

Proof. Take a point $z \in \eta_{\vec{xp}}$ and consider the four points p, q, x, z. If $z \in pq$, there is nothing to prove. Otherwise, $pzqx$ or $pqzx$ is a convex quadrilateral (possibly with three of the four vertices collinear) or $z \in pqx$. In each case $|qz| < |pq|$ can be shown by using the $\square$-Lemma or the Δ-Lemma. This implies that $z \in \lambda_{pq}$. $\square$

The following lemma is of fundamental importance to the quadratic-time triangulation algorithm.

1-EDGE LEMMA. *Let pq be a 1-edge of an rng-polygon R, and let x be a vertex of R that lies in $\eta_{\vec{pq}}$ and minimizes the distance from pq. Then px is either an edge of R or a 1-edge with pqx beneath px, and the same is true for qx.*

Proof. We have $\eta_{x\vec{p}} \subseteq \lambda_{pq}$ by the Containment Lemma. The part of $\eta_{x\vec{p}}$ in $\eta_{q\vec{p}}$ contains no point of S because $\eta_{q\vec{p}} \cap S = \emptyset$ by assumption. Also, the part of $\eta_{x\vec{p}}$ in $\eta_{p\vec{q}}$ contains no point of S because a point $y \in \eta_{x\vec{p}} \cap \eta_{p\vec{q}}$ would be closer to pq than x is, as can be shown using the $\square$-Lemma for $px'yx$ (see Fig. 7). So px is an edge of R if $\eta_{p\vec{x}}$ contains no point of S either, and it is a 1-edge with triangle pqx on its beneath side otherwise. The argument for qx is symmetric. $\square$

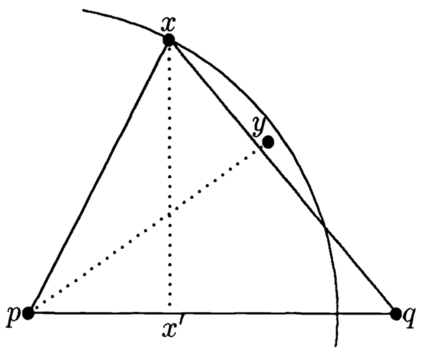


FIG. 7. *Vertex x is visible from p and from q, so pqx is empty. It follows that if $y \in \eta_{x\vec{p}} \cap \eta_{p\vec{q}}$ then $pqyx$ is a convex quadrilateral.*

5.1. Incomplete rng-polygons. The above lemmas are sufficient for efficiently triangulating an incomplete rng-polygon. As defined earlier, all edges of an incomplete rng-polygon R are rng-edges, except for one 1-edge, $pq \in ch(S) - rng(S)$, which has R on its beyond side. The algorithm below can triangulate more general incomplete rng-polygons, that is, it is not necessary that $pq \in ch(S)$, but it must be that pq is a 1-edge and R lies beyond pq.

Input. An incomplete rng-polygon R that lies beyond its 1-edge pq.

Output. A minmax length triangulation of R.

Algorithm.
1. Find a vertex x in λ_{pq} that minimizes the distance from pq.
2. Draw edges px and qx. This decomposes R into the triangle pqx and two possibly empty incomplete rng-polygons R_1 and R_2.
3. Recursively triangulate R_1 and R_2.

The correctness of this algorithm follows from the 1-Edge Lemma. Indeed, it implies that if R_1 is nonempty, then it lies beyond px, which is the only 1-edge of R_1. Similarly, R_2 lies beyond its 1-edge qx, provided that R_2 is nonempty. Thus the input invariant is maintained all the way through the recursion. This implies that the algorithm successfully triangulates. By the choice of point x, the edges px and qx are both shorter than pq. It follows that the diagonals are monotonically decreasing in length down a single branch of the recursion, and therefore all diagonals constructed by the algorithm are shorter than pq. A straightforward implementation of the algorithm takes time that is quadratic in the number of vertices of R.

Remark. Instead of choosing a vertex x that minimizes the distance to pq, step 1 of the algorithm could also choose other vertices as long as they are visible from p and

q and lie in their lune. An interesting choice among these vertices is the vertex y that minimizes $\max\{|yp|, |yq|\}$. As long as y is unique, which is the nondegenerate case, this choice leads to a triangulation of the polygon R that lexicographically minimizes the sorted vector of edge lengths. Another possible choice is the vertex z that minimizes $|zp| + |zq|$. This vertex is automatically visible from p and from q and might be useful in actual implementations because it is often considerably less expensive to compute the distance between two points than between a point and a line segment.

5.2. Lemma on polygon retriangulation. This subsection presents a technical lemma on retriangulating a polygonal region. It will find application in §§5.3 and 6 and is also of independent interest. In order to conveniently distinguish between boundary and nonboundary edges of a triangulation, we call a nonboundary edge a *diagonal*. Let X be a polygonal region, let $t(X)$ be a triangulation of X, and let xx' be a diagonal of X that is not in $t(X)$. We say that xx' *generates* $t(X)$ if it intersects every diagonal of $t(X)$. We give an algorithmic description of a particular triangulation of X, called the *fan-out triangulation* $f_x(X)$ with (*fan-out*) *center* x. The triangulation is illustrated in Fig. 8.

1. Connect x to all vertices of X that are visible from x. Call these vertices and also the two vertices connected to x by edges of X *neighbors* of x.

2. Two neighbors of x are said to be *adjacent* if they are consecutive in the angular order around x. Connect any two adjacent neighbors u, v of x unless uv is an edge of X.

3. Every edge uv created in step 2 decomposes X into two parts, and the part that does not contain x is called the *pocket* X_{uv} of uv. Assume that u is the endpoint of uv so that the other incident edge of the pocket uw is partially visible from x. Recursively construct the fan-out triangulation of X_{uv} with center v.

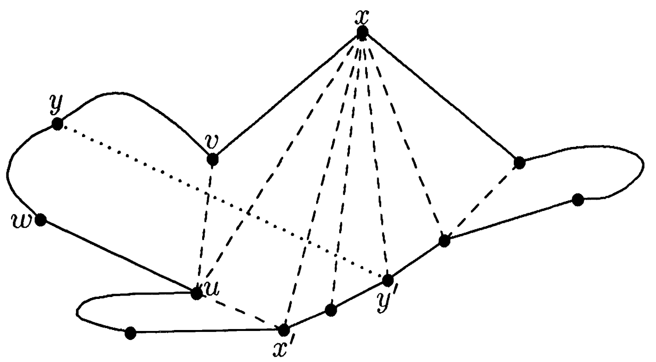


FIG. 8. *Polygonal region X is triangulated by fanning out from x, connecting adjacent neighbors of x, and recursing in the thus created pockets. The illustration of this process is schematic and ignores some of the inherent shape constraints for X.*

We introduce some terminology. Among the diagonals of $f_x(X)$ we distinguish between *fan-out* edges constructed in step 1 and *cut-off* edges constructed in step 2 of the above algorithm. Each call of the algorithm triangulates part of a pocket and recurses in each component (pocket) of the remainder. We call a pocket V a *child* of another pocket Z if $V \subset Z$ and V is maximal. The original polygonal region X is also called a pocket and forms the root of the tree defined by the child relation. This tree is exactly the recursion tree of the algorithm. Each pocket Z is associated with a fan-out center z. The maximum distance between z and any other vertex of Z is called the *width* of Z.

The lengths of the diagonals of $f_x(X)$ are constrained by the length of the longest edge of X, the length of the longest diagonal of $t(X)$, and the width of X. More specifically, we prove the following result.

FAN-OUT LEMMA. *Let X be a polygonal region, with δ_1 the length of its longest edge, let $t(X)$ be a triangulation of X, with δ_2 the length of its longest diagonal, let xx' be a generator of $t(X)$, and let δ_3 exceed the maximum distance of x from any vertex of X. Then $|ab| < \max\{\delta_1, \delta_2, \delta_3\}$ for every diagonal ab of $f_x(X)$.*

Proof. Note that the assertion follows if we prove that $\max\{\delta_1, \delta_2, \delta_3\}$ exceeds the width of every pocket Z created during the algorithm. To see this notice that the width of Z is an upper bound on the length of any fan-out edge emanating from the center of Z. Each cut-off edge uv that creates a child pocket V of Z is incident to the fan-out center of V, which implies that the width of V is an upper bound on its length.

The proof of the upper bound on the widths of all pockets proceeds inductively from the top to the bottom of the tree. The width of X is less than δ_3 by assumption, and it is therefore also less than $\max\{\delta_1, \delta_2, \delta_3\}$. For the inductive step consider a pocket Z and a child V of Z. We show that the bound on the width of Z is inherited by V, with some environmental influence from X and $t(X)$. Let z be the fan-out center of Z, let δ be the width of Z, let v be the fan-out center of V, let uv be the cut-off edge that creates V, and let w be the other vertex of V adjacent to u.

First, we prove $|uv| < \max\{\delta_1, \delta\}$. By the definition of a fan-out center, v lies inside the triangle uwz. The Δ-Lemma thus implies that $|uv| < \max\{|uw|, |uz|\}$, and we obtain the claimed inequality because $|uw| \le \delta_1$ and $|uz| \le \delta$. Second, we show that $\max\{\delta_2, \delta\}$ exceeds the maximum distance between v and any vertex of V other than u. Let $y \ne v, u$ be such a vertex, and let yy' be a diagonal of $t(X)$ that intersects xx'. Such a diagonal exists because xx' generates $t(X)$. It follows that yy' intersects uv and that therefore v lies inside the triangle $yy'z$. Using the Δ-Lemma we get $|yv| < \max\{|yy'|, |yz|\} \le \max\{\delta_2, \delta\}$ because $|yy'| \le \delta_2$ and $|yz| \le \delta$. The two bounds together imply that the width of V is less than $\max\{\delta_1, \delta_2, \delta\}$, and induction shows that it is less than $\max\{\delta_1, \delta_2, \delta_3\}$. □

In §6 we will need a result as given in the Fan-Out Lemma but restricted to the fan-out triangulation on one side of the generator. More specifically, we will need the following corollary whose proof is almost the same as the one of the Fan-Out Lemma.

FAN-OUT COROLLARY. *Suppose that W is a polygonal region, $t(W)$ is a triangulation of W, xx' is a generator of $t(W)$, and X is the part of W on one side of xx'. Let δ_1 be the length of the longest edge of X, let δ_2 be the length of the longest diagonal of $t(W)$, and let δ_3 exceed the maximum distance of x from any vertex of X. Then $|ab| < \max\{\delta_1, \delta_2, \delta_3\}$ for every diagonal ab of $f_x(X)$.*

Remark. The Fan-Out Lemma can also be formulated without the assumption of an initial triangulation. The condition on the diagonal xx' is now that each vertex of X must be visible from some point of xx'. The parameter δ_2 needs to be redefined as the maximum, over all vertices y of X, of the infimum, over all points a of xx' visible from y, of the distance between y and a.

5.3. Complete rng-polygons. It will be convenient to assume that no two diagonals and edges of the rng-polygon R are equally long. With this assumption we can show that every triangulation of R, and therefore also every minmax length triangulation, contains a 2-edge. To see this take the longest edge pq of a triangulation. It is not an edge of R because the third vertex of the incident triangle lies in its lune λ_{pq}. It is therefore a diagonal with incident triangles pqr and pqs, and we have $r, s \in \lambda_{pq}$ by maximality of pq. Since r and s lie on different sides of pq it follows that pq is a 2-edge.

We prove below that there is a minmax length triangulation $mlt(R)$ of R that contains only one 2-edge pq. By the argument above pq is the longest edge of $mlt(R)$. We call pq *expandable* if there are vertices r and s in λ_{pq}, on different sides of pq and both visible from p and q, so that $E = \{pr, qr, ps, qs\}$ is a set of rng- and 1-edges and the quadrilateral $prqs$ lies beneath the 1-edges in E. It should be clear that once we draw an expandable 2-edge, we can complete the triangulation by using the algorithm for incomplete rng-polygons (§5.1). The resulting triangulation uses no 2-edge other than pq, which is thus the longest edge of the triangulation.

We first present the algorithm, and then we prove its correctness by showing that every complete rng-polygon R has a minmax length triangulation that contains an expandable 2-edge. This, however, assumes that no two diagonals or edges of R have equal length. If this nondegeneracy constraint is not satisfied it is necessary to run the algorithm with a simulation of nondegeneracy; see [EdMü90]. The side effects of this simulation and how they can be undone will be discussed in §5.4.

Input. A complete rng-polygon R.

Output. A minmax length triangulation of R.

Algorithm.
1. Find the shortest expandable 2-edge pq, together with corresponding rng- and 1-edges pr, qr, ps, qs.
2. Triangulate the incomplete rng-polygons defined by pr, qr, ps, qs.

As mentioned in §5.1, step 2 takes time that is only quadratic in the number of vertices of R. In §6 we will see how step 1 can be implemented so that it runs in quadratic time too. We now formulate and prove the lemma that implies the correctness of the algorithm.

2-EDGE LEMMA. *Let R be a complete rng-polygon with no two diagonals or edges of the same length. Then there exists a minmax length triangulation $mlt(R)$ of R that contains an expandable 2-edge.*

Proof. We assume that there is no minmax length triangulation of R that contains an expandable 2-edge. A contradiction to this assumption will be derived by using a minmax length triangulation $t(R)$ that is defined as follows. Let pq be the longest edge of $t(R)$, and let pqr and pqs be the incident triangles. By the nondegeneracy assumption pq is the longest edge of *every* minmax length triangulation of R. Choose $t(R)$ so that the sum of heights of pqr and pqs (that is, the distance of r from pq plus the distance of s from pq) is a minimum. We will prove below that pq is expandable and that r and s are witnesses thereof, that is, that the quadrilateral $prqs$ lies beneath every 1-edge in $E = \{pr, qr, ps, qs\}$.

Case 1. Assume that $prqs$ lies beyond at least one 1-edge in E, say, beyond pr. Then we can retriangulate R on this side of pr by using the algorithm for incomplete rng-polygons. This algorithm removes edge pq, among others, and all new edges are shorter than pr, which itself is shorter than pq. This contradicts the assumption that $t(R)$ is a minmax length triangulation.

Case 2. Assume that one of the edges of E, say, pr, is a 2-edge, and assume without loss of generality that $r \in \eta_{\vec{pq}}$. Thus there is a nonempty set of vertices z of R contained in the half-lune $\eta_{\vec{rp}}$. By the Containment Lemma these vertices z lie in λ_{pq}, and by the Visibility Lemma a nonempty subset S' of the z are visible from both p and r.

If a vertex z is in S', then either $pz \cap rq \neq \emptyset$ or $rz \cap pq \neq \emptyset$; see Fig. 9. Let S'_p be the subset of vertices z of the first kind, and let S'_r be the subset of vertices of the second kind. If $S'_p \neq \emptyset$, choose $x \in S'_p$ so that the number of edges of $t(R)$ that intersect px, is a minimum. Next, remove all edges from $t(R)$ that intersect px, and denote by X the polygonal region thus generated. If, on the other hand, $S'_p = \emptyset$, then choose $x \in S'_r \neq \emptyset$ so that the number of edges in $t(R)$ that intersect rx is a minimum, again remove all edges from $t(R)$ that intersect rx, and denote the resulting polygonal region by X. For convenient reference we set $x' = p$ in the first case and $x' = r$ in the second. In either case we construct a retriangulation $f_x(X)$ of X by fanning out from x, as described in §5.2.

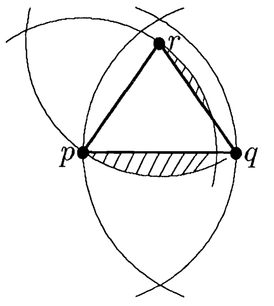


FIG. 9. *Points z lie in the interior of $\eta_{r\vec{p}} - pqr$, which consists of one or two connected components, depending on whether or not the angle at r in triangle pqr is nonacute.*

We will show below that the new triangulation of R has properties that contradict the assumptions of case 2. Most important, the Fan-Out Lemma of §5.2, together with a few claims that we are about to prove, imply that the edges of $f_x(X)$ do not exceed pq in length.

CLAIM 1. *Except for x, all vertices of X lie outside the half-lune $\eta_{r\vec{p}}$.*

Proof of Claim 1. Let $y_1y_2, y_3y_4, \ldots, y_{m-1}y_m$ be the edges, sorted from x' to x, that are removed from $t(R)$ when X is constructed. Suppose that the claim is not true. Then there is a smallest index $j, 3 \leq j \leq m-1$, so that one endpoint of y_jy_{j+1}, say, y_{j+1}, is in $\eta_{r\vec{p}}$. Consider the polygonal region X_j of $t(R)$ that is created by removing the edges $y_1y_2, y_3y_4, \ldots, y_{j-2}y_{j-1}$ from $t(R)$. Since y_{j+1} is the only vertex of X_j that lies in $\eta_{r\vec{p}}$, it is visible from p *and* from r, inside X_j. But this means that $y_{j+1}x'$ intersects fewer edges of $t(R)$ than does xx'. This contradicts the choice of x and completes the proof of Claim 1.

CLAIM 2. *For each vertex y of X we have $|xy| < |pq|$.*

Proof of Claim 2. Clearly, both px and rx are shorter than pq. So let y be any vertex different from p, r, x, and let yy' be an edge of $t(R)$ that intersects $x'x$. Because of Claim 1, x is visible within X from p and also from r, so $pyxy'$ and $ryxy'$ are convex quadrilaterals. Since y' lies outside $\eta_{r\vec{p}}$ it cannot lie inside both of the circles $(p, |pr|)$ and $(r, |pr|)$. If y' lies inside $(r, |pr|)$, then $|py'| > |px|$ which implies $|yy'| > |xy|$ by the $\square$-Lemma for $pyxy'$. Otherwise, we have $|ry'| > |rx|$ which implies $|yy'| > |xy|$ by the $\square$-Lemma for $ryxy'$. This concludes the proof of Claim 2 because yy' is an edge of $t(R)$ and is therefore no longer than pq.

Claim 2 and the Fan-Out Lemma imply that all diagonals of $f_x(X)$ are shorter than pq. In the case for which $pq \cap rx \neq \emptyset$ we now have a contradiction because the retriangulating process of X eliminates pq and all edges of the resulting new triangulation of R are shorter than pq. In the case for which $rq \cap px \neq \emptyset$ the new triangulation still includes pq. We will show below that the height of the new triangle incident to pq is smaller than the height of pqr and thus arrive at a contradiction.

So assume $rq \cap px \neq \emptyset$; in this case pq is an edge of the boundary of X and p is visible from x. If q is also visible from x, then the new triangle incident to pq is pqx with height $|xx'|$, where $x' \in pq$ minimizes the distance to x. Analogously, define $r' \in pq$ that minimizes the distance to r. Since $|pr| > |px|$, we have $|rr'| > |xr'|$ by the $\square$-Lemma for $prxr'$. Together with $|xr'| \geq |xx'|$, this implies $|rr'| > |xx'|$. If q is not visible from x, then pq belongs to the pocket X_{uv} defined by a cut-off edge uv. We have $u = p$, $w = q$, and the center v of X_{uv} lies inside pqx. So again, either pqv is a triangle, and its height is less than that of pqx and therefore that of pqr, or q is not visible from v, in which case the argument can be repeated. Eventually, we arrive at a triangle incident to pq whose height is less than that of pqr. $\square$

Remark. Recall that the assertion of the 2-Edge Lemma is made under the condition that no two diagonals or edges of the complete rng-polygon R are equally long. Indeed, the assertion is false without this condition. Take, for example, two equilateral triangles abc and abd and move d slightly towards the common edge ab. For $S = \{a, b, c, d\}$ we have that $rng(S) = \{ac, cb, bd, da\}$, ab is a 1-edge, and cd is a 2-edge. So $acbd$ is a complete rng-polygon. There is only one minmax length triangulation of $acbd$, namely, the one obtained by drawing the diagonal ab. But ab is not a 2-edge.

5.4. Undoing the simulated perturbation. For every finite point set S in $\Re^2$ there is an arbitrarily small perturbation S' so that S' satisfies convenient nondegeneracy assumptions (see [EdMü90]). For a point $p \in S$ we denote its perturbed version by p'. In the case of relative neighborhood graphs and minmax length triangulations this means that no two pairs of points in S' define the same distance. Because the perturbation is arbitrarily small, the nondegenerate properties of S are maintained, that is, for four not necessarily distinct points $p, q, r, s \in S$ with $|pq| < |rs|$ we have $|p'q'| < |r's'|$.

Let us consider the effect of the perturbation on the computation of a minmax length triangulation. Clearly, if $p'q' \in rng(S')$, then $pq \in rng(S)$, but not vice versa. The fact that in the perturbed setting the relative neighborhood graph has potentially fewer edges than in the unperturbed setting does not adversely influence the triangulation algorithm since $rng(S')$ is still connected and spans S'. When the edges of $ch(S')$ are added and the polygonal regions defined by $ch(S') \cup rng(S')$ are triangulated, it can happen that triangles $a'b'c'$ are constructed whose unperturbed counterparts abc are flat, that is, a, b, c are collinear. Although this is not a problem for the algorithm, it is somewhat distressing when this triangulation is interpreted as a triangulation of S. The remainder of this section shows how to remedy this deficiency.

Let $t(S')$ be a minmax length triangulation of S', and consider its unperturbed version $t(S)$, that is, $pq \in t(S)$ if and only if $p'q' \in t(S')$. A longest edge of $t(S)$ is no longer than a longest edge of any minmax length triangulation $mlt(S)$ of S since $mlt(S')$, the perturbed version of $mlt(S)$, is a valid triangulation of S' and would otherwise contradict that $t(S')$ is a minmax length triangulation of S'. The reverse is also true, namely, that a longest edge of $t(S)$ is no shorter than a longest edge of $mlt(S)$. We show this by converting $t(S)$ into a minmax length triangulation of S.

Consider the dual graph $t^*(S')$ of $t(S')$, and call a node $a'b'c'$ *flat* if a, b, c, are collinear. Determine the connected components of the subgraph of $t^*(S')$ induced by the set of all flat nodes. Each component corresponds to a collection of collinear points in S interconnected by flat triangles; see Fig. 10. Carry out the following steps for one component at a time. Remove all edges of the flat triangles of the component, sort the corresponding points along the supporting line, and add edges connecting points that are adjacent in the sorted order. This produces regions bounded by more than three edges, as shown in Fig. 10. All vertices x of such a region are collinear, except for one vertex y that is connected to the first and last of the vertices x. Triangulate this region by connecting y to all other vertices x. By the Δ-Lemma the newly introduced edges are no longer than the longer of the two original edges incident to y.

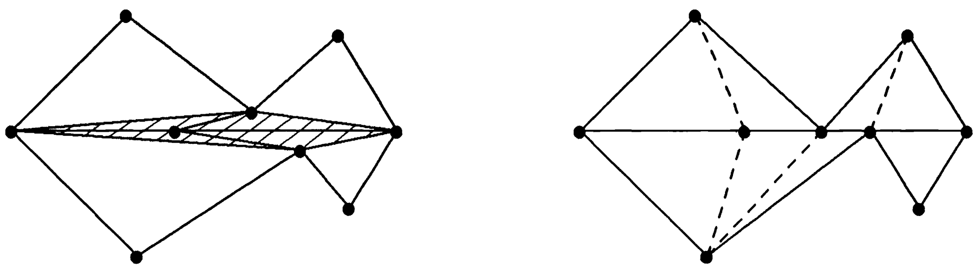


FIG. 10. *The five points in the middle of the left triangulation are the perturbed versions of five collinear points in the right triangulation.*

6. Finding the shortest expandable 2-edge. This section shows how the first step of the algorithm for triangulating a complete rng-polygon R can be made to run in time $O(n^2)$, where n is the number of vertices of R. As in §5.3, we assume that no two diagonals or edges of R are equally long; so the shortest expandable 2-edge is unique. For convenience we also assume that no three vertices of R are collinear.

Input. A complete rng-polygon R.

Output. The shortest expandable 2-edge of R.

Algorithm.
1. Determine the type of each diagonal pq of R.
2. For each 2-edge pq find vertices p', p'', q', q'' that minimize the counterclockwise angles $\angle p'pq, \angle qpp'', \angle q'qp, \angle pqq''$ contingent on pp', pp'', qq', qq'' being rng-edges or 1-edges with pq on their beneath sides (see Fig. 11).
3. Return the shortest 2-edge pq for which pp', qq', pp'', qq'' are such that $p' = q''$ or $pp' \cap qq'' \neq \emptyset$ and that $p'' = q'$ or $pp'' \cap qq' \neq \emptyset$.

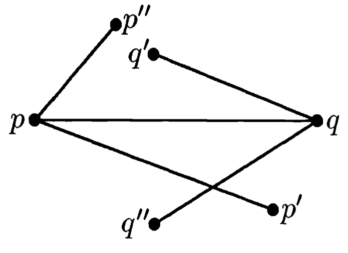


FIG. 11. *By the choice of p' the counterclockwise angle $\angle p'pq$ contains no 1-edge with pq on its beneath side. Symmetric statements hold for p'', q', and q''.*

Below we give the algorithmic details of the above steps.

Step 1. *Classifying diagonals*. For each vertex p of R we compute all incident diagonals pq and their angular order around p. Furthermore, we determine whether or not the half-lune $\eta_{p\bar{q}}$ contains any vertex of R. Recall that by the Visibility Lemma $\eta_{p\bar{q}}$ contains a vertex visible from p if it contains a vertex of R at all. We can thus base the decision of whether or not $\eta_{p\bar{q}}$ is empty of vertices solely on the vertices visible from p. As defined earlier, pq is a 2-edge if both half-lunes of pq contain vertices of R. Otherwise, pq is a 1-edge and its beyond side is where the half-lune contains vertices of R. We now show that the computation for p can be done in time $O(n)$. It follows that $O(n^2)$ time suffices for Step 1.

Computing the sorted sequence of diagonals $pp_1, pp_2, \dots, pp_m$ incident to p is a standard operation for simple polygons and can be done in time $O(n)$; see, e.g., [ElAv81], [JoSi87], [Lee83]. Let pp_0 and pp_{m+1} be the two edges of R incident to p, and assume that $p_0, p_1, p_2, \dots, p_m, p_{m+1}$ is in a counterclockwise order around p. To determine whether there is a vertex of R in the half-lune $\eta_{p\bar{p}_i}$ for $1 \leq i \leq m$, we scan the list $p_0, p_1, \dots, p_{m+1}$ once, from smallest index to largest. During the scan we maintain a stack of diagonals pp_l whose half-lunes $\eta_{p\bar{p}_l}$ are not yet found to contain any vertex of R. Before pushing pp_i onto the stack, we remove all diagonals pp_l whose half-lunes contain p_i. Using a straightforward extension of the Containment Lemma, we can show that the order of processing implies that the edges whose half-lunes contain p_i lie on top of the ones whose half-lunes do not contain p_i. Thus the former can be removed by simply repeatedly popping the topmost diagonal. When the scan is complete, the stack contains exactly all diagonals pp_l whose half-lunes contain no vertex of R. Since a diagonal can be pushed and popped only once each, the entire process takes constant time per diagonal.

Step 2. *Finding rng- and 1-edges*. For each vertex p we scan $pp_1, pp_2, \dots, pp_m$ in this order. In the process we keep track of the most recent rng-edge or 1-edge $p\bar{p}$ whose beneath side is in the direction of the scan. Initially, $p\bar{p} = pp_0$. When a 2-edge pq is encountered, then $p\bar{p}$ is the edge pp' that belongs to pq. A symmetric scan is carried out to find the edge pp'' that belongs to pq. The total time for all vertices p of R is clearly $O(n^2)$.

Step 3. *Returning the solution*. Step 3 is computationally trivial. It takes time $O(n^2)$ since constant time suffices to test whether or not pp', pp'', qq', qq'' satisfy the conditions of Step 3. However, it is not trivial to see that the edge pq returned in Step 3 is also the shortest expandable 2-edge. First, note that the shortest expandable 2-edge is no shorter than pq. This is because all 2-edges shorter than pq fail the test of Step 3. The following straightforward topological lemma implies that these 2-edges are not expandable.

CROSSING LEMMA. *Let $v_1, v_2, \dots, v_n$ be the sequence of vertices of a simple polygon, and let v_1v_i and v_jv_n be two diagonals. Then $v_1v_i \cap v_jv_n \neq \emptyset$ if and only if $j < i$.*

Proof. The edge v_jv_n decomposes the polygon into two disjoint polygons with vertex sequences $v_1, v_2, \dots, v_j, v_n$ and $v_j, v_{j+1}, \dots, v_n$. If $j < i$, then neither of the two polygons has v_1 *and* v_i on its boundary. It follows that v_1v_i crosses from one polygon into the other, and because v_1v_i is a diagonal, this is only possible by crossing v_jv_n. To prove the other direction we assume $v_1v_i \cap v_jv_n \neq \emptyset$ and observe that v_1 and v_i belong to different polygons because there is no way that v_1v_i can enter the second polygon and leave it again. Thus $j < i$. □

So it remains to show that the edge pq computed in Step 3 is indeed expandable.

EXPANDABILITY LEMMA. *The shortest 2-edge pq of R that satisfies the conditions of Step 3 is also expandable.*

Proof. We show below that R can be triangulated on both sides of pq by using only edges shorter than pq. If we now assume that pq is not expandable, we get a contradiction to the 2-Edge Lemma because pq is the longest edge of the triangulation and all expandable 2-edges are longer than pq.

We describe how to triangulate the part of R to the right of $\vec{pq}$; the other part is symmetric.

Case 1. $p' = q''$. Assume that $|qq''| > |pp'|$. Then $|qq''| < |pq|$, for otherwise $p \in \eta_{q\vec{q''}q}$ and qq'' would be neither an rng-edge nor a 1-edge with pq on its beneath side. If we apply the triangulation algorithm for incomplete rng-polygons (§5.1), once for pp' and once for qq'', we obtain a triangulation with the desired properties.

Case 2. $pp' \cap qq'' \neq \emptyset$. In this case pp' and qq'' are 1-edges. Because pp' and qq'' intersect, it must be that p' is closer to q than to p or that q'' is closer to p than to q. Assume without loss of generality that $|q''p| < |q''q|$. As in Case 1, we also have $|q''q| < |pq|$, but note that we do not necessarily have $|pp'| < |pq|$.

We now describe the triangulation process. It takes three steps, illustrated in Figs. 12 and 13.

1. Construct the triangulation $t_{qq''}$ of R beyond qq'' by using the algorithm for incomplete rng-polygons (see Fig. 12).

2. Find the subset V of vertices of R that lie inside the triangle pqq'', and compute the convex hull C of $V \cup \{p, q''\}$. Add the edges of C that are diagonals of R to the triangulation, and connect q to all vertices of C (see Fig. 12).

3. Step 2 creates untriangulated pockets Y_{uv}, one for each edge uv of C that is a diagonal of R. Assume that u precedes v on the clockwise path from p to q'' on the boundary of C. The pocket Y_{uv} is triangulated as follows.

 3.1 Set $u_L := v$ if uv is a 1-edge and pq lies on the beneath side of uv. Otherwise, find a vertex u_L so that $|uu_L| < |pq|$, uu_L is a 1-edge, pq lies beneath uu_L, and uu_L does not intersect C. (The existence of such a vertex u_L will be established shortly.)

 3.2 Construct the triangulation t_{uu_L} of R beyond uu_L, again using the algorithm for incomplete rng-polygons, but retain only the triangles that lie completely inside the pocket Y_{uv}. Let X_{uv} denote the untriangulated part of Y_{uv}.

 3.3 Construct the fan-out triangulation $f_v(X_{uv})$.

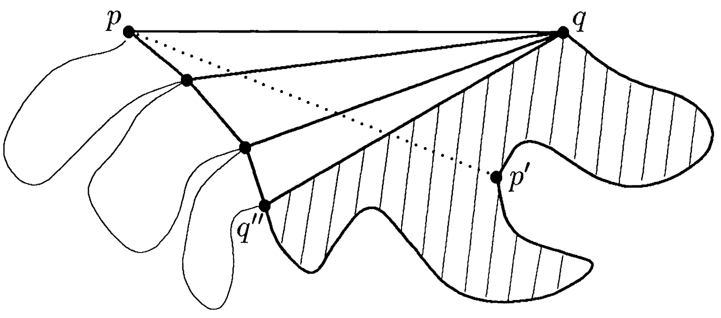


FIG. 12. *The shaded portion represents the triangulation beyond qq''; it forms part of the final triangulation.*

The remainder of the proof establishes that all diagonals of the thus constructed triangulation are shorter than pq. This is indeed obvious for $t_{qq''}$ as constructed in step 1. We now prove an easy extension of the Δ-Lemma that implies that all edges created in step 2 are shorter than pq.

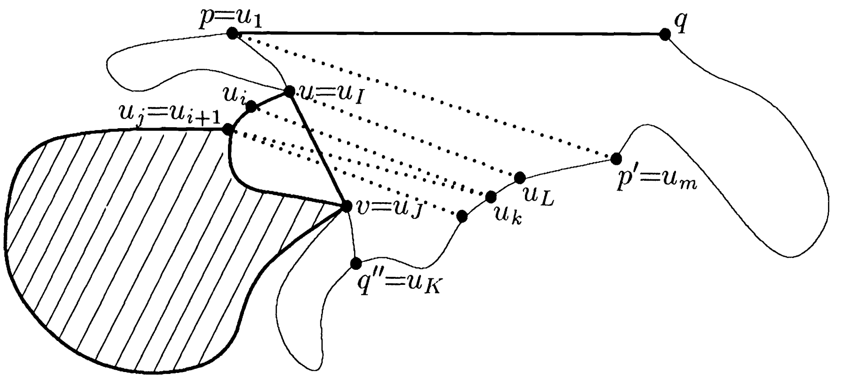


FIG. 13. *The shaded portion of the pocket Y_{uv} represents the part of the triangulation t_{uu_L} beyond uu_L that is retained for the final triangulation. The remaining portion is triangulated by fanning out from v.*

CLAIM 1. *Let abc be a triangle, and let d, e be two points inside abc. Then* $|de| < \max\{|ab|, |ac|, |bc|\}$.

Proof of Claim 1. Assume without loss of generality that e lies inside abd. The Δ-Lemma for abd implies that $|de| < \max\{|ad|, |bd|\}$, and the same lemma for abc implies that $\max\{|ad|, |bd|\} < \max\{|ab|, |ac|, |bc|\}$. This completes the proof of Claim 1.

If $uu_L = uv$, then $|uu_L| < |pq|$ which implies that all edges of t_{uu_L}, as constructed in step 3.2, are shorter than pq. In this case the proof is complete since $X_{uv} = \emptyset$ and no edges are added to Y_{uv} in step 3.3. For the remainder of the proof we thus assume that $u_L \neq v$, which is the case only if $\eta_{\vec{u}\vec{v}}$ contains at least one vertex of R. We show that a vertex u_L satisfying the conditions of step 3.1 indeed exists, and that all edges of the fan-out triangulation $f_v(X_{uv})$ are shorter than pq. Assume that the sequence of vertices of the part of R beyond pp' is $p = u_1, u_2, \dots, q'' = u_K, \dots, u_m = p'$ (see Fig. 13).

CLAIM 2. *There exists a* 1*-edge uu_L that satisfies the conditions of step* 3.1.

Proof of Claim 2. Construct a triangulation $t_{pp'}$ of R beyond pp' by using the algorithm for incomplete rng-polygons. This triangulation contains at least one edge uu_l disjoint from C. The main invariant of the algorithm (described in §5.1) implies that uu_l is a 1-edge and that pq lies on its beneath side. If $|uu_l| < |pq|$, then u_l satisfies the conditions for u_L and we are done.

So assume that $|uu_l| > |pq|$. As we showed for the Containment Lemma, we can show that the part of $\eta_{\vec{u}\vec{v}}$ to the left of $\vec{u}\vec{u}_l$ is contained in $\eta_{u\vec{u}_l}$ and thus contains no vertex of R. It follows that the vertices in $\eta_{\vec{u}\vec{v}}$ must be among $u_{K+1}, u_{K+2}, \dots, u_{l-1}$. By the Visibility Lemma at least one of these vertices is visible from u. Let U be the subset of vertices that are visible from u (including the ones outside $\eta_{\vec{u}\vec{v}}$), and let $u_L \in U$ minimize the distance to u. We have $|uu_L| < |uv| < |uu_l|$ and, as above, the part of $\eta_{u\vec{u}_L}$ to the left of $\vec{u}\vec{u}_l$ is contained in $\eta_{u\vec{u}_l}$. Therefore, this part contains no vertex of R. The part of $\eta_{u\vec{u}_L}$ to the right of $\vec{u}\vec{u}_l$ contains no vertex of R by the choice of u_L. It follows that uu_L is a diagonal that satisfies the conditions of step 3.1, which completes the proof of Claim 2.

We now show two easy facts about t_{uu_L} before examining the edges constructed by step 3.3.

CLAIM 3. *If $u_iu_ju_k$, with $i < j < k$, is a triangle of t_{uu_L}, then u_iu_k is its longest edge.*

Proof of Claim 3. The first triangle constructed is $u_Iu_lu_L$, for some $I < l < L$, and its longest edge is u_Iu_L because $u_l \in \lambda_{u_Iu_L}$. The general assertion follows by induction, which completes the proof of Claim 3.

CLAIM 4. *The edges of t_{uu_L} that intersect uv, sorted from u to v, are monotonically decreasing in length.*

Proof of Claim 4. If $u_iu_ju_k$, with $i < j < k$, intersects uv, $u = u_I$ and $v = u_J$, then either $I \leq i < j = i+1 \leq J < k$ or $I < i < J \leq j < k$ (see Fig. 13). In both cases u_iu_k intersects uv closer to u than the other intersecting edge, u_ju_k or u_iu_j. By Claim 3, u_iu_k is longer than both, which implies the assertion.

Note that if we delete edges from t_{uu_L} that intersect uv, then we get a polygonal region, say, W_{uv}, of which X_{uv} is the part on one side of uv. We can thus interpret uv as a generator of t_{uu_L} restricted to W_{uv}. Since the edges of X_{uv} and t_{uu_L} are shorter than $|pq|$, we need to show only that all vertices of X_{uv} are closer to v than $|pq|$, and the rest follows from the Fan-Out Corollary. Indeed, we prove a stronger bound on the maximum distance from v to a vertex of X_{uv}.

CLAIM 5. *For each vertex x of X_{uv} we have that $|vx| \leq |vu|$.*

Proof of Claim 5. Consider the vertices of X_{uv} in turn from $u = u_I$ to $v = u_J$, and assume inductively that $|vu_i| \leq |vu|$ for all $I \leq i < j$. Consider u_j and the triangle $u_{j-1}u_ju_k$ in t_{uu_L}. By Claim 4, we have that $|u_{j-1}u_k| > |u_ju_k|$. If $u_{j-1}u_jvu_k$ is a convex quadrilateral, then the $\square$-Lemma implies that $|vu_{j-1}| > |vu_j|$, as desired. Otherwise, u_j is contained in vu_ku_{j-1} and therefore also in vuu_{j-1}. The Δ-Lemma implies $|vu_j| < \max\{|vu|, |vu_{j-1}|\}$, which completes the proof of Claim 5.

This also completes the proof of the lemma. $\square$

The following theorem summarizes the algorithmic implications of all of this.

MINMAX LENGTH THEOREM. *A minmax length triangulation of a set of n points in $\Re^2$ can be constructed in time $O(n^2)$.*

The algorithm that constructs a minmax length triangulation in the claimed amount of time is a combination of the algorithms given in §§3, 5.1, 5.3, and 6. Its correctness has been demonstrated in §§4, 5.3, and 6.

7. Arbitrary normed metrics.

An open convex region $D \subseteq \Re^2$ that is symmetric with respect to the origin can be used to impose a *norm* on $\Re^2$: for a point $x \in \Re^2$ define $\|x\| = \|x\|_D = \alpha$ if x lies on the boundary of $\alpha D = \{\alpha y \in \Re^2 : y \in D\}$. The norm can then be used to impose a (*normed*) *metric* on $\Re^2$: for two points $x, y \in \Re^2$ define $|xy| = |xy|_D = \|y - x\|_D$. D is the *unit disk* of the metric and the boundary of D is its *unit circle*. Notice that the three requirements for a metric are indeed satisfied. First, $|ab| = 0$ if and only if $a = b$ because $\|x\| = 0$ if and only if x is the origin. Second, $|ab| = |ba|$ because D is centrally symmetric and therefore $\|x\| = \|-x\|$. Third, the triangle inequality $|ac| \leq |ab| + |bc|$ follows from the convexity of D. Examples of normed metrics are the l_p-metrics, for $1 \leq p \leq \infty$, and the so-called A-metric discussed in [WWW85] for its applications to VLSI.

In this section we assume that the triangle inequality is strict unless a, b, c lie on a line in this order. This is the case if and only if the defining convex region D is strictly convex, that is, no line intersects the boundary of D in more than two points. This assumption is convenient and, in fact, without loss of generality since every convex but not strictly convex region D' can be approximated arbitrarily closely by a strictly convex region D. Computationally, this approximation can be simulated by defining

$$\|x\|_D = \|x\|_{D'} + \epsilon\|x\|_2,$$

where $\|x\|_2$ is the Euclidean or l_2-norm and ϵ is an arbitrarily small but positive real number. Clearly, if ϵ is sufficiently small then a minmax length triangulation under D is also a minmax length triangulation under D'.

In the remainder of this section we point out where the developments in §§2–6 need to be adjusted when the Euclidean metric is replaced by an arbitrary normed metric. Most important, the graphs defined in §2 can be extended in a natural way. More specifically, the definition of $ch(S)$ remains unchanged since it makes no reference to any distance notion. If we now stipulate that "circle" means a homothetic copy of the unit circle as defined above and "$|ab|$" means the distance under the normed metric defined by D, then the definitions of $mlt(S)$, $dt(S)$, $rng(S)$, and $mst(S)$ can be taken verbatim. The minimum spanning tree $mst(S)$ is connected and spans S, and the Delaunay triangulation $dt(S)$ is plane because any two circles intersect in at most two points. Since we still have $mst(S) \subseteq rng(S) \subseteq dt(S)$, we conclude that all three graphs are connected and plane and that they span S. We remark that these three graphs are not necessarily plane if D is not strictly convex.

As mentioned in §1, the developments in §2–6 are all based on a small number of basic facts, namely, the distance relations expressed by the $\Box$-Lemma and the $\triangle$-Lemma, the convexity of the lune of an edge, and the straightness of the bisector of two points. The $\Box$-Lemma and the $\triangle$-Lemma are direct consequences of the triangle inequality and hold in the stated form (with strict inequality) for arbitrary normed metrics as long as D is strictly convex. The lune of two points is clearly convex since it is the intersection of two homothetic copies of D. Unfortunately, the bisector of two points $p \neq q$, $\ell_{pq} = \{x : |xp| = |xq|\}$, is not necessarily straight. Nevertheless, ℓ_{pq} is still a simple curve that divides $\Re^2$ into two unbounded regions, called *half-planes*, one containing p and the other containing q. The two half-planes are star-shaped with respect to p and q, that is, any line through p or q intersects ℓ_{pq} in at most one point. In addition, ℓ_{pq} is symmetric with respect to $\frac{p+q}{2}$ because D is centrally symmetric.

There is only one place where the straightness of the bisector is used in a substantial way, and that is in the proof of Fact 3 in §4. We restate this fact and show how to prove it without using the straightness of the bisector. We suggest that the reader go back to §4 and review Facts 1 and 2. Recall, in particular, that bd (respectively, $b'd'$) is said to be *switchable* if ac (respectively, $a'c'$) is no longer than the longest edge of the triangulation $t(S)$.

FACT 3. *It is not possible that both* bd *and* $b'd'$ *are nonswitchable.*

Proof. As established in Fact 2, if bd is nonswitchable, then a and d are contained in the open half-plane defined by ℓ_{pq} that contains q. Symmetrically, if $b'd'$ is nonswitchable, then, a' and d' are contained in the other open half-plane. Unlike in the Euclidean case, it is possible that ad and $a'd'$ intersect ℓ_{pq}. It is thus also possible that ad precedes $a'd'$ in the order of edges sorted from p to q by their intersections with pq (see Fig. 14). Below we will argue that if this is the case, then ad (and symmetrically $a'd'$) is switchable. In particular, we show that $|ad| > |ap|$, which, together with $|ap| \geq |ac|$ from Fact 2, implies that ad is switchable.

One characteristic of the described situation is that ad intersects ℓ_{pq} in at least one point inside the lune of pq. Let x be such an intersection point closest to a. If $pq \cap dx \neq \emptyset$ then $pdqx$ is a convex quadrilateral with $|pd| \geq |pq|$ by construction. The $\Box$-Lemma thus implies that $|dx| > |qx| = |px|$. It follows that $|ad| = |ax| + |dx| > |ax| + |px| > |ap|$. On the other hand, if $pq \cap dx = \emptyset$ then consider the point $y = ad \cap pq$ and note that $|py| \leq |qy|$. We derive $|dy| > |py|$ from $|py| + |dy| > |pd| \geq |pq| \geq 2|py|$. Therefore, $|ad| = |ay| + |dy| > |ay| + |py| > |ap|$, as desired. $\Box$

All other steps of the proof of the Subgraph Theorem go through unchanged for arbitrary normed metrics. We thus obtain the following generalization.

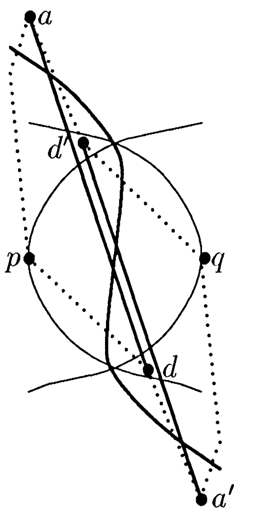


FIG. 14. *Although a and d lie on q's side of the bisector and a' and d' lie on p's side, ad intersects pq closer to p than a'd' does. This is not possible if the bisector is a line, as for the Euclidean metric, see Fig. 4.*

GENERAL SUBGRAPH THEOREM. *Let S be a finite point set in $\Re^2$ equipped with a normed metric with strictly convex unit disk. Then S has a minmax length triangulation $mlt(S)$ so that $rng(S) \subseteq mlt(S)$.*

So the algorithm for computing a minmax length triangulation is clear—it is the same as for the Euclidean metric, only the length of edges is now measured in terms of a normed metric that is possibly different from the Euclidean metric. We assume that the length of an edge in this metric can be computed in constant time. A careful reexamination of §§5 and 6 shows that the specialized polygon triangulation algorithm works also in the context of arbitrary normed metrics. We remark, however, that it includes the distance computation between a point and a line segment. Although it is certainly reasonable to assume that this can also be done in constant time, the observation in the remark at the end of §5.1 can be used to avoid this computation. We thus have the following algorithmic result, which generalizes the MinMax Length Theorem of §6.

GENERAL MINMAX LENGTH THEOREM. *Let S be a set of n points in $\Re^2$ equipped with a normed metric with strictly convex unit disk. Given the relative neighborhood graph, a minmax length triangulation of S can be constructed in time $O(n^2)$.*

The algorithmic result extends to arbitrary normed metrics. As mentioned above, a norm with nonstrictly convex unit disk can be simulated by one with strictly convex unit disk. It follows that the quadratic-time bound also holds for arbitrary normed metrics. The result stated in the General MinMax Length Theorem raises the question of how fast $rng(S)$ can be constructed. The trivial algorithm tests all $\binom{n}{2}$ edges, each in time $O(n)$, and therefore takes time $O(n^3)$. Faster algorithms are known for the l_p-metrics for which $O(n \log n)$ time suffices (see [JKY90] and [Lee85]).

8. Discussion. The main contribution of this paper is the first polynomial-time algorithm for computing a minmax length triangulation of a set S of n points in $\Re^2$. Given the relative neighborhood graph of S, the algorithm takes time $O(n^2)$. The algorithm works for arbitrary normed metrics. The polynomial time bound follows because the relative neighborhood graph of S can be found in polynomial time. The question of whether or not a minmax length triangulation can be computed in less than quadratic time remains.

The results of this paper are an outgrowth of our general efforts to understand triangulations that optimize length criteria. There are, however, still many related problems whose complexities remain open. These include the problem of minimizing the entire vector of edge lengths, the minimum length triangulation problem, and the maxmin length triangulation problem.

Acknowledgment. The authors thank an anonymous referee for suggestions on the organization of this paper.

REFERENCES

[ACNS82] M. AJTAI, V. CHVÁTAL, M. M. NEWBORN, AND E. SZEMERÉDI, *Crossing-free subgraphs*, Ann. Discrete Math., 12 (1982), pp. 9–12.

[BrZl70] J. BRAMBLE AND M. ZLÁNAK, *Triangular elements in the finite element method*, Math. Comput., 24 (1970), pp. 809–820.

[Del34] B. DELAUNAY, *Sur la sphère vide*, Izv. Akad. Nauk SSSR, Otdel. Mat. Estest. Nauk, 7 (1934), pp. 793–800.

[Edel87] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[EdMü90] H. EDELSBRUNNER AND E. P. MÜCKE, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graphics, 9 (1990), pp. 66–104.

[ETW92] H. EDELSBRUNNER, T. S. TAN, AND R. WAUPOTITSCH, *An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 994–1008.

[ElAv81] H. ELGINDY AND D. AVIS, *A linear algorithm for computing the visibility polygon from a point*, J. Algorithms, 2 (1981), pp. 186–197.

[Gilb79] P. D. GILBERT, *New results in planar triangulations*, M. Sc. thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1979.

[GuSt85] L. J. GUIBAS AND J. STOLFI, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graphics, 4 (1985), pp. 74–123.

[JKY90] J. W. JAROMCZYK, M. KOWALUK, AND F. F. YAO, *An optimal algorithm for constructing β-skeletons in L_p metric*, SIAM J. Comput., submitted.

[JoSi87] B. JOE AND R. B. SIMPSON, *Corrections to Lee's visibility polygon algorithm*, BIT, 27 (1987), pp. 458–473.

[Klin80] G. T. KLINCSEK, *Minimal triangulations of polygonal domains*, Ann. Discrete Math., 9 (1980), pp. 121–123.

[Lank69] P. M. LANKFORD, *Regionalization: Theory and alternative algorithms*, Geographical Anal., 1 (1969), pp. 196–212.

[Laws77] C. L. LAWSON, *Software for C^1 surface interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, San Diego, CA, 1977, pp. 161–194.

[Lee83] D. T. LEE, *Visibility of a simple polygon*, Comput. Vision, Graphics, and Image Process., 22 (1983), pp. 207–221.

[Lee85] ———, *Relative neighborhood graphs in the L_1-metric*, Pattern Recognition, 18 (1985), pp. 327–332.

[Ling87] A. LINGAS, *A new heuristic for minimum weight triangulation*, SIAM J. Algebraic Discrete Meth., 8 (1987), pp. 646–658.

[Lloy77] E. L. LLOYD, *On triangulations of a set of points in the plane*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 228–240.

[PlHo87] D. A. PLAISTED AND J. HONG, *A heuristic triangulation algorithm*, J. Algorithms, 8 (1987), pp. 405–437.

[PrSh85] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry —An Introduction*, Springer-Verlag, New York, 1985.

[Raja91] V. T. RAJAN, *Optimality of the Delaunay triangulation in $\Re^d$*, in Proc. 7th Annual ACM Symposium on Computational Geometry, 1991, pp. 357–363.

[Sibs78] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243–245.

[Supo83] K. J. SUPOWIT, *The relative neighborhood graph, with an application to minimum spanning trees*, J. Assoc. Comput. Mach., 30 (1983), pp. 428–448.

[Tous80] G. T. TOUSSAINT, *The relative neighbourhood graph of a finite planar set*, Pattern Recognition, 12 (1980), pp. 261–268.

[WGS90] F. W. WILSON, R. K. GOODRICH, AND W. SPRATTE, *Lawson's triangulation is nearly optimal for controlling error bounds*, SIAM J. Numer. Anal., 27 (1990), pp. 190–197.

[Wism80] S. K. WISMATH, *Triangulations: An algorithmic study*, Tech. Report 80-106, M. Sc. thesis, Department of Computer Science, Queen's University, Kingston, Ontario, Canada, 1980.

[WWW85] P. WIDMAYER, Y. F. WU, AND C. K. WONG, *Distance problems in computational geometry with fixed orientation*, in Proc. 1st Annual ACM Symposium on Computational Geometry, 1985, pp. 186–195.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 552–559, June 1993

INVERTING POLYNOMIALS AND FORMAL POWER SERIES*

K. KALORKOTI†

Abstract. The problem of inverting a general polynomial of degree $d > 0$, the inverse being a formal power series with coefficients $z_0, z_1, z_2, \ldots$ is considered. It is shown that computing $z_0, \ldots, z_n$ can be carried out in $n + (2d-1)\lceil \log_2 n \rceil + 2$ nonscalar operations over any infinite field. A lower bound of $n+1$ for all fields follows from standard linear-independence arguments. (The model of computation consists of straight-line algorithms.)

For fields of characteristic 0, it is also shown that computing $z_{n+1}, \ldots, z_{n+s}$ for $n \geq 0$, $s \geq 1$, requires at least $s + \min(n+1, d)/2 - 1$ nonscalar operations. For the case of inverting a general formal power series a corresponding lower bound of $s + (n+1)/2 - 1$ exists. In particular, computing the nth coefficient of the inverse of a general formal power series requires at least $n/2$ nonscalar operations (the coefficients are numbered from zero).

Key words. algebraic complexity, polynomials, formal power series, inverse

AMS(MOS) subject classifications. 68Q20, 68Q25, 68Q40

1. Introduction.

Let k be any infinite field, and let $t, x_0, x_1, x_2, \ldots$ be indeterminates over k. Define $y_0, y_1, y_2, \ldots$ by

$$\sum_{i=0}^{\infty} y_i t^i = \left(\sum_{i=0}^{\infty} x_i t^i \right)^{-1}.$$

Let L_n be the nonscalar complexity of computing $y_0, \ldots, y_n$, i.e.,

$$L_n = L(y_0, \ldots, y_n).$$

(See Borodin and Munro [2] for background.) It is easy to see that

$$y_i = \begin{cases} x_0^{-1} & \text{for } i = 0, \\ -x_0^{-1} \sum_{j=1}^{i} x_j y_{i-j} & \text{otherwise.} \end{cases} \tag{1}$$

The naïve algorithm implied by these equations clearly requires $\Omega(n^2)$ nonscalar operations. However, Sieveking [8] showed that $L_n \leq 5n - 2$. Kung [5] improved this to $L_n \leq 4n - \log_2 n$. Kung also showed that $L_n \geq n + 1$ and conjectured that $L_n \geq 4n -$ (lower-order terms). That the conjecture is false was observed by Schönhage [9], who noted that the second algorithm given in [5] requires at most $3.75n$ nonscalar operations. The improvement is obtained from the identity

$$\left(\sum_{i=0}^{n} a_i t^i \right)^2 \equiv \sum_{i=0}^{(n+1)/2-1} a_i t^i \left(\sum_{i=0}^{(n+1)/2-1} a_i t^i + 2 \sum_{i=(n+1)/2}^{n} a_i t^i \right) \qquad (\text{mod } t^{n+1}),$$

so that the first $n+1$ coefficients of the square can be computed in $3(n+1)/2$ nonscalar multiplications (this follows from the fact that the coefficients of the product of two polynomials of degree m and n, respectively, can be computed in $m + n + 1$ nonscalar multiplications—e.g., see Winograd [13]). Note that for fields of characteristic 2 we have

*Received by the editors May 8, 1991; accepted for publication (in revised form) February 27, 1992.

†Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, United Kingdom.

the equality $(a_0+a_1+\cdots+a_m)^2 = a_0^2+a_1^2+\cdots+a_m^2$, which means that the coefficients can be computed in just $n/2+1$ nonscalar multiplications and that this leads to an upper bound of $3.25n$.

In this paper we examine the problem of inverting a general polynomial of degree $d > 0$. Set

$$\sum_{i=0}^{\infty} z_i t^i = \left(\sum_{i=0}^{d} x_i t^i\right)^{-1}$$

We show that

$$L(z_{n+1},\ldots,z_{n+s}) \geq s + \min(n+1,d)/2 - 1, \qquad n \geq 0, s \geq 1,$$

for all fields of characteristic 0. For arbitrary fields we have

$$L(z_0,\ldots,z_n) \geq n+1, \qquad n \geq 0.$$

For an upper bound we have, when $s \leq n+1$,

$$L(z_{n+1},\ldots,z_{n+s}) \leq L(z_0,\ldots,z_n) + s + 2d - 1$$

for all infinite fields. This yields

$$L(z_0,\ldots,z_n) \leq n + (2d-1)\lceil \log_2 n \rceil + 2$$

for all $n \geq 1$.

Taking $n = d-1$, $s = d$, we have

$$L(z_d,\ldots,z_{2d-1}) \geq 1.5d - 1$$

for all fields of characteristic 0. For an upper bound we have

$$\begin{aligned} L(z_d,\ldots,z_{2d-1}) &\leq L(z_0,\ldots,z_{d-1}) + 3d - 1 \\ &\leq L(y_0,\ldots,y_{d-1}) + 3d - 1 \\ &\leq 6.75d - 4.75. \end{aligned}$$

We also have that for all fields of characteristic 0

$$L(y_{n+1},\ldots,y_{n+s}) \geq s + (n+1)/2 - 1, \qquad n \geq 0, s \geq 1.$$

Consequently

$$L(y_n) \geq \frac{n}{2}$$

for all n.

It is worthwhile to remark that Kung's lower bound is, in fact, shown for computing the first $n+1$ coefficients of the inverse of $x_0 + t$. The argument is, therefore, not valid as a proof of a lower bound on L_n for the usual (symbolic) model of straight-line algorithms. However, the lower bound, which holds for all characteristics, is an immediate consequence of straightforward arguments based on linear independence, such as those given by Fiduccia [3].

An alternative and more fruitful approach to obtaining a lower bound for L_n, due to Stoss [10], is as follows. Suppose that $\mathcal{A}$ is an algorithm that computes $y_0, \ldots, y_n$. It is clear from equations (1) that each y_i is a polynomial in x_0^{-1} and $x_1, \ldots, x_i$. Regard $\mathcal{A}$ as having scalars $k(x_0)$, and transform it to an algorithm $\mathcal{B}$, which computes only terms of degree 2 (Strassen [11]). The nonscalar complexity of $\mathcal{B}$ is no higher than that of $\mathcal{A}$, and its results are the coefficients of

$$x_0^{-3}\left(\sum_{i=1}^{n-2} x_i t^{i-1}\right)^2 \qquad (\text{mod } t^{n-1}).$$

For fields of characteristic other than 2 it can be seen that computing the coefficients of $(a_0 + a_1 t + \cdots + a_r t^r)$ mod t^{r+1} (where the a_i are indeterminates over k) requires at least $r + \lceil (r+1)/2 \rceil - 1$ nonscalar operations. This yields the lower bound

$$L_n \geq (n-2) + \lceil (n-1)/2 \rceil - 1 \geq 1.5n - 4,$$

which seems to be the best that is currently known.

We now give a sketch proof of the lower bound result on squaring polynomials (the author has not been able to find a proof in the literature and does not have a copy of [9] so the approach given here might be different). The proof uses the simple observation that any algorithm which computes l linearly independent quadratic forms must use at least l non-scalar steps. Let

$$b_0 + b_1 t + \cdots + b_r t^r \equiv (a_0 + a_1 t + \cdots + a_r t^r)^2 \quad (\text{mod } t^{r+1}).$$

The b_i are quadratic forms and so, by [11], they can be computed as

$$(2) \qquad (b_0, b_1, \ldots, b_r)^T = M(u_1 \times v_1, \ldots, u_m \times v_m)^T,$$

where M is an $(r+1) \times m$ matrix of scalars and each u_i, v_i is a k-linear combination of $a_0, \ldots, a_r$. It is easily shown that the b_i are linearly independent (over k), and so $m \geq r+1$. Now a_r must be present in some step $u_j \times v_j$, and this can be killed by substituting for a_r a k-linear combination of $a_0, \ldots, a_{r-1}$. A similar observation applies in turn to $a_{r-1}, \ldots, a_{s+1}$ where $s = \lceil (r+1)/2 \rceil$. Thus after killing $r - s$ nonscalar steps in (2) we obtain an algorithm that computes $b_0, \ldots, b_s, \bar{b}_{s+1}, \ldots, \bar{b}_r$, where

$$b_0 + \cdots + b_s t^s + \bar{b}_{s+1} t^{s+1} + \cdots + \bar{b}_r t^r \equiv (a_0 + \cdots + a_s t^s + l_1 t^{s+1} + \cdots + l_{r-s} t^r)^2 \quad (\text{mod } t^{r+1}),$$

where each l_i is a k-linear combination of $a_0, \ldots, a_s$. It is now easy to see that the b_i, $\bar{b}_j$ are linearly independent (consider the term $a_s a_j$ that occurs in $\bar{b}_{s+j}$ but not in $\bar{b}_{s+i}$ for $i < j$). Thus the computation of $b_0, \ldots, b_s, \bar{b}_{s+1}, \ldots, \bar{b}_r$ requires at least $r+1$ nonscalar steps and so in (2) we have $m \geq r + 1 + r - s \geq r + \lceil (r+1)/2 \rceil - 1$.

2. The upper bound. We describe a recursive algorithm for computing $z_{n+1}, \ldots$, z_{n+s}, where $s \leq n+1$, from $z_0, \ldots, z_n$. (This is the same as the algorithm given in §2 of [5] but with the power series replaced by a polynomial.)

The *order* of a formal power series $\sum_{i=0}^{\infty} c_i t^i$ is the least i such that $c_i \neq 0$ if this exists and is undefined otherwise. Trivially,

$$\sum_{i=n+1}^{\infty} z_i t^i = \left(1 - \sum_{i=0}^{n} z_i t^i \sum_{i=0}^{d} x_i t^i\right) \sum_{i=0}^{\infty} z_i t^i.$$

Noting that the parenthesized expression has order at least $n+1$, we immediately have

$$\sum_{i=n+1}^{n+s} z_i t^i = \left(1 - \sum_{i=0}^{n} z_i t^i \sum_{i=0}^{d} x_i t^i\right) \sum_{i=0}^{s-1} z_i t^i \qquad (\text{mod } t^{n+s+1}).$$

Define $a_{n+1}, \ldots, a_{n+d}$ by

$$\sum_{i=0}^{n} z_i t^i \sum_{i=0}^{d} x_i t^i = 1 + \sum_{n+1}^{n+d} a_i t^i,$$

and note that the a_i can be computed in d non-scalar multiplications. Thus

$$\sum_{i=n+1}^{n+s} z_i t^i = -\sum_{n+1}^{n+d} a_i t^i \sum_{i=0}^{s-1} z_i t^i \qquad (\text{mod } t^{n+s+1}),$$

and the product can be computed in $s+d-1$ nonscalar multiplications. Thus

$$L(z_{n+1}, \ldots, z_{n+s}) \leq L(z_0, \ldots, z_n) + d + s + d - 1.$$

From this it follows that

$$L(z_0, \ldots, z_n) \leq n + (2d-1)\lceil \log_2 n \rceil + 2.$$

The lower order terms in this bound could be improved by observing that for $n \leq d$

$$L(z_0, \ldots, z_n) = L(y_0, \ldots, y_n) \leq 3.75n.$$

The identities

$$\left(\sum_{i=0}^{\infty} x_i t^i\right)^{p-1} \left(\sum_{i=0}^{\infty} y_i t^i - \sum_{i=0}^{n} y_i t^i\right)^{p} \equiv 0 \qquad (\text{mod } t^{pn+p}),$$
$$\left(\sum_{i=0}^{d} x_i t^i\right)^{p-1} \left(\sum_{i=0}^{\infty} z_i t^i - \sum_{i=0}^{n} z_i t^i\right)^{p} \equiv 0 \qquad (\text{mod } t^{pn+p}),$$

can be used to obtain many different recursive algorithms for computing the y_i and z_i. For example, the algorithm given above corresponds to $p=2$ (and $s=n+1$), and the second algorithm of [5] corresponds to $p=3$.

An alternative way of computing the z_i for $i \geq d$ is to view them as the solutions of a dth-order linear recurrence. Fiduccia [4] shows how to compute z_n in $O(\mu(d,k)\log n)$ operations, where $\mu(d,k)$ is the cost of multiplying two degree $d-1$ polynomials in $k[t]$ and $z_0, \ldots, z_{d-1}$ are assumed to be given. Interpreting Fiduccia's results with the nonscalar complexity measure, we have that z_n can be computed in $O((2d-1)\log n)$ nonscalar operations since $z_0, \ldots, z_{d-1}$ can be computed in $3.75(d-1)$ nonscalar operations.

3. The lower bounds.

3.1. Preliminaries. This subsection provides a couple of variants of well known results and various technicalities. Let X_1, X_0 be disjoint finite sets of indeterminates over k. A step $a \circ b$ in an algorithm with scalars k and input indeterminates $X_1 \cup X_0$ is X_0*-inactive* if $\circ \in \{\times, /\}$ and one of the following holds

1. $b \in k$,
2. $a \in k$ and $\circ = \times$,
3. $a, b \in k(X_0)$

(see Pan [7] or Borodin and Munro [2]). We define X_0*–active* operations in the obvious manner. (When X_0 is a singleton set $\{x_0\}$ we abuse notation slightly and write x_0-active, etc.) If $X_0 = \emptyset$, then the X_0-active steps are just the nonscalar ones.

Let $f_1, \ldots, f_m \in k(X_1, X_0)$ and set

$$A_k(X_1, X_0) = \sum_{x \in X_1} kx + k(X_0).$$

We say that $f_{i_1}, \ldots, f_{i_s}$ are linearly independent mod $A_k(X_1, X_0)$ if whenever

$$\alpha_1 f_{i_1} + \cdots + \alpha_s f_{i_s} \in A_k(X_1, X_0)$$

for $\alpha_1, \ldots, \alpha_s \in k$, then

$$\alpha_1 = \cdots = \alpha_s = 0.$$

It is easy to show that if $f_1, \ldots, f_m$ has at least r such members, then any algorithm that computes $f_1, \ldots, f_m$ requires at least r X_0-active operations [3] (see also [2]).

Let $X = \{x_1, \ldots, x_d\}$, and suppose that $f \in k(X, x_0)$. Let the formal Laurent series of f with respect to x_0 be

$$f = \sum_{i=-e}^{\infty} f_i(X) x_0^i,$$

where $f_i(X) \in k(X)$, and set

$$\text{coeff}(f, x_0) = \{ f_i(X) \mid i \geq -e \}.$$

We define $\text{td}(f, x_0)$ to be the maximum number of elements of $\text{coeff}(f, x_0) - \{f_0(X)\}$ that are algebraically independent over k. (See Zariski and Samuel [12] for material on algebraic independence.) The next lemma gives us a lower bound on the number of x_0-active operations required to compute f in terms of $\text{td}(f, x_0)$ (see Motzkin [6]).

LEMMA 3.1. *Let f be as above. Then any algorithm that computes f has at least $\text{td}(f, x_0)/2$ x_0–active operations.*

Proof. Let $\mathcal{A}$ be an algorithm that computes f and whose results at x_0-active steps are $g_1, \ldots, g_r$ and set

$$C_r = \text{coeff}(g_1, x_0) \cup \cdots \cup \text{coeff}(g_r, x_0).$$

We show by induction on r that C_r has at most $2r$ elements that are algebraically independent over k.

The result is trivial for $r = 0$. For the induction step we have

$$g_r = a_1 \circ a_2,$$

where

$$a_i = l_i(X) + u_i(x_0) + \sum_{j=1}^{r-1} \alpha_{ij} g_j,$$

with $l_i(X) \in \sum_{x\in X} kx$, $u_i(x_0) \in k(x_0)$, and $\alpha_{ij} \in k$ for $1 \le i \le 2$ and $1 \le j \le r-1$. Thus C_r depends algebraically on $\{l_1(X), l_2(X)\} \cup C_{r-1}$, and so it has at most $2+2(r-1)$ algebraically independent elements. This completes the induction.

Now

$$f = l(X) + u(x_0) + \sum_{i=1}^{r} \beta_i g_i,$$

where $l(X) \in \sum_{x\in X} kx$, $u(x_0) \in k(x_0)$, and $\beta_i \in k$ for $1 \le i \le r$. Thus coeff$(f, x_0) - \{f_0(X)\}$ is algebraically dependent on C_r, and so it has at most $2r$ algebraically independent elements. □

The next lemma is an analogue to Fiduccia's method for combining row and column ranks in lower bounds [3] (see also Aho, Hopcroft, and Ullman [1]). A proof is included for the sake of completeness.

LEMMA 3.2. *Suppose that the sequence $f_1, \ldots, f_m \in k(X, x_0)$ has r members that are linearly independent* mod $A_k(X, x_0)$. *Let*

$$s = \min\{\, \mathrm{td}(\alpha_1 f_1 + \cdots + \alpha_m f_m, x_0) \mid \alpha_1, \ldots, \alpha_m \in k, \alpha_i \ne 0, \textit{ for some } i\,\}.$$

Then any algorithm that computes $f_1, \ldots, f_m$ requires at least $r + s/2 - 1$ nonscalar operations.

Proof. We may assume that $r > 0$; otherwise, the result is trivial. Let $e_1, \ldots, e_l$ be the sequence of nonscalar steps of an algorithm that computes $f_1, \ldots, f_m$. Thus there is an $m \times l$ matrix M of elements of k and a column vector $\mathbf{a}$ of m elements of $k + \sum_{x \in X\cup\{x_0\}} kx$ such that

$$\begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} = M \begin{pmatrix} e_1 \\ \vdots \\ e_l \end{pmatrix} + \mathbf{a}.$$

From the remarks preceding the lemma we know that $l \ge r$, and so we can partition M into an $m \times (l - r + 1)$ matrix M_1 and an $m \times (r-1)$ matrix M_2. We also set $\mathbf{e}_1 = (e_1, \ldots, e_{l-r+1})^T$ and $\mathbf{e}_2 = (e_{l-r+2}, \ldots, e_m)^T$. Thus

$$\begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} = M_1\mathbf{e}_1 + M_2\mathbf{e}_2 + \mathbf{a}.$$

Since M_2 has more rows than columns, it follows that there are $\alpha_1, \ldots, \alpha_m \in k$, not all 0, such that $(\alpha_1, \ldots, \alpha_m)M_2 = 0$. Thus

$$\alpha_1 f_1 + \cdots + \alpha_m f_m = (\alpha_1, \ldots, \alpha_m)M_1\mathbf{e}_1 + (\alpha_1, \ldots, \alpha_m)\mathbf{a}.$$

Lemma 3.1 now implies that $l - r + 1 \ge s/2$. □

LEMMA 3.3. *Suppose that $f_1, \ldots, f_i \in k(x_1, \ldots, x_i) - k(x_1, \ldots, x_{i-1})$ for $1 \le i \le m$. Then $f_1, \ldots, f_m$ are algebraically independent over k.*

Proof. We claim that x_i depends algebraically on $f_i, x_1, \ldots, x_{i-1}$ for $1 \le i \le m$. To see this, set

$$f_i = \frac{p_i(x_1, \ldots, x_{i-1}, x_i)}{q_i(x_1, \ldots, x_{i-1}, x_i)},$$

where $p_i, q_i \in k[x_1, \dots, x_i]$ are coprime and at least one of them is not in $k[x_1, \dots, x_{i-1}]$. Let ξ be a new indeterminate, and set

$$P(\xi) = q_i(x_1, \dots, x_{i-1}, \xi) f_i - p_i(x_1, \dots, x_{i-1}, \xi).$$

It is clear from the conditions on f_i, p_i, and q_i that $P(\xi) \neq 0$. Since $P(x_i) = 0$, the claim is established.

The lemma now follows by induction on m. For $m = 1$ we have that x_1 depends algebraically on f_1, and so f_1, must be transcendental over k. For the induction step suppose that $f_1, \dots, f_{m-1}$ are algebraically independent but that f_m depends algebraically on $f_1, \dots, f_{m-1}$. Since $f_1, \dots, f_{m-1}$ depend on $x_1, \dots, x_{m-1}$, it follows that f_m depends on $x_1, \dots, x_{m-1}$. Now x_m depends on $f_m, x_1, \dots, x_{m-1}$ and hence on $x_1, \dots, x_{m-1}$, which is a contradiction. □

LEMMA 3.4. *For arbitrary fields, z_m is a polynomial in x_0^{-1} of degree $m+1$ for all $m \geq 0$. Moreover, for fields of characteristic* 0 *and $m > 0$*

$$z_m = p_{m1} x_0^{-m-1} + p_{m2} x_0^{-m} + \cdots,$$

where $p_{mi} \in k[x_1, \dots, x_i] - k[x_1, \dots, x_{i-1}]$ for $1 \leq i \leq \min(m, d)$.

Proof. We have that

$$z_m = \begin{cases} x_0^{-1} & \text{for } m = 0, \\ -x_0^{-1} \sum_{i=1}^{\min(m,d)} x_i z_{m-i} & \text{otherwise.} \end{cases}$$

The result now follows from an easy induction on m. □

3.2. Lower bound results.

THEOREM 3.5.

1. *Let k be any field. Then*

$$L(z_0, \dots, z_n) \geq n + 1$$

for all $n \geq 0$.

2. *Let k be a field of characteristic* 0. *Then*

$$L(z_{n+1}, \dots, z_{n+s}) \geq s + \min(n+1, d)/2 - 1$$

for $n \geq 0$ and $s \geq 1$.

Proof. 1. Without loss of generality, we may assume that k is infinite. By Lemma 3.4 z_i is a polynomial in x_0^{-1} of degree $i + 1$ and so the z_i are all linearly independent mod $A_k(\{x_0, \dots, x_d\}, \emptyset)$.

2. Suppose that $\alpha_{n+1}, \dots, \alpha_{n+s} \in k$ are not all 0, and let α_{n+r} be the last nonzero member of the sequence. From Lemma 3.4 we have

$$\alpha_{n+1} z_{n+1} + \cdots + \alpha_{n+r} z_{n+r} = q_1 x_0^{-(n+r)-1} + q_2 x_0^{-(n+r)} + \cdots,$$

where for $1 \leq i \leq \min(n+r, d)$

$$q_i = \sum_{j=0}^{\min(i,d)-1} \alpha_{n+r-j} p_{n+r-j, i-j}.$$

Now $p_{n+r-j,i} \in k[x_1, \dots, x_i] - k[x_1, \dots, x_{i-1}]$ and $p_{n+r-j,i-j} \in k[x_1, \dots, x_{i-j}] - k[x_1, \dots, x_{i-j-1}]$. Since $\alpha_{n+r} \neq 0$, it follows that $q_i \in k[x_1, \dots, x_i] - k[x_1, \dots, x_{i-1}]$ for $1 \leq i \leq \min(n+r, d)$. Lemma 3.3 now implies that

$$\mathrm{td}(\alpha_{n+1} z_{n+1} + \cdots + \alpha_{n+s} z_{n+s}, x_0) \geq \min(n+r, d) \geq \min(n+1, d).$$

An application of Lemma 3.2 completes the proof. □

COROLLARY 3.6. *Let k be a field of characteristic* 0. *Then*

$$L(y_{n+1}, \dots, y_{n+s}) \geq s + (n+1)/2 - 1$$

for $n \geq 0$ and $s \geq 1$.

Proof. This is an immediate consequence of Theorem 3.5, for if we take $d \geq n+s$, then $z_i = y_i$ for $0 \leq i \leq n+s$. □

Acknowledgment. I should like to thank an anonymous referee whose helpful comments led to improvements in the paper.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[3] C. M. FIDUCCIA, *Fast matrix multiplication*, in Proc. 3rd Annual ACM Symposium on Theory of Computing, (1971), pp. 45–49.

[4] ———, *An efficient formula for linear recurrences*, SIAM J. Comput., 14 (1985), pp. 106–112.

[5] H. T. KUNG, *On computing reciprocals of power series*, Numer. Math., 22 (1974), pp. 341–348.

[6] T. S. MOTZKIN, *Evaluation of polynomials and evaluation of rational functions*, Bull. Amer. Math. Soc., 61 (1955), p. 163.

[7] V. Y. PAN, *Methods of computing values of polynomials*, Russian Math. Surveys, 21 (1966), pp. 105–136.

[8] M. SIEVEKING, *An algorithm for division of powerseries*, Computing, 10 (1972), pp. 153–156.

[9] A. SCHÖNHAGE, private communication, 1981.

[10] H.-J. STOSS, unpublished, 1981.

[11] V. STRASSEN, *Vermeidung von Divisionen*, J. Reine Angewandte Math., 264 (1973), pp. 184–202.

[12] O. ZARISKI AND P. SAMUEL, *Commutative Algebra*, Vol. I, Springer, Berlin, 1986.

[13] S. WINOGRAD, *Some bilinear forms whose multiplicative complexity depends on the field of constants*, Math. Systems Theory, 10 (1977), pp. 169–180.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 560–572, June 1993

NONDETERMINISM WITHIN P*

JONATHAN F. BUSS† AND JUDY GOLDSMITH‡

Abstract. Classes of machines using very limited amounts of nondeterminism are studied. The $P =? NP$ question is related to questions about classes lying within P. Complete sets for these classes are given.

Key words. nondeterminism, quasilinear time, computational complexity

AMS(MOS) subject classifications. 68Q15, 68Q05

1. Introduction. Traditional complexity theory gives a distinguished role to the class P of languages that are computable in polynomial time. Languages not in P are considered to be computationally intractable. However, the converse assertion, that languages in P have efficient solutions, is untenable. Historically, one reason for the choice of polynomial time bounds was that no smaller class appears to be as robust against change of model. We consider here subclasses of P, defined (roughly) according to the exponent of the polynomial bounding the running time and also according to the small amount of nondeterminism allowed. Of particular interest is the class of languages computable in deterministic "quasilinear" time, which is less robust than P with respect to change of model, but is much closer to being a practical notion of feasible computation. We develop a complexity theory based on quasilinear-time many-one reducibility that is analogous to the theory of polynomial-time reducibility.

We primarily consider the trade-off between time and nondeterminism. To replace nondeterminism by time in Turing machine computations is easy: one binary nondeterministic choice may be eliminated by doubling the time. Two fundamental questions are whether nondeterminism can be eliminated at a lower cost in time, and whether computation time can be decreased by adding nondeterminism. We define subclasses of P by limiting the time bound to fixed-degree polynomials and allowing nondeterminism over a fixed polynomial-size search space. The inclusion relationships among the classes are particular cases of the two fundamental questions. We show that sufficient separation among these classes lying inside P implies that $P \neq NP$.

Let f and g be integer functions. We will write "$f \in Q(g)$" and say "f is of quasi-order g" if there is a positive constant k such that[1] $f \in O(g \log^k g)$. For any positive constant l, we denote by P_l the class of languages that can be accepted by a multitape Turing machine whose running time on inputs of length n is of quasi-order n^l. For integers $l \geq 1$ and $m \geq 0$, we denote by N^mP_l the class of languages accepted by nondeterministic machines in time of quasi-order n^l making at most $m \log n$ binary nondeterministic choices. For any m and l, N^mP_l is contained in P_{m+l}. We will also consider the classes NP_l having no bound on the amount of nondeterminism. The classes P_1 and NP_1 have been previously studied by Schnorr [16] under the names QL and NQL.

*Received by the editors October 29, 1990; accepted for publication (in revised form) March 13, 1992.

†Department of Computer Science, University of Waterloo, Ontario, Canada N2L 3G1. The work of this author was supported in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

‡Department of Mathematics and Computer Science, Dartmouth College, Hanover, New Hampshire 03755. Present address, Department of Computer Science, University of Manitoba, Canada R3T 2N2. The work of this author was supported in part by National Science Foundation grant RII-900305.

[1]We will write "$\log n$" to mean $\lfloor \log_2(n+1) \rfloor$ throughout the paper.

The $\log^k n$ factors in the allowed time bounds have a number of important consequences. First, the classes are independent of the number of tapes of a multitape Turing machine. Second, quasilinear-time reducibility is transitive and preserves membership in N^mP_l, for any m and l. Third, quasilinear time is large enough to admit basic algorithmic techniques such as sorting and multiplication. For computation of practical importance, the input size is generally less than, say, 10^{12}; hence $\log n$ is bounded by 40, and ignoring arbitrary logarithmic factors is no worse than ignoring arbitrary constant factors.

The notion of classifying problems according to the amount of nondeterminism required has appeared previously; see, e.g., Kintala and Fischer [13] and Wolf [17]. Àlvarez, Díaz, and Torán , [4], [8] consider a hierarchy, similar to the present one, of classes lying between P and NP. Their hierarchy differs from the present one in two ways. First, any polynomial is allowed in the running times, and second, the bound on the number of nondeterministic choices that defines each class is a power of $\log n$ rather than a multiple of $\log n$. Thus each class contains all of P, and the hierarchy does not provide a classification among computations feasible in practice.

More similar in method to the present paper is the work of Geske [9], who considers relations among classes $TIME(n^l)$ and $NTIME(n^l)$, which are similar to P_l and NP_l, but more restrictive. Weaker variants of our Theorems 5.1 and 5.4 are obtained.

In a related work, Kaye [12] gives two logical characterizations of a family of classes $L^{(\alpha)}$, $\alpha \geq 0$, lying between linear time and P. Kaye's class L^2 is precisely quasilinear time.

Gurevich and Shelah [10] consider quasilinear-time reductions among various models of random-access machines. They define *nearly linear time*, or NLT, and show that the models they consider, including Kolmogorov machines, Schönhage machines, and random-access Turing machines, all compute the same NLT functions in quasilinear time. They also show that the languages accepted nondeterministically by any of these machines in quasilinear time are precisely those languages accepted nondeterministically in quasilinear time by a multitape Turing machine (denoted NQL in [16], herein NP_1). They conjecture that the deterministic classes NLT and QL (herein P_1) are different. We offer some candidate languages for this separation.

Quasilinear-time reductions have been studied in other contexts as well. Hunt and Stearns [11] define the notion of Turing-SAT-hard (n polylog n, n) in order to make precise the notion that sets in NP have deterministic time and space complexity no less than that of SAT. Their definition restricts the reductions considered to quasilinear time *and* linear space complexity. Because the sets they consider are in $NP - P$ unless $P = NP$, their results do not provide much insight into the classes considered here.

Abrahamson et al. [1], [2] study the complexity of families of languages parameterized by the size of an associated search space. The present paper concerns the complexity of individual problems, which may or may not be part of a larger family. The relationship between the two approaches is investigated in §4 below.

This paper is organized as follows. Section 2 gives some basic properties of the classes N^mP_l and P_l. Section 3 shows that the classes have complete sets, and some natural problems are considered as candidate complete sets. Section 4 contains a discussion of the relationship of the present work to that of Abrahamson et al. Section 5 investigates relationships among the various classes N^mP_l. Section 6 concludes with some open problems and directions for future work.

2. Preliminaries. The classes N^mP_l can be characterized in terms of witnesses, analogously to NP.

THEOREM 2.1. *For all m and l, a set A is in N^mP_l if and only if there is an R in P_l such that for all strings x, $x \in A \Leftrightarrow \exists y\,[|y| \leq m \log |x| \wedge R(x,y)]$.*

The proof is left to the reader.

Like most hierarchies of complexity classes, the limited-nondeterminism classes exhibit downward separation (upward collapse).

THEOREM 2.2. *For all m and l,*

1. *if $N^1P_l \subseteq P_l$, then $N^1P_k \subseteq P_k$ for all $k \geq l$, and*
2. *if $N^{m+1}P_l \subseteq N^mP_l$, then $N^kP_l \subseteq N^mP_l$ for all $k \geq m$.*

Proof (sketch). Suppose that $N^1P_l \subseteq P_l$, and fix $k > l$ and $A \in N^1P_k$. Let R be as guaranteed in the previous result. Let $A' = \{\, x0^{\lfloor |x|^{k/l} \rfloor - |x|} \mid x \in A \,\}$. Then $x0^j \in A'$ if and only if $j = \lfloor |x|^{k/l} \rfloor - |x| \wedge \exists y\, |y| \leq \log |x| \wedge R(x,y)$ if and only if $\exists y\, |y| \leq \log |x0^j| \wedge R'(x0^j, y)$, where $R' = \{\, \langle x0^j, y \rangle \mid j = \lfloor |x|^{k/l} \rfloor - |x| \wedge y \leq \log |x| \wedge R(x,y) \,\}$. Hence $A' \in N^1P_l \subseteq P_l$, and thus $A \in P_k$.

Now suppose that $N^{m+1}P_l \subseteq N^mP_l$, and fix $k > m+1$ and $B \in N^kP_l$. There exists S such that $x \in B \Leftrightarrow \exists y\, |y| \leq k \log |x| \wedge \langle x, y \rangle \in S$. The latter condition is equivalent to $\exists y\, |y| \leq (k - m - 1) \log |x| \wedge S'(x,y)$, where $S' = \{\, \langle x, y \rangle \mid \exists z\, z \leq (m+1) \log |x| \wedge \langle x, yz \rangle \in S \,\}$ is in $N^{m+1}P_l$ and hence in N^mP_l by assumption. Thus $B \in N^{k-1}P_l$, which is contained in N^mP_l by induction. □

The hierarchy also exhibits a kind of upward separation.

THEOREM 2.3. *Suppose that for all l, there is an m such that N^mP_1 is not contained in P_l. Then $P \neq NP$.*

The result follows immediately from Theorem 2.4.

THEOREM 2.4. *If $P = NP$, then there is an l such that $NP_k \subseteq P_{kl}$ for all k.*

Proof. Let L be a complete set for NP_1 with respect to quasilinear-time reductions. (Schnorr has shown that SAT, the set of satisfiable Boolean formulas, is such a set [16].) Suppose $P = NP$; i.e., L is in P_l for some l. Let $A \in NP_k$. Membership questions of length n about A are reducible in time $Q(n^k)$ to membership questions of length $Q(n^k)$ about L (using a padding argument and Schnorr's result), which are answerable in time $Q(n^{kl})$ by assumption. Hence $NP_k \subseteq P_{kl}$. □

3. Complete sets.

The sets N^mP_l all have complete sets under quasilinear-time reducibility. We consider first a problem concerning acceptance by Turing machines. Let $\langle \cdot \rangle$ be a coding function for Turing machines such that given $\langle M \rangle$, a single step of M can be simulated by a multitape Turing machine in time proportional to $\log |\langle M \rangle|$. A unary code will have the required property.

Define $G_l^m = \{\, \langle M \rangle \# x \# 1^j \mid M$ accepts input x within $j|x|^{l-1}$ steps using at most $m \log |x|$ nondeterministic choices $\}$.

THEOREM 3.1. *For all m and l, G_l^m is complete for N^mP_l with respect to quasilinear-time reductions.*

Proof (sketch). The coding convention ensures that $G_l^m \in N^mP_l$. For $A \in N^mP_l$, fix a machine M and constants r and s such that M halts in at most $rn^l \log^s n$ steps, uses $m \log n$ nondeterministic choices on inputs of length n, and accepts A. Let $f(x) = \langle M \rangle \# x \# 1^{r|x| \log^s |x|}$. Then $x \in A$ if and only if $f(x) \in G_l^m$, and f is computable in time in $Q(n)$. Hence f is the desired reduction from A to G_l^m. □

In some cases, complete sets for N^mP_l can be obtained from NP-complete problems by bounding the size of allowable witnesses for membership. We first consider topologically ordered Boolean circuits.[2] A circuit is *satisfiable* if some setting of the inputs results

[2]A circuit is topologically ordered if each gate is numbered, and the inputs to a gate all have lower numbers than the gate itself.

in an output of 1. Let CSAT(k) be the set of satisfiable, topologically ordered circuits that have fewer than $k \log n$ inputs, where n is the number of gates in the circuit. (Constants 0 and 1 are not counted as inputs.)

THEOREM 3.2. *For all k,* CSAT(k) *is complete for N^kP_1 with respect to quasilinear-time reductions.*[3]

Proof (sketch). The standard methods for proving the formula-satisfiability problem to be NP-complete can easily be modified to produce a circuit rather than a formula. Schnorr [16] and Cook [7] have shown that the circuit (or formula) corresponding to a given input for a fixed Turing machine may be constructed in quasilinear time. In particular, the circuit has quasilinear size. Also, it is topologically ordered.

Let $A \in N^kP_1$, and let $R \in P_1$ be such that $x \in A \Leftrightarrow \exists y\, |y| \leq k \log |x| \wedge \langle x, y \rangle \in R$, for all strings x. Let M accept R in quasilinear time. The circuit $C_M(x)$ corresponding to M with input $\langle x, y \rangle$, for fixed x and undetermined y, has size quasilinear in $|x|$ and only $k \log |x|$ inputs. Hence $x \in A$ if and only if $C_M(x) \in$ CSAT(k), and CSAT(k) is hard for N^kP_1.

Evaluation of a circuit at given inputs can be accomplished in quasilinear time on a Turing machine by using an algorithm of Pippenger [14]. Hence CSAT(k) is in N^kP_1. □

The satisfiability problem for formulas with a limited number of variables, SAT(k), is perhaps easier than CSAT(k). The above proof of hardness of CSAT(k) does not apply to SAT(k), because too many auxiliary variables are required in a formula to simulate a Turing machine.[4] In the absence of a hardness proof for the restricted formula-satisfiability problem, hardness proofs for restrictions of other standard NP-complete problems are problematic. In some cases, the natural restriction is essentially as hard as the general case. For example, graph k-colorability remains NP-complete when k is fixed at 3.

The restricted vertex-cover problem exhibits different behaviour. Let VC(k) be the language of undirected graphs that have a vertex cover of size k. It is easy to show that VC(k) is in $N^{k-1}P_1$: the first $k-1$ vertices of a cover can be chosen nondeterministically, and the last vertex found by a quasilinear-time search. In fact, the language VC(k) can be accepted in linear time by the following algorithm due to S. Buss [6]. Given a graph $G = (V, E)$,

1. Let U be the set of vertices of degree more than k. If $|U| > k$, then reject; there is no cover of size k or less.
2. Let G' be the subgraph of G induced by $V - U$. Every k-cover of G consists of U together with a $(k - |U|)$-cover of G'. If G' has more than $k(k - |U|)$ edges, then reject; G' has no $(k - |U|)$-cover.
3. If G' has a cover of size $k - |U|$, then accept; otherwise reject.

Steps 1 and 2 can easily be implemented in quasilinear time. If step 3 is reached, then G' has a bounded number of edges; hence step 3 requires $Q(1)$ time. Therefore, VC(k) is in P_1, and a completeness proof for VC(k) would have strong consequences.

THEOREM 3.3. *If* VC(k) *is hard for N^jP_1 for some $j \geq 1$, then $N^mP_1 \subseteq P_1$ for all m.*

The result follows immediately from Theorem 2.2.

The k-clique problem, although often regarded as a trivial variation on the k-vertex cover problem, cannot be substituted for VC(k) in the above proof, because the condition on the degrees does not hold. Thus k-clique remains as a candidate hard problem for $N^{k-1}P_1$.

[3] Díaz and Torán [8] have shown that a variant of CSAT(k) is complete under logspace reductions for their class β_k, for each k.

[4] We return to the formula-satisfiability problem in §4.

Next, we consider the following problem concerning context-free grammars. Given a grammar $G = (N, \Sigma, P, S)$, does G generate any string comprising repetitions of a single terminal; i.e., is $\mathcal{L}(G) \cap \bigcup_{\sigma \in \Sigma} \sigma^* \neq \emptyset$? Let UNARYGEN be the set of grammars for which the answer is yes.

THEOREM 3.4. CSAT(1) $\leq_{ql}$ UNARYGEN.

Proof. Let C be a circuit with n gates $g_1, \ldots, g_n$. Assume without loss of generality that gates g_i with $i \leq \log n$ are the inputs and g_n is the output. Each gate g_j is represented by two nonterminal symbols A_j and A'_j. There are also n terminal symbols $b_0, \ldots, b_{n-1}$, each corresponding to one setting of the values of the input variables. The start symbol is A_n.

The productions are determined as follows.

• If g_i is an and-gate with predecessors g_j and g_k, use productions $A_i \longmapsto A_j A_k$ and $A'_i \longmapsto A'_j \mid A'_k$.

• If g_i is an or-gate with predecessors g_j and g_k, use productions $A_i \longmapsto A_j \mid A_k$ and $A'_i \longmapsto A'_j A'_k$.

• If g_i is a not-gate with predecessor g_j, use productions $A_i \longmapsto A'_j$ and $A'_i \longmapsto A_j$.

• For $1 \leq i \leq \log n$, for all j such that bit i of the binary expansion of j is 1, use $A_i \longmapsto b_j$.

• For $1 \leq i \leq \log n$, for all j such that bit i of the binary expansion of j is 0, use $A'_i \longmapsto b_j$.

The variable A_j derives a unary word if and only if g_j evaluates to 1, and A'_j derives a unary word if and only if g_j evaluates to 0.

LEMMA 3.5. *For all* $1 \leq i \leq n$ *and* $0 \leq j \leq n-1$, $A_i \longmapsto^* b_j^k$ *for some* k *if and only if* g_i *evaluates to* 1 *when the input vector is the binary expansion of* j. *Similarly*, $A'_i \longmapsto^* b_j^k$ *for some* k *if and only if* g_i *evaluates to* 0 *when the input vector is the binary expansion of* j.

Proof. Fix j. Define the *level* of a gate to be the length of the longest directed path from the gate to any input. We use induction on the level of gate g_i.

The base case is that g_i is an input. Then g_i evaluates to 1 if and only if bit i of j is 1 if and only if $A_i \longmapsto b_j$ if and only if $A_i \longmapsto^* b_j$. Also g_i evaluates to 0 if and only if bit i of j is 0 if and only if $A'_i \longmapsto b_j$ if and only if $A'_i \longmapsto^* b_j$.

Suppose that the lemma holds for all gates with level at most $l-1$, and let g_i have level l. If g_i is an and-gate with predecessors g_{i_1} and g_{i_2}, then g_i evaluates to 1 if and only if both g_{i_1} and g_{i_2} evaluate to 1 if and only if $A_{i_1} \longmapsto^* b_j^{k_1}$ and $A_{i_2} \longmapsto^* b_j^{k_2}$ for some k_1 and k_2 if and only if $A_i \longmapsto^* b_j^{k_1+k_2}$. Likewise, g_i evaluates to 0 if and only if either g_{i_1} or g_{i_2} evaluates to 0 if and only if $A'_{i_1} \longmapsto^* b_j^k$ or $A'_{i_2} \longmapsto^* b_j^k$ for some k if and only if $A'_i \longmapsto^* b_j^k$. The cases where g_i is a negation gate or an or-gate are similar. □

The theorem follows immediately from the lemma. □

The reduction given in the proof holds even if the circuit is not topologically ordered. If the circuit is ordered, then in the resulting grammar, every appearance of each nonterminal in the left-hand side of a production occurs after every appearance of the nonterminal on the right-hand side of a production. Call a grammar satisfying this condition an ordered grammar. Let UNARYGEN$_{\text{ord}}$ be the set of ordered grammars in UNARYGEN.

THEOREM 3.6. UNARYGEN$_{\text{ord}}$ *is complete for* N^1P_1.

Proof. Because the reduction of the previous theorem produces an ordered grammar from an ordered circuit, UNARYGEN$_{\text{ord}}$ is hard for N^1P_1. □

To show that UNARYGEN$_{\text{ord}}$ is in N^1P_1, we use the following lemma.

LEMMA 3.7. *The emptiness problem for ordered context-free grammars is in deterministic quasilinear time.*

Proof. Call a nonterminal "true" if some terminal string can be derived from it and "false" otherwise. The grammar is then a network in the sense of Pippenger [14] and thus can be evaluated in time $Q(n)$. □

Let $G = (N, \Sigma, P, S)$ be an ordered grammar. Nondeterministically guess a terminal $\sigma \in \Sigma$, using $\log |\Sigma| \leq \log |G|$ bits. Then eliminate all productions involving terminals other than σ. Accept if and only if the resulting grammar generates a nonempty language.

The above algorithm places UNARYGEN$_{\text{ord}}$ in N^1P_1, and hence it is complete. □

4. Families of constrained-search problems. Many NP-complete problems involve a parameter that defines a constraint on the associated search problem. Restriction of this parameter produces a problem whose complexity falls into one of at least three classes: (1) The restricted problem may be NP-complete; e.g., graph k-coloring, for any $k \geq 3$. (2) Every fixed value of the parameter may give the same polynomial complexity; e.g., vertex k-cover. (3) Every fixed value of the parameter may give polynomial-time complexity, but the exponent of the polynomial may depend on k. No instances of the third case are proven, but CSAT(k), k-clique and many other problems are candidates. The distinction between the second and third case has been recently investigated by Abrahamson et al. [1], [2], who considered families of associated search problems. We show here that limited nondeterminism provides an alternative to their methods.

Consider the case of CSAT(k), for $k \geq 1$. These languages form an infinite collection of complete sets, one for each class N^kP_1. It is not an arbitrary collection, however, since the complete set for N^kP_1 can easily be determined given k. We describe this situation using the concept of *language family*. A language family is a subset of $\Sigma^* \times \mathbf{N}$. The kth slice of a family B, denoted B_k, is the language $\{ x \mid (x, k) \in B \}$.

The class N^*P_j (for all $j \geq 1$) consists of all families B with the following properties.

• The kth slice of B is a language in N^kP_j.

• There is a quasilinear-time-computable function f such that for all k, the value of $f(k)$ is the code of a Turing machine witnessing that the kth slice of B is in N^kP_j. (An object x *witnesses* an existential property $\exists y\, p(y)$ if and only if $p(x)$ is true. The computability requirement on f may be relaxed without changing the essential character of the definition.)

A language family C in N^*P_j is *$\leq_{ql}$-complete for N^*P_j* if the following hold.

• For all k, the slice C_k is complete for N^kP_j.

• There is a quasilinear-time-computable function f such that for all k, the value of $f(k)$ is the code of a Turing machine witnessing that C_k is complete for N^kP_j (i.e., the Turing machine performs a quasilinear-time reduction from G_j^k to C_k).

For example, the family CSAT($*$) $= \{ (x, k) \mid x \in$ CSAT(k) $\}$ is complete for N^*P_1.

We draw a second example from Abrahamson et al. [2]. Consider a Boolean formula. A partial assignment of truth values to variables in the formula may induce values on other variables if the formula is to be true. Consider a formula in conjunctive normal form with three or fewer variables per clause (3CNF). If $(a_1 \vee a_2 \vee a_3)$ is a clause, and all but one of the a_i's are false, then the remaining variable is induced to assume the value true; if one variable a_i is true, then the clause is true, and no additional assignment is induced. If this procedure can be applied iteratively until all clauses evaluate to true, then we say the original partial assignment caused the formula to *unravel*. Let SHORTSAT(k) be the set of 3CNF formulas B such that some assignment to the first $k \log |B|$ variables causes B to unravel.[5]

[5]The definition in [2] allowed arbitrary CNF formulas instead of 3CNF. The difference appears to be necessary for Theorem 4.1. It does not affect the results of [1].

A formula $B = \bigwedge_{i=1}^{k} C_i$ in conjunctive normal form *unravels in order* if each initial subformula $\bigwedge_{i=1}^{j} C_i$ unravels. Let $\text{SHORTSAT}_{\text{ord}}(k)$ be the set of formulas B such that some assignment to the first $k \log |B|$ variables causes B to unravel in order.

THEOREM 4.1. *$\text{SHORTSAT}_{\text{ord}}(k)$ is $\leq_{ql}$-complete for N^kP_1, and $\text{SHORTSAT}(k)$ is $\leq_{ql}$-hard for N^kP_1, for all k.*

Proof (sketch). We first show that $\text{CSAT}(k) \leq_{ql} \text{SHORTSAT}_{\text{ord}}(k)$. Let Γ be an instance of $\text{CSAT}(k)$. The corresponding formula φ has one variable for each gate of the circuit, and two additional variables denoted z_1 and z_2. The first $k \log n$ clauses of φ represent the input. The connections of a gate g in Γ are represented in φ by a conjunction B of clauses. The conjunction B always unravels in φ, and the only unravelling induces the variable representing g to take on the value of g. The final clause of φ unravels if and only if the circuit evaluates to 1.

The clauses are as follows.

• For each input variable x_i of Γ, the corresponding clause is $(x_i \vee \neg x_i)$.

• If g is an and-gate with predecessors a and b, then the corresponding conjunction is $(\neg a \vee \neg b \vee g) \wedge (a \vee \neg g) \wedge (b \vee \neg g)$.

• If g is an or-gate with predecessors a and b, then the corresponding conjunction is $(a \vee b \vee \neg g) \wedge (\neg a \vee g) \wedge (\neg b \vee g)$.

• If g is a not-gate with predecessor a, then the corresponding conjunction is $(a \vee \neg g) \wedge (\neg a \vee g)$.

• If the output gate is g_{out}, the final clause is $(g_{\text{out}} \vee z_1 \vee z_2)$.

The reader can easily check that the formula has the required properties.

To show that $\text{SHORTSAT}_{\text{ord}}(k) \leq_{ql} \text{CSAT}(k)$, let φ be a formula in 3CNF, with clauses $\{\varphi_i\}_{i=1}^{c}$. Each clause φ_i will be represented by a circuit C_i, whose inputs will be outputs of the circuits representing earlier clauses. We assume without loss of generality that no variable appears twice in any one clause.

Consider $\varphi_i = v_{i1} \vee v_{i2} \vee v_{i3}$. The corresponding circuit C_i will have six inputs and six outputs. The circuit C_i will compute the status of each literal v_{ij} (either a variable or a negated variable) and produce two corresponding output values. Output $v_{ij}(1)$ is given value 1 if and only if the unravelling procedure assigns a value to v_{ij} at or before clause φ_i. If $v_{ij}(1) = 1$, then output $v_{ij}(2)$ is given the value assigned to v_{ij}.

The inputs to C_i are determined by the last occurrences of the variables of φ_i previous to φ_i itself. Denote by v'_{ij} the last literal containing the variable of v_{ij} appearing in φ before φ_i. The first input corresponding to v_{ij} is $v'_{ij}(1)$. The second input corresponding to v_{ij} is $v'_{ij}(2)$ if v_{ij} and v'_{ij} are both positive or both negative literals, and is $\neg v'_{ij}(2)$ otherwise. The construction of C_i is straightforward except for the identification of the input literals v'_{ij}.

If each v'_{ij} is found individually, the construction of the entire circuit may take quadratic time. Hence the successive occurrences of each variable must all be determined at once, as follows. For each v_{ij}, form a triple (x, i, j), where x is the index of the variable occurring in v_{ij}. Sort the triples, using the order $(x, i, j) < (y, k, l)$ if and only if $x < y$ or $x = y$ and $i < k$. If (x, i, j) and (x, k, l) are adjacent in the sorted list, then $v'_{kl} = v_{ij}$; form a quadruple (i, j, k, l). Sorting these quadruples gives the required adjacency information in the order needed to construct the circuit in topological order.

The final circuit comprises all subcircuits C_i and an additional output subcircuit. The output subcircuit computes $\bigwedge_{i=1}^{c} \bigvee_{j=1}^{3} \Big(v_{ij}(1) \wedge v_{ij}(2)\Big)$, which has value 1 if and only if the original formula φ unravels in order. □

The family $\text{SHORTSAT}_{\text{ord}}(*) = \{ (x, k) \mid x \in \text{SHORTSAT}_{\text{ord}}(k) \}$ is thus complete for N^*P_1.

THEOREM 4.2. *For all k,* SHORTSAT$(k) \in N^kP_2$.

Proof (sketch). To determine whether a formula φ is in SHORTSAT(k), it suffices to nondeterministically guess the values of the initial $k \log |\varphi|$ variables, and then make at most $|\varphi|$ passes through φ, inducing variables and propagating their values. □

A language family C is *$\leq_{ql}$-hard for N^*P_j* if the following hold.

• For all k, there is a $k' \geq k$ such that $C_{k'}$ is $\leq_{ql}$-hard for N^kP_j.

• There is a quasilinear-time-computable function f such that for all k, the value of $f(k)$ is the code of a Turing machine witnessing that $C_{k'}$ is $\leq_{ql}$-hard for N^kP_j.

For example, SHORTSAT(*)= $\{(x,k) \mid x \in \text{SHORTSAT}(k)\}$ is $\leq_{ql}$-hard for N^*P_1. The family $G_* = \{(G_k^0, k) \mid k \geq 1\}$ is $\leq_{ql}$-hard for N^*P_l, for every l.

Finally, a language family C is *weakly $\leq_{ql}$-complete for N^*P_j* if it is both in N^*P_j and $\leq_{ql}$-hard for N^*P_j. One example of a weakly complete set for N^*P_1 is $\{(x, 2k) \mid x \in \text{CSAT}(k)\}$.

The definition of N^*P_j is motivated by the work of Abrahamson et al. [1], [2] on their class PGT. Their work focused on families of problems in P, usually generated by parameterized problems in NP, such as SHORTSAT() and CSAT(). The class PGT is roughly equivalent to the notion "in some N^*P_j and hard for some N^*P_k." The major differences are that PGT includes nonuniform families and that reductions may be nonuniform; in the terms of this paper, the functions f above are not required to be computable. All of the complete families considered by Abrahamson et al., however, are uniform and are examples of families in some N^*P_j and hard for some N^*P_k. Thus, their work provided examples of sets in the classes N^iP_k, some of which were also hard for classes N^iP_1. Further work on limited nondeterminism is likely to yield additional complete sets for PGT.

5. Relationships among the classes.

The classes N^mP_l bear a strong analogy to NP; however, the analogy should not be taken too far. For example, one might conclude from Theorem 2.3 that NP_l behaves as a limiting case of N^mP_l as m goes to infinity. This conclusion, however, can be false in the presence of an oracle. Denote by C^X the class C relative to oracle X. The results of §2 hold relative to any oracle X.

THEOREM 5.1. *There is an oracle A such that $N^mP_k^A = P_k^A$ for all m and k, and yet $NP^A \not\subseteq P^A$.*

Proof. By Theorem 2.2, we need only show that $N^1P_1^A = P_1^A$ and $P^A \neq NP^A$.

Fix an enumeration $\{M_i\}_{i=1}^{\infty}$ of nondeterministic machines such that M_i runs in quasilinear time and makes at most $\log n$ nondeterministic choices on inputs of length n. Fix an enumeration $\{T_i\}_{i=1}^{\infty}$ of deterministic machines such that T_i runs for at most $p_i(n) = n^i + i$ steps on inputs of length n.

We construct A to have the following two properties for all i:

C$_i$: The coding condition: For all strings x and all positive integers s, the string $0^{2s}\#i\#x$ is in A if and only if $M_i^A(x)$ accepts within s steps. (Note that $M_i^A(x)$ cannot query whether $0^{2s}\#i\#x$ is in A within s steps.)

D$_i$: The diagonalization condition: The language

$$L_A = \{ 1^n \mid \exists y\, |y| = n \wedge 1^n y \in A \}$$

is not accepted by T_i.

The construction proceeds in stages. During stage k, all strings of length k are fixed to be in or out of A, in such a way that conditions C$_i$ hold up to length k. In addition, some strings of length longer than k may be fixed to satisfy some condition D_i. Because each string is fixed exactly once during the construction, the oracle A is well defined. In fact, A is computable in space $n^{O(\log n)}$.

Initially, all strings are unfixed. Set $k = d = l = 1$; k is the stage number, d is the current diagonalization condition, and l is a lower bound on the input at which condition D_d can be satisfied.

Stage k: For all strings z of length k of the form $z = 0^{2s}\#i\#x$ for some i, x, and s, fix z in A if M_i accepts x with oracle A in $\leq s$ steps; otherwise fix z out of A. (Note that this condition is determined by the preceding stages of the construction.) Fix all strings of length k not of the above form out of A. If $k < l$ or $2^k \leq p_d(k)^{1+\log p_d(k)}$, then stage k is complete—continue to stage $k+1$.

If $k \geq l$ and $2^k > p_d(k)^{1+\log p_d(k)}$, then a diagonalization condition can be satisfied. Simulate T_d on input 1^k. When T_d queries a string z not yet fixed, there are two cases.

1. If $z = 0^{2s}\#i\#x$ for some i, x, and s, then simulate M_i on input x for s steps, and fix z accordingly. Fix strings queried during the simulation by applying these cases recursively.

2. Otherwise, fix z out of A.

If T_d accepts 1^k, then fix all remaining strings of the form $1^k y$ with $|y| = k$ out of A. If T_d rejects 1^k, then fix all remaining strings of the form $1^k y$ with $|y| = k$ to be in A. In either case, set $l = p_d(k)$, increase d by one, and continue to stage $k+1$.

End of stage k.

We first show that the construction can be carried out.

LEMMA 5.2. *If the construction simulates T_d on input 1^k, then some string $1^k y$ with $|y| = k$ remains unfixed at the end of the simulation.*

Proof. The parameter l is an upper bound on the length of strings that have been fixed due to any diagonalization attempt prior to stage k. Because $k \geq l$ at the start of the simulation of T_d, all of the 2^k strings $1^k y$ are unfixed at that time.

Let $S(n)$ be the maximum number of strings fixed during the simulation of some $M_i^A(x)$ for s steps, where $|0^{2s}\#i\#x| = n$. In s steps, $M_i^A(x)$ queries fewer than $s2^{\log|x|} < n^2$ strings over all of its nondeterministic computations, and each queried string has length less than $n/2$. Hence $S(n) < n^2 S(n/2)$, and $S(1) = 1$. Therefore, $S(n) \leq n^{\log n}$.

T_d runs for at most $p_d(k)$ steps; therefore, the total number of strings queried during the simulation is at most $p_d(k) \cdot S(p_d(k)) \leq p_d(k)^{1+\log p_d(k)} < 2^k$. Hence some string of the form $1^k y$ with $|y| = k$ remains unfixed. $\square$

LEMMA 5.3. *The set A constructed as above satisfies conditions C_i and D_i for all i.*

Proof. Each string of the form $0^{2s}\#i\#x$ is eventually fixed in or out of A. Such a string is fixed only after the acceptance or rejection of $M_i^A(x)$ within s steps is determined, and is fixed according to that acceptance or rejection. Hence condition C_i holds for all i.

Because $p_d(n)^{1+\log p_d(n)} = o(2^n)$, each value of d is eventually considered at some stage k. The previous lemma ensures that condition D_d is established following the completion of stage k. $\square$

Conditions C_i together imply that $N^1P_1^A = P_1^A$, and hence $N^mP_k^A = P_k^A$ for all k and m by Theorem 2.2. Conditions D_i together imply that $L^A \in NP^A - P^A$. $\square$

Informally, Theorem 5.1 shows that large amounts of nondeterminism can be useful even if small amounts are not. The reverse is also possible—a large amount of nondeterminism may be no more advantageous than a small amount.

THEOREM 5.4. *There is an oracle B such that $P_m^B \neq N^1P_m^B$ and $N^1P_m^B = NP_m^B$ for all m.*

Proof. The proof depends in a small way on the definition of an oracle machine. There are two possibilities to be considered. If the query tape is erased following a query, define $\delta = 0$. If the query tape is not erased following a query, define $\delta = 1$. Although the relativized classes depend on the model used (see [5]), the necessary properties for the present proof are captured by the value of δ.

Let $\{M_{l,i}\}_{l,i=1}^{\infty}$ be an enumeration of nondeterministic oracle machines, and let $\{T_{l,i}\}_{l,i=1}^{\infty}$ be an enumeration of deterministic oracle machines, such that both $M_{l,i}$ and $T_{l,i}$ are clocked to run for $p_{l,i}(n) = n^l \log^i n + i$ steps. We construct B to meet the following conditions:

1. *Coding conditions.* For all l and i, for all sufficiently large x, $M_{l,i}^B(x)$ accepts if and only if there is a string y of length $(1+\delta l)\log|x|$ such that $0^{p_{l,i}(|x|)}\#l\#i\#x\#y$ is in B.

2. *Diagonalization conditions.* For all l, the language

$$L_l = \{\, x \mid \exists y\, |y| = (1+\delta l)\log|x| \wedge 1^{|x|^l} xy \in B \,\}$$

is not accepted by $T_{l,i}^B$ for any i.

Condition 1 implies that $NP_l^B \subseteq N^1P_l^B$ for all l, because $\log|x|$ bits of y can be guessed nondeterministically, and if $\delta = 1$, the remaining $l \log|x|$ bits found by exhaustive search in time $O(n^l)$. (If a string is erased upon being queried, each query of a coding string requires $Q(n^l)$ steps, and the search cannot be carried out deterministically.) Condition 2 implies that $L_l \in N^1P_l^B - P_l^B$.

Let $\langle \cdot, \cdot \rangle$ be a pairing function from positive integers to positive integers; e.g., let $\langle l, i \rangle = (l^2 + i^2 + 2li - 3l - i + 2)/2$. To start the construction, set $k = d = b = 1$; k is the stage number, d the current diagonalization condition, and b the barrier so that diagonalizations do not overlap. Throughout, l and i will be determined by $d = \langle l, i \rangle$. Once again, at stage k, all strings of length k will be fixed in or out of B.

Stage k: If possible, pick an integer $\hat{k}$ such that $b < \hat{k} < k^{1/l}$, $p_{l,i}(\hat{k}) < \hat{k}^{l+(1/d)}$, and $\hat{k}^l + \hat{k} + (1+\delta l)\log\hat{k} \geq k$. If $\hat{k}$ exists, then a diagonalization condition can be satisfied. Otherwise, continue immediately to the coding part of the stage.

To satisfy the diagonalization condition, simulate T_d on input $1^{\hat{k}}$. If T_d queries a string not yet fixed, fix that string to be out of B. (This may fix a coding string. See below.) At the end of the simulation, if $T_d^B(1^{\hat{k}})$ accepts, fix all strings of the form $1^{\hat{k}^l+\hat{k}}y$ with $|y| = (1+\delta l)\log k$ to be out of B. If $T_d^B(1^{\hat{k}})$ rejects, fix all remaining strings $1^{\hat{k}^l+\hat{k}}y$ with $|y| = (1+\delta l)\log k$ to be in B. Set $b = p_{l,i}(\hat{k}) + 1$, and increase d by one.

After the diagonalization condition is considered, satisfy the possible coding conditions up to length k, as follows. For all unfixed strings z of length k having the form $0^{p_{m,j}(|x|)}\#m\#j\#x\#y$ for some m, j, x, and y with $|y| = (1+\delta m)\log|x|$, fix z in B if $M_{m,j}^B(x)$ accepts and fix z out of B if $M_{m,j}^B(x)$ rejects.

End of stage k.

For each fixed l, i, and d, the conditions on $\hat{k}$ are satisfied for infinitely many values of k. Therefore, each diagonalization condition will eventually be satisfied, and $L_l \in N^1P_l^B - P_l^B$ for all l.

Coding strings may be fixed during diagonalization stages. However, for each fixed m and j, there is a bound on the size of x such that a coding string for $M_{m,j}(x)$ may be fixed during a diagonalization. A given coding string for $M_{m,j}$ may be fixed out of B during the diagonalization against at most one $T_{l,i}$, at some stage k. Since all strings of length less than k are fixed before stage k, the stage number satisfies $k \leq |0^{p_{m,j}(|x|)}\#m\#j\#x\#y| = Q(|x|^m)$. Note that $k < c|x|^{m+1/3}$ for some c and all sufficiently long strings x. At most $p_{l,i}(\hat{k})/k^{1-\delta}$ strings of length k are fixed during the diagonalization. The conditions on $\hat{k}$ imply that $p_{l,i}(\hat{k})/k^{1-\delta}$ is bounded by $\hat{k}^{l+(1/d)}k^{\delta-1} < k^{\delta+(1/dl)}$, which is less than $|x|^{1+\delta m}$ when $d > 3m$ and x is sufficiently long. Since d is nondecreasing and unbounded, each $M_{m,j}$ will be correctly coded for all but finitely many x, and condition 1 will be satisfied.

Therefore, the oracle B meets the required conditions. □

We expect that neither oracle A nor B represents the unrelativized case. The following result seems more likely.

THEOREM 5.5. *There is an oracle C such that $N^iP_j^C \neq N^kP_l^C$ for all distinct pairs (i,j) and (k,l). Also $P^C \neq NP^C$.*

Proof. Each class N^iP_j satisfies the following conditions.

1. Let X be any oracle, and let $A \in N^iP_j^X$. For each string x, to test if $x \in A$ requires only finite knowledge of X.

2. For all oracles X and Y that differ on only finitely many strings, $N^iP_j^X = N^iP_j^Y$.

3. For all oracles X, and for all sets A and B that differ on only finitely many strings, $A \in N^iP_j^X$ if and only if $B \in N^iP_j^X$.

4. Let X be any oracle, and let machine M accept A in $N^iP_j^X$. Then there are an oracle X^* and machine M^* running in N^iP_j such that
   - $N^iP_j^X = N^iP_j^{X^*}$,
   - M^* with oracle X^* accepts the language A, and
   - for all oracles Y and Z that differ on only finitely many strings, the language accepted by M^* with oracle Y differs from the language accepted by M with oracle Z on at most finitely many strings.

Conditions 1, 2, and 3 are immediate. The oracle X^* of condition 4 is simply $X \times \mathbf{N}$. Machine M^* acts as M except that a query s is replaced by $\langle s, n\rangle$, where n is the length of the input.

These properties permit the use of a result of Poizat [15].

LEMMA 5.6 (Poizat). *Let $\mathcal{S}$ and $\mathcal{T}$ be two classes that satisfy conditions* 1–4 *above, with the same mapping $X \mapsto X^*$. Suppose that there is an oracle X such that $\mathcal{S}^X \neq \mathcal{T}^X$. Then $\mathcal{S}^G \neq \mathcal{T}^G$ for all generic sets G.*

Hence one may reverse the order of quantifiers in the statement of the theorem.

LEMMA 5.7. *For all distinct pairs (i,j) and (k,l), there is an oracle C such that $N^iP_j^C \neq N^kP_l^C$.*

Proof (sketch). If $j \neq l$, the desired oracle is the set A of Theorem 5.1, because $P_j^A \neq P_l^A$. (An alternative proof may be obtained via a diagonalization using queries of length $n^{\max\{j,l\}}$ on inputs of length n.)

Suppose $j = l$ and $i < k$. For an oracle C, define $L_C = \{x \mid \exists y\, |y| = k \log x \wedge xy \in C\}$. A standard diagonalization constructs a C such that $L_C \notin P_{i+j}^C$. Because $L_C \in N^kP_l^C$, the desired result follows. □

The previous lemmas imply that $N^iP_j^C \neq N^kP_l^C$ for all generic oracles C. □

Theorems 5.1 and 5.4 indicate that mere equality or inequality among the classes N^mP_l is insufficient to determine whether $P = NP$. It is consistent either that small amounts of nondeterminism ($m \log n$ bits) are useful, but large amounts (polynomially many bits) are no more useful, or that small amounts of nondeterminism are useless, but large amounts are quite powerful. However, additional information about the complexity of these classes, such as in the hypothesis of Theorem 2.3, can suffice to decide $P =?\ NP$.

6. Conclusions and open problems. We have shown several problems to be complete for quasilinear time with restricted nondeterminism. Satisfiability of topologically ordered circuits with $k \log n$ inputs and ordered unravelling of Boolean formulas are each complete for N^kP_1, and a problem concerning context-free grammars is complete for N^1P_1. All of these complete problems require that the input be ordered appropriately. The corresponding unordered problems remain hard for N^kP_1 but do not appear to be in any class below N^kP_2. If random-access machines were used as the underlying model of computation instead of Turing machines, then the unordered problems would be in quasilinear time, but the hardness proofs would fail, due to our inability to simulate random access.

One motivation for this work, as yet unrealized, was to provide general lower bounds for problems of "nearly feasible" complexity. Existing lower bounds usually apply to restricted models of computation, such as one-tape Turing machines or constant-depth circuits, or prove only very large lower bounds, as in the proofs of hardness for exponential time. A proof that some problem S is hard for a particular P_k or N^jP_k ($k > 1$) is a general lower bound on the time complexity of problem S. Although S may already be known to be in P, and thus be considered "feasible," such a hardness proof would give concrete information about the practical complexity of S. The only results of this kind known to the authors are for certain pebble games [3].

This paper leaves many questions unanswered. Although the most dramatic question is whether the hypothesis of Theorem 2.3 holds, there are other, more approachable questions. There are several problems that are easily shown to be in Gurevich and Shelah's class NLT but are not known to be in QL. These include topological sorting, and the problem of generalized Boolean formulas (shown to be NLT-complete in [10]). Providing QL algorithms for these problems, or showing that none exist (i.e., that $NLT \neq QL$), would give us important information about the power of random access.

The fundamental question remains whether there are problems in P that can be computed more quickly with limited nondeterminism than without it. For instance, can one show a $Q(n^k)$ deterministic lower bound for $(k+1)$-clique, or for CSAT(k)?

Acknowledgments. The authors thank A. Klapper, N. Immerman, M. Groszek, A. Peterson, W. L. Ruzzo, and A. Selman for useful discussions of this material.

REFERENCES

[1] K. R. Abrahamson, J. A. Ellis, M. R. Fellows, and M. E. Mata, *On the complexity of fixed parameter problems*, in Proceedings of 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Washington, DC, 1989, pp. 210–215.

[2] ———, *Completeness for Families of Fixed Parameter Problems*, University of Victoria, Victoria, British Columbia, Canada, Report DCS-141-IR, 1990.

[3] A. Adachi, S. Iwata, and T. Kasai, *Some combinatorial game problems require $\Omega(n^k)$ time*, J. Assoc. Comput. Mach., 31 (1984), pp. 361–376.

[4] C. Álvarez, J. Díaz, and J. Torán, *Complexity classes with complete problems between P and NP-complete*, in Proceedings of Foundations of Computation Theory, Lecture Notes in Computer Science 380, Springer-Verlag, New York, 1989, pp. 13–24.

[5] J. F. Buss, *Relativized alternation and space-bounded computation*, J. Comput. System Sci., 36 (1988), pp. 351–378.

[6] S. R. Buss, personal communication.

[7] S. A. Cook, *Short propositional formulas represent nondeterministic computations*, Inform. Process. Lett., 26 (1987/88), pp. 269–270.

[8] J. Díaz and J. Torán, *Classes of bounded nondeterminism*, Math. Systems Theory, 23 (1990), pp. 21–32.

[9] J. G. Geske, *On the Structure of Intractable Sets*, Ph.D. thesis, Iowa State University, Ames, IA, 1987.

[10] Y. GUREVICH AND S. SHELAH, *Nearly linear time*, in Logic at Botik '89, Lecture Notes in Computer Science 363, A. R. Meyer and M. A. Taitslin, eds., Springer-Verlag, 1989, Berlin, New York, pp. 108–118.

[11] H. B. HUNT, III AND R. E. STEARNS, *The complexity of very simple Boolean formulas with applications*, SIAM J. Comput., 19 (1990), pp. 44–70.

[12] R. KAYE, *Characterizing some low complexity classes using theories of arithmetic*, M. Sc. thesis, University of Manchester, Manchester, U.K., 1985.

[13] C. M. R. KINTALA AND P. C. FISCHER, *Refining nondeterminism in relativized polynomial-time bounded computations*, SIAM J. Comput., 9 (1980), pp. 46–53.

[14] N. PIPPENGER, *Fast simulation of combinational logic circuits by machines without random-access storage*, in Proceedings of the Fifteenth Allerton Conference on Communication, Control, and Computing, 1977, pp. 25–33.

[15] B. POIZAT, $\mathcal{Q} = \mathcal{NQ}$?, J. Symbolic Logic, 51 (1986), pp. 22–32.

[16] C. P. SCHNORR, *Satisfiability is quasilinear complete in NQL*, J. Assoc. Comput. Mach., 25 (1978), pp. 136–145.

[17] M. J. WOLF, *Limited Nondeterminism in Parallel Models of Computation*, Ph.D. thesis, University of Wisconsin, Madison, WI, 1990.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 573–586, June 1993

PARALLEL COMPLEXITY OF THE CONNECTED SUBGRAPH PROBLEM*

LEFTERIS M. KIROUSIS[†‡], MARIA SERNA[§], AND PAUL SPIRAKIS[†‡¶]

Abstract. This paper shows that the problem of testing whether a graph G contains an induced subgraph of vertex (edge) connectivity at least k is P-complete for any fixed $k \geq 3$. Moreover, if $k_{\max}$ is the largest vertex (edge) connectivity of any subgraph of G, it is shown that unless P = NC there is no NC algorithm that approximates $k_{\max}$ within any approximation factor $\frac{1}{2} < c < 1$ (such an algorithm is by definition one that outputs a number in the interval $[ck_{\max}, k_{\max}]$). In contrast, it is known that the problem of finding the Tutte (triconnected) components of G (i.e., the maximal subgraphs of G such that for any four vertices in any of them, any two of these vertices can be connected by a path in G that avoids the other two) is in NC. On the positive side, it is shown, by proving extremal graph results, that the maximum k for which there is a k-edge-connected induced subgraph of G can be approximated in NC for any approximation factor strictly less than $\frac{1}{2}$ and that the same is true for vertex connectivity for any approximation factor strictly less than $\frac{1}{4}$.

Key words. graphs, connectivity, connected subgraphs, parallel algorithms, algorithms in NC, P-complete problems

AMS(MOS) subject classifications. 68Q15, 68Q20, 68Q25, 68R10, 05C40, 05C35

1. Introduction.

The *vertex* (respectively, *edge*) *connectivity* of a graph is defined to be the least number of vertices (respectively, edges) that must be removed from the graph in order to obtain a disconnected or trivial (i.e., with one vertex) graph. A graph G is said to be *k-vertex-connected* (respectively, *k-edge-connected*) if its vertex (respectively, edge) connectivity is at least k. We are interested here in the parallel complexity of testing whether a graph has an induced subgraph of vertex (or edge) connectivity at least k. We show these problems to be P-complete under logspace reductions for any fixed $k \geq 3$. Our results imply that to test whether a *k-block* (a maximal, under inclusion, induced subgraph that is k-vertex-connected) exists in a graph is P-complete. Matula [1978] characterizes the k-blocks of a graph and argues that they are an interesting alternative to cliques for graph-theory-based cluster analysis. Harary [1972] uses the term *n-connected components* for n-blocks.

A graph-theoretic notion similar to that of the 3-blocks is the notion of the *Tutte components*. The Tutte components of a graph G are usually defined to be the subgraphs of G obtained by a process of repetitively splitting the graph by using separation pairs of vertices. However, they can also be defined to be the maximal induced subgraphs of G that cannot be split apart by a pair of vertices, i.e., for any four vertices of the subgraph, any two of them must be connected by a path in G that avoids the other two. Notice that since the connecting path need not lie entirely in the subgraph, a Tutte component may not be a 3-vertex-connected graph on its own, and therefore it may not be a 3-block. For details see, e.g., Tutte [1966] or Hopcroft and Tarjan [1973]. The Tutte components are also called *triconnected components*, but we avoid this terminology to avert confusion

*Received by the editors August 8, 1990; accepted for publication (in revised form) March 19, 1992. This research was done during the visit of the second author to Patras University, and it was partially supported by a Spanish Research Scholarship, by the Ministry of Industry, Energy and Technology of Greece, and by the ESPRIT II Basic Research Actions Program of the European Community under contract 3075 (project ALCOM). A preliminary version of this paper was presented at the 30th Annual Symposium on Foundations of Computer Science, held October 30–November 1, 1989, in Research Triangle Park, North Carolina. The symposium was sponsored by the IEEE Technical Committee on Mathematical Foundations of Computing.

[†]Department of Computer Engineering and Informatics, University of Patras, 26500 Rion, Greece.
[‡]Computer Technology Institute, P.O. Box 1122, 261 10 Patras, Greece.
[§]Polytechnic University of Catalonia, 08028, Barcelona, Spain.
[¶]Courant Institute of Mathematical Sciences, New York University, New York, New York 10012.

with the term "3-connected components" of Harary. In a possibly unexpected contrast to 3-blocks, the global characterization of Tutte components permits the construction of algorithms in NC for locating them or for testing whether at least one nontrivial such component exists (see, e.g., Miller and Ramachandran [1987] and the earlier works by Miller and Reif [1985] and JaJa and Simon [1982]). Although the definition of k-blocks (and 3-blocks especially) as maximal k-vertex-connected subgraphs is traditional, the Tutte components helped in characterizing planar graphs (see Bruno, Steiglitz, and Weinberg [1970]); thus they received more attention in the computer science literature.

Since the testing of whether a graph G contains a k-block (or a k-edge-connected maximal induced subgraph) is P-complete for $k \geq 3$, we lowered our sights and asked for efficient approximations (in NC) of the optimization versions of these problems. The optimization version is to find the maximum k for which there is a k-vertex-connected (or k-edge-connected) subgraph of G. Approximating this maximum k within a factor of $c(c \in (0,1))$ is defined to mean computing a k' such that $ck \leq k' \leq k$. We prove that unless P = NC no such approximations exist in NC within a factor c for any $c > \frac{1}{2}$, whereas we provide approximations in NC within a factor c for any $c < \frac{1}{2}$ for edge connectivity and for any $c < \frac{1}{4}$ for vertex connectivity. Our results increase the set of combinatorial P-complete problems whose approximations exhibit a *threshold behavior*. The first such problem, that of the high-degree subgraph, was posed by Anderson and Mayr [1984]. The high-degree subgraph problem consists of finding the largest d for which there is an induced subgraph S of G such that all nodes in S have degree $\geq d$ (with respect to S). Anderson and Mayr showed that approximating this maximum d within a factor $c < \frac{1}{2}$ is in NC, whereas it is not in NC for any factor $c > \frac{1}{2}$ unless P = NC. To obtain the analogous results for connectivity, we prove an extremal graph result that states that any graph with m edges and n vertices has an induced subgraph of vertex (respectively, edge) connectivity at least equal to $\lceil (m+n)/2n \rceil$ (respectively, $\lfloor m/(n-1) \rfloor$).

2. Parallel complexity of the vertex-connected subgraph problem.

2.1. Preliminaries. Let $\kappa(G)$ denote the vertex connectivity of a graph $G = (V, E)$. A *separating set* S of G is a vertex set $S \subseteq V$ such that $G - S$ is disconnected or trivial. A *minimum separating* set of G is a subset $S \subseteq V$ such that $|S| = \kappa(G)$. It is easy to see (see also, Matula [1978]) the following.

LEMMA 1. *Let $S \subseteq V$ be a minimum separating set of a noncomplete graph G, and let $\{A, B\}$ be a partition of the vertices of $G - S$ into two nonempty sets so that any path from a vertex in A to a vertex in B contains a vertex in S. Then for any $k > |S|$ the vertices of any k-block of G are contained in exactly one of the sets $A \cup S$ or $B \cup S$.*

Proof. Notice that if a k-block K intersects both A and B, then $K-S$ is disconnected. Now since $K - S = K - (K \cap S)$, we have that the connectivity of K is at most equal to the cardinality of S, a contradiction, because K is k-connected. □

The k-vertex-connected subgraph (k-VCS) problem is the following: given a graph $G = (V, E)$ and an integer k, does G contain an induced subgraph of vertex connectivity at least k?

2.2. P-completeness of the k-vertex-connected subgraph problem.

THEOREM 1. *The k-VCS problem has time complexity that is a polynomial function of $|G|$ and k.*

Proof. By Lemma 1 the procedure Test (G, k) given below answers the k-VCS problem.

```
**Procedure** Test($G, k$)
begin
if $G$ is $k$-connected return "true"
else if $G$ has $\leq k$ vertices return "false"
else do
    begin
    find a minimum separating set $S$ of $G$
    let $\{A, B\}$ be a partition of the vertices of $G - S$ such that
                any path from a vertex in $A$ to a vertex in $B$ contains a vertex in $S$
    return ((Test($A \cup S, k$)) $\vee$ (Test($B \cup S, k$)))
    end
end
```

Now let $T(n, k)$ be the complexity of the above procedure (n is the number of vertices of G). It is known (see, e.g., Even [1979]) that finding a minimum separating set of a graph G is polynomial in n. Therefore for some polynomial $p(n, k), T(n, k) = p(n, k) + T(|A| + k, k) + T(|B| + k, k)$ if $n > k$, whereas $T(n, k) = O(1)$ if $n \leq k$. Since $|A| + |B| + k = n$, it follows that $T(n, k)$ is polynomial. □

THEOREM 2. *For each $k \geq 3$ the k-VCS problem is P-complete under logspace reductions.*

Proof. We first prove the theorem for $k = 3$. The P-complete problem that we use for the reduction is the problem of computing the value of a Boolean circuit C with the following properties:

- C is monotone, i.e., the only gates in C are either OR or AND gates with in-degree 2.
- The out-degree of all gates and the input nodes is at most 2 (fan-out-2).
- There is a single output node of degree 1.
- There is at least one input with value 1.

The P-completeness of the above problem, known also as the monotone circuit value problem, was originally proved by Goldschlager [1977] (in the original proof it is not assumed that the fan-out is at most 2). For a proof of the P-completeness of the monotone, fan-out-2 circuit value problem see, e.g., Gibbons and Rytter [1988]. Assume now that C is as described above. For the reduction we generate a graph G_C as follows:

- Each input node of the circuit C as well as its output node are replaced by a $K_{2,2}$ graph, such as the one depicted in Fig. 1. We call the upper nodes of this gadget *in-nodes* and the lower ones *out-nodes*.
- Each OR gate of C is replaced by a copy of the graph depicted in Fig. 2. The four upper nodes of this gadget are called in-nodes, whereas, the lower ones are called out-nodes. The in-nodes are grouped in two pairs, each pair corresponding to an incoming edge of this gate.
- Each AND gate of C is replaced by a copy of the graph depicted in Fig. 3. Again, the upper nodes are the in-nodes, whereas the lower ones are the out-nodes, and the in-nodes are grouped in two pairs, each pair corresponding to the two incoming edges of the gate.
- An edge of the circuit connecting a gate (or input node) a to another gate (or output node) b is replaced by two parallel edges connecting the pair of out-nodes of the gadget that replaced a to one of the pairs of in-nodes of the gadget that replaced b (see Fig. 4). Different incoming edges to the same gate use different pairs of in-nodes, whereas different outgoing edges from the same gate use the same pair of out-nodes.

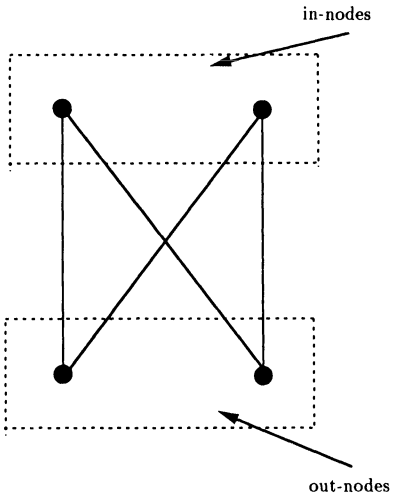


FIG. 1

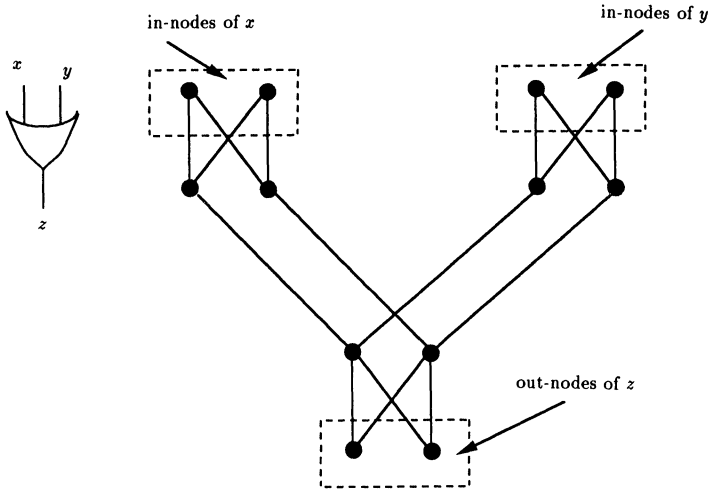


FIG. 2

• Besides the above modifications, a new node v_{new} is added as part of G_C, and this node is connected to the out-nodes of the gadget that replaced the output node of C and to all in-nodes of the gadgets that replaced those inputs of C that had value 1.

Notice that by the above construction an in-node in G_C has only one incoming edge, whereas an out-node, since the fan-out is at most 2, may have at most two outgoing edges (see Fig. 5). See Fig. 6 for an example of such a construction. It is not difficult to prove that the transformation of C described above can be carried out in logspace in the number of nodes of the graph. Indeed, it suffices to note that the transformation is a local one, i.e., at worst only the present node need be kept in the memory.

To prove now that the output of C is 1 if and only if G_C has a subgraph of vertex connectivity 3, we define an *elimination process*: Given a graph G, erase all nodes of

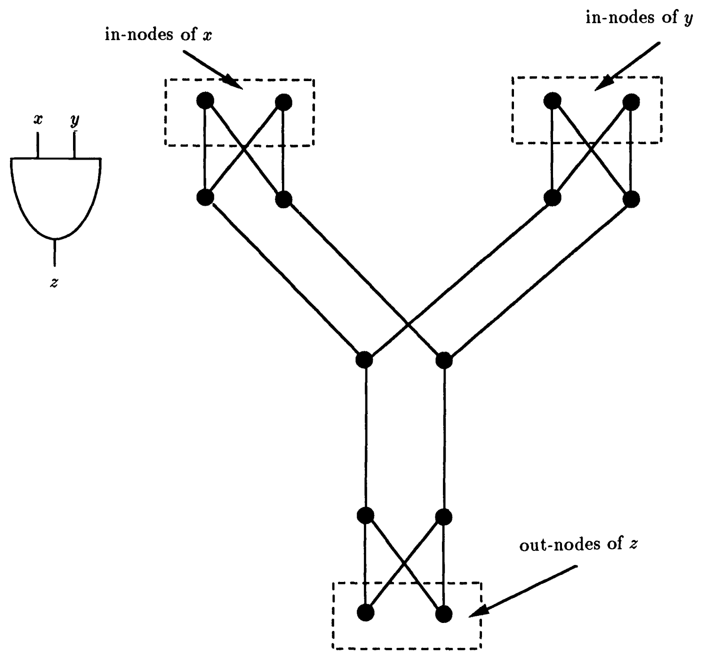


FIG. 3

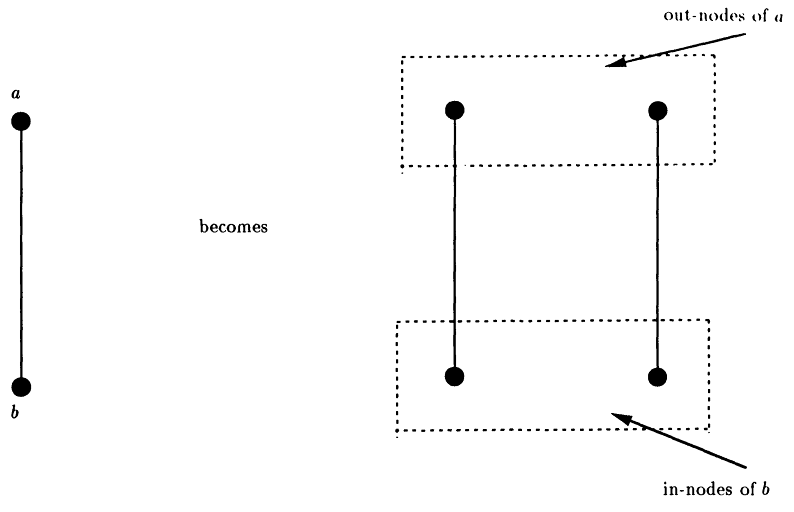


FIG. 4

degree <3 and their incident edges, and repeat this procedure for the new graph thus obtained until the graph contains either only nodes of degree ≥ 3 or no nodes at all. Then the following claims are true.

CLAIM 1. *Given a graph G, if by application of the elimination process the whole graph disappears, then G does not contain a 3-connected induced subgraph.*

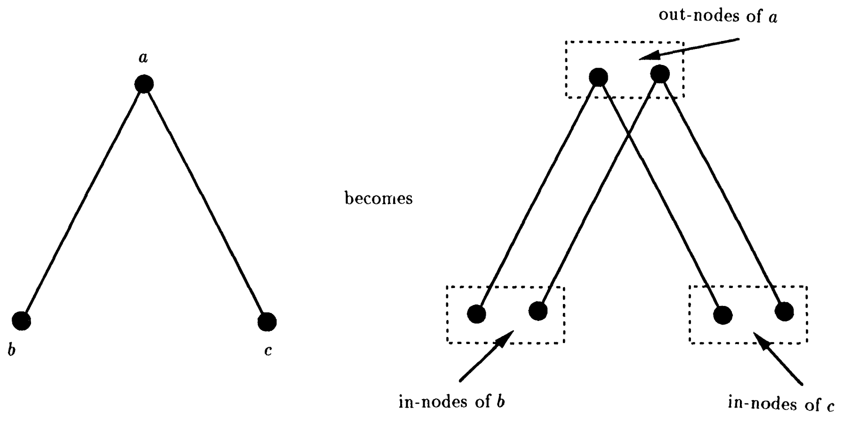

a
b
c
becomes
out-nodes of a
in-nodes of b
in-nodes of c

FIG. 5

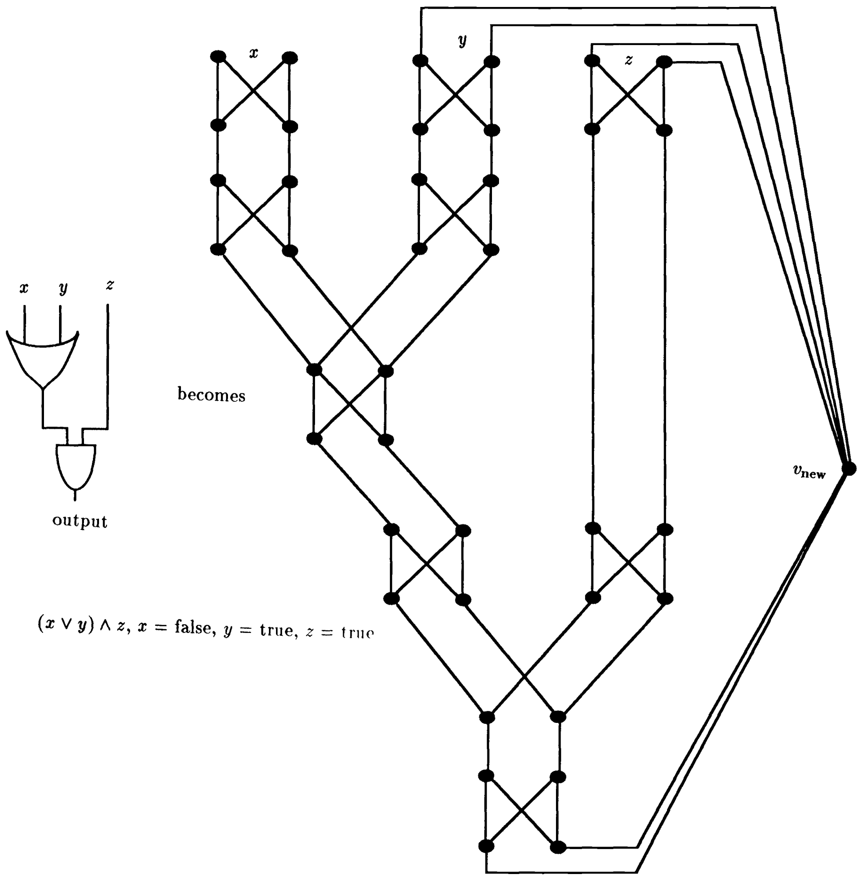

x
y
z
x y z
output
becomes
v_{new}
$(x \vee y) \wedge z$, x = false, y = true, z = true

FIG. 6

The claim immediately follows from the fact that no 3-connected subgraph contains a node with degree <3.

CLAIM 2. *If the output of the monotone fan-out-2 circuit C is 0, then the whole graph G_C is eliminated by the elimination process; hence in this case G_C does not contain a 3-vertex-connected subgraph.*

Proof of Claim 2. Because v_{new} is connected only with the in-nodes of gadgets replacing input nodes with value 1, the in-nodes of gadgets replacing inputs of value 0 are eliminated by the elimination process. As a consequence, and since the fan-out is at most 2, the out-nodes of these gadgets are eliminated as well. Now, because of the way the AND and OR gadgets were constructed and by an easy induction on the longest distance of a gate from an input node, it follows that also all nodes of gadgets corresponding to value-0 gates are eliminated. Hence the output gadget is eliminated. To show now that nodes of gadgets corresponding to a value-1 gate a are eliminated, we use induction on he longest path that starts from a and ends at an immediate predecessor either of a value-0 gate or of the output and contains exclusively value-1 gates. Similarly, we show that nodes corresponding to gadgets of inputs with value 1 are eliminated. Therefore, v_{new} and hence the whole graph is eliminated. This proves Claim 2.

CLAIM 3. *If the output of C is 1, then G_C contains a 3-connected subgraph.*

Proof of Claim 3. Let G'_C be the subgraph of G_C induced by the set of nodes containing

- The node v_{new};
- All nodes in paths that start from an in-node of a gadget corresponding to an input with value 1 and end at an out-node of the output gadget and pass through nodes of gadgets of 1-result gates only.

The graph G'_C is 3-connected. Indeed, if any two nodes different from v_{new} are subtracted from G'_C, any two remaining nodes can be connected through v_{new}. If, on the other hand, v_{new} together with one more node are subtracted from G'_C, any two remaining nodes can still be connected by establishing a path through the gadgets that avoids the single subtracted node. This completes the proof of Claim 3 and the proof of Theorem 2 for the case $k = 3$.

For the case $k > 3$ the same proof works except that we have to modify the gadgets so that all copies of $K_{2,2}$ (even those that appear as parts of the gadgets of AND and OR gates) are replaced by copies of $K_{k-1,k-1}$ and all pairs of parallel edges (either appearing inside the gadgets or replacing edges of the circuit C) are replaced by a set of $k-1$ parallel edges. □

2.3. Approximations to the vertex-connected subgraph problem. The optimization version of k-VCS asks, What is the maximum k such that there is an induced subgraph of vertex connectivity k? Let max-VCS(G) denote this largest k. An approximate solution to this problem is to find a k' such that

$$\text{max-VCS}(G) \geq k' \geq c(\text{max-VCS}(G))$$

for some fixed $c < 1$. It will be shown that this approximation problem cannot be solved in NC for $c > \frac{1}{2}$ unless P = NC. However, max-VCS(G) can be approximated in NC for $c < \frac{1}{4}$. To show this we first prove an extremal graph result (Theorem 3 below). Of course, since by Theorem 1 the k-VCS(G) problem can be answered in time polynomial with respect to $|G|$ and k, computing max-VCS(G) is polynomial in $|G|$.

THEOREM 3. *If a graph has n vertices and m edges $(m \neq 0)$, then it has an induced subgraph that is $\lceil (m+n)/2n \rceil$-vertex-connected.*

It suffices to apply the following lemma for $p = \lfloor m/n \rfloor$ and to observe that

$$\lceil (\lfloor m/n \rfloor + 1)/2 \rceil = \lceil ((m/n) + 1)/2 \rceil .$$

LEMMA 2. *Let G be a graph with n vertices and m edges, where $n \geq 2$. For any integer p, if*

$$p(n - (p+1)/2) \leq m \quad \text{and} \quad 0 < p < n,$$

then G has an induced subgraph that is $\lceil (p+1)/2 \rceil$-vertex connected.

Proof. The proof is by induction on the number of vertices. The induction basis ($n = 2$) is trivial to check. As the inductive hypothesis now, suppose that the lemma is true for graphs with $<n$ vertices, where $n > 2$. We will now prove the result for a graph G with n vertices. Observe first that if $p = n - 1$, then by the first inequality in the hypothesis the graph is complete; therefore, the required follows immediately. So we may assume that $p < n - 1$. Suppose first that G has minimum degree at most p. In this case let G' be the graph obtained from G by deleting a vertex with minimum degree. Because $p < n-1$, the second inequality in the statement of the lemma is satisfied for G'. To show that the first inequality is also satisfied, observe that for G' the right-hand side of the inequality is equal to m decreased by the degree of the deleted vertex, whereas its left-hand side is reduced by p. The required follows because we have assumed that the minimum degree of G is at most p.

Suppose now that the minimum degree of G is $>p$. Let S be a minimum separating set of G, and let $\{A_1, A_2\}$ be a partition of the vertices of $G - S$ in two sets so that no vertex in A_1 is connected to a vertex in A_2 by a path that avoids the vertices in S. Let n_i (respectively, m_i) be the number of vertices (respectively, edges) of the graph induced by the vertices in $A_i \cup S$. Also, let n_S (respectively, m_S) be the number of vertices (respectively, edges) of the graph induced by S. Since G has minimum degree $>p$, it follows that $p < n_i$ for $i = 1, 2$. Therefore, by the induction hypothesis and by the first inequality in the statement of the lemma it follows that for $i = 1, 2$ either $A_i \cup S$ has a $\lceil (p+1)/2 \rceil$-vertex-connected subgraph or $p(n_i - (p+1)/2) > m_i$. Since a $\lceil (p+1)/2 \rceil$-vertex-connected subgraph of $A_i \cup S$ is also a $\lceil (p+1)/2 \rceil$-vertex-connected subgraph of G, we may assume that

$$(1) \qquad p(n_i - (p+1)/2) > m_i \quad \text{for } i = 1, 2.$$

Now taking into account that

$$n_1 + n_2 = n + n_S$$

and that

$$m_1 + m_2 = m + m_S,$$

and summing up the inequalities (1), we obtain

$$p(n + n_S - (p+1)) > m + m_S.$$

Subtracting from this last inequality the inequality $p(n - (p+1)/2) \leq m$ given by the hypothesis, we obtain

$$n_S \geq \frac{m_S}{p} + \frac{p+1}{2},$$

from which it follows that the minimum separating set of G has cardinality $\geq(p+1)/2$ and hence that G itself is at least $\lceil(p+1)/2\rceil$-vertex-connected. □

THEOREM 4. *For any $c < \frac{1}{4}$ an approximation of* max-VCS(G) *can be found in* NC *with approximation factor c.*

Proof. We first define a routine Test(k) that returns an answer that is either "the graph has no subgraph of vertex connectivity $\geq k$" or "the graph has a subgraph of vertex connectivity $\lceil ck\rceil$." The routine Test(k) discards all vertices of degree $<k$ from the graph until either the graph is empty of fewer than $(1-4c)n'$ vertices have degree $<k$, where n' is the number of vertices currently in the graph. It is easy to check that this routine can be implemented on an Exclusive Read Exclusive Write (EREW) parallel random-access machine (PRAM) with n processors in $O(\log^2 n)$ time (n is the number of nodes originally in the graph). Indeed, since a constant fraction of the existing vertices is deleted at each phase, there are $O(\log n)$ deletion phases. Each such phase requires $O(\log n)$ time to compute the fraction of the vertices with degree $<k$. Observe, however, that the $O(\log^2 n)$ complexity function depends on c.

Now if the routine stops with an empty graph, then G cannot have a subgraph of vertex connectivity at least k. If, on the other hand, Test(k) stops with n' vertices, where $n' \neq 0$, then at least $4cn'$ of these vertices have degree $\geq k$, and therefore the remaining graph will have at least $2ckn'$ edges. Therefore, by Theorem 3, G will have a subgraph of vertex connectivity at least $\lceil ck+\frac{1}{2}\rceil \geq \lceil ck\rceil$.

By now applying Test(k) in parallel for all $k = 1, \ldots, n$, a k_0 will be found such that G has no subgraph of connectivity $\geq k_0$ but has a subgraph of connectivity at least $\lceil ck_0\rceil$. Therefore,

$$k_0 \geq \text{max-VCS}(G) \quad \text{and} \quad \lceil ck_0\rceil \leq \text{max-VCS}(G).$$

It suffices now to choose as an approximation to max-VCS(G) the number $\lceil ck_0\rceil$. The approximation factor then is c, and the complexity of the algorithm depends on it. □

THEOREM 5. *If* P $\neq$ NC, *then* max-VCS(G) *cannot be approximated in* NC *by any factor $c > \frac{1}{2}$.*

Proof. Suppose, towards a contradiction, that such an approximation exists for a $c > \frac{1}{2}$. Given any $k \geq 2$, we will provide a logspace transformation that transforms a monotone Boolean circuit C satisfying the restrictions described in the proof of Theorem 2 to a graph G_C such that max-VCS$(G_C) = 2k$ if the output of C is 1 and max-VCS$(G_C) = k+1$ if the output of C is 0. Then it is easy to see that if we choose k so that $k+1 < 2ck$, we could decide in NC the output of C, a contradiction. To make the figures simpler, we give the transformation only for the case $k = 3$. To generalize the result to arbitrary $k \geq 2$, we substitute into Figs. 7–10 each group of three vertices with a group of k vertices and, moreover, we add the necessary edges so as to have the same type of connection between two groups of vertices. Also, in Fig. 9 not only do we change the cardinality of each group of vertices but we also make the number of groups equal to k.

The graph G_C is generated according to the following rules:

- Each input node of C and the output node of C are replaced by 3 (k in the general case) nodes totally disconnected from one another (Fig. 7).

FIG. 7

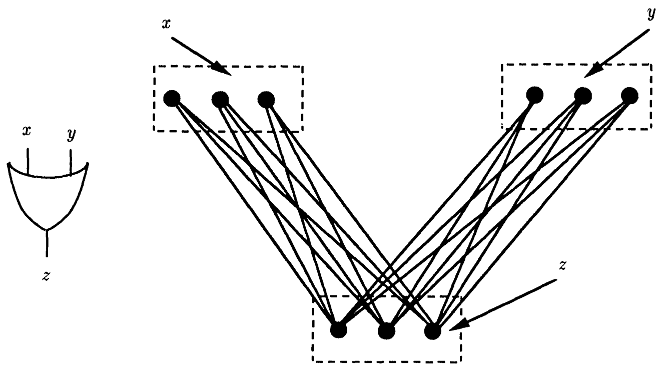


FIG. 8

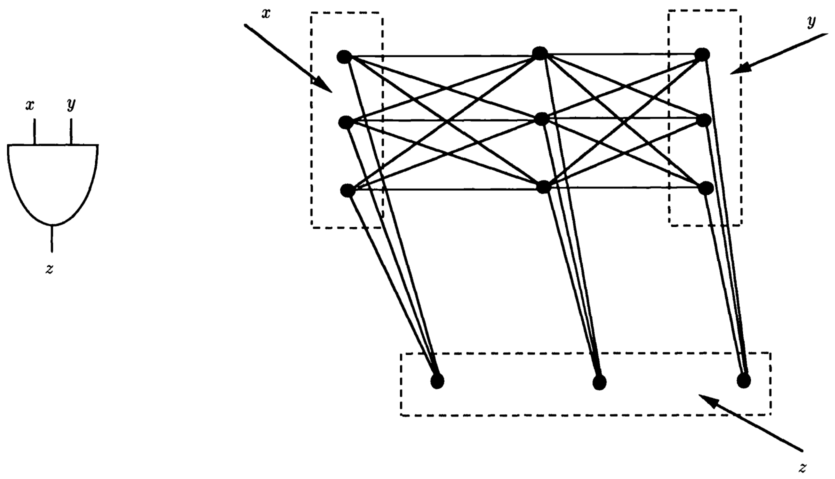


FIG. 9

• Each OR gate of C is replaced by a copy of the graph depicted in Fig. 8. The upper nodes of this gadget are called in-nodes and are divided into two groups, each corresponding to an incoming edge. The lower nodes are called out-nodes.

• Each AND gate is replaced by a copy of the graph depicted in Fig. 9. Adding more (or fewer) layers to this graph, we can generalize to an arbitrary k.

• The fanning out of the value of a gate (or an input node) to (at most 2) other gates (or the output node) is accomplished by using the fan-out gadget depicted in Fig. 10. The nodes a_1, a_2, a_3 of this gadget are the out-nodes of the gadget corresponding to gate a, whose value must be fanned out to gates b and c, respectively. On the other hand, nodes b_1, b_2, b_3 and c_1, c_2, c_3 are the in-nodes of the gadgets of b and c, respectively. Notice that by using more (or fewer) layers in this fan-out gadget, we could have implemented fan-out greater than 2 (less than 2, respectively). See also Anderson and Mayr [1984].

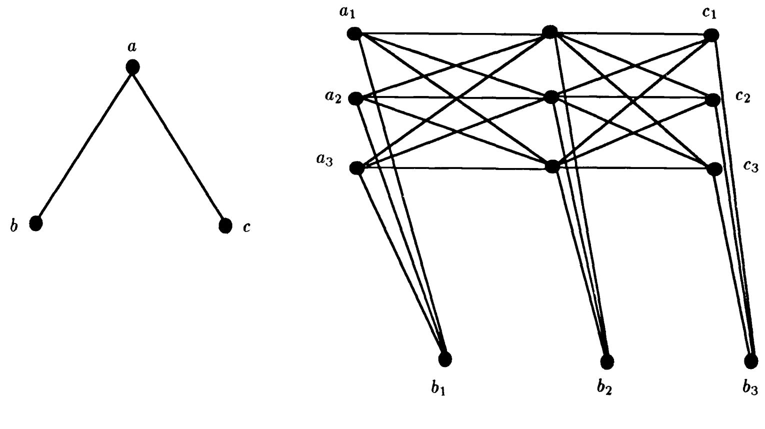


FIG. 10

• Finally, the nodes of the gadget corresponding to the output node of C are connected to the nodes of all value-1 input nodes of C by means of a multilayered fan-out gadget as described above.

Notice that thus every vertex in a group of out-nodes is connected by 4 ($k+1$ in the general case) edges to the nodes of the gadget leading to the corresponding in-nodes. Also, every vertex in a group of in-nodes is connected to the gadget leading to corresponding out-nodes with 3 or 4 (k or $k+1$ in the general case) edges.

Now it is not difficult to see that if all nodes of degree <6 ($<2k$ in the general case) are removed by an elimination process as described in the proof of Theorem 2, then all nodes of gadgets corresponding to nodes of C with value 0 will be removed. This is so because by construction the out-nodes of an OR gate are removed if and only if the vertices of both its groups of in-nodes are removed. Similarly, the vertices of a group of out-nodes of an AND gate are removed if and only if the vertices of at least one group of in-nodes are removed. Finally, the nodes of an input are removed if and only if either the input has value 0 or it has the value 1 and the output nodes are removed (this is so because only value-1 input nodes are connected to the output nodes).

Also, from the above we conclude that in case the output of C is 1, the nodes corresponding to gadgets of value 1 will remain. Actually, in this case the remaining graph will have vertex connectivity 6. Moreover, if all nodes of degree ≤ 6 are removed, everything is removed. This shows that if the output of C is 1, then max-VCS$(G_C) = 6$.

In case the output of C is 0, the removal of all gates of degree ≤ 4 ($\leq (k+1)$ in the general case) leaves nothing on the graph, whereas, since we have assumed that there is at least one input with value 1, there is a subgraph of G_C with vertex connectivity 4. This shows that if the output of C is 0, then max-VCS$(G_C) = 4$. $\square$

3. Parallel complexity of the edge-connected subgraph problem.

3.1. Preliminaries. Let $\lambda(G)$ denote the edge connectivity of a graph $G = (V, E)$. An *edge separating set* F of G is an edge-set $F \subseteq E$ such that $G - F$ is disconnected or trivial. An *l-edge-block* of G is a maximal, under inclusion, subgraph of G that is l-edge-connected. A *minimum edge separating set* of G is an edge separating subset $F \subseteq E$ such that $|F| = \lambda(G)$. By a proof completely analogous to that of Lemma 1 we can show the following.

LEMMA 3. *Let $F \subseteq E$ be an edge separating set of a graph G, and let $\{A, B\}$ be a partition of the vertices of G into two sets so that any path from a vertex in A to a vertex in B contains an edge in F. Then for any $l > |F|$ the vertices of any l-edge-block of G are contained in exactly one of the sets A or B.*

The l-edge-connected subgraph (l-ECS) problem is the following: given a graph $G = (V, E)$ and an integer l, does G contain an induced subgraph of edge connectivity at least l?

3.2. P-completeness of l-edge-connected subgraph problem.

THEOREM 6. *The l-ECS problem has time complexity that is a polynomial function of $|G|$ and l.*

Proof. By Lemma 3 the procedure Test(G, l) given below answers the l-ECS problem.

```
**Procedure** Test$(G, l)$
**begin**
**if** $G$ is $l$-edge-connected **return** "true"
**else if** $G$ has $<l$ edges **return** "false"
**else do**
    **begin**
    find a minimum edge separating set $F$ of $G$
    let $\{A, B\}$ be a partition of the vertices of $G$ such that
                any path from a vertex in $A$ to a vertex in $B$ contains an edge in $F$,
                and let $G_A(G_B)$ be the graphs induced by $A$ (respectively, $B$)
    **return** ((Test$(G_A, l)$) $\vee$ (Test$(G_B, l)$))
    **end**
**end**
```

Now let $T(n, l)$ be the complexity of the above procedure (n in the number of vertices of G). It is known (see, e.g., Even [1979]) that finding a minimum edge separating set of a graph G is polynomial in n. Therefore, for some polynomial $p(n, l)$

$$T(n, l) = p(n, l) + T(|A|, l) + T(|B|, l).$$

Since $|A| + |B| = n$, it follows that $T(n, l)$ is polynomial. □

THEOREM 7. *For each $l \geq 3$ the l-ECS problem is P-complete under logspace reductions.*

Proof. A proof completely analogous to the proof of Theorem 2 is sufficient for the following two reasons: (i) A graph containing a node with degree $<l$ cannot be l-edge-connected, and (ii) any l-vertex-connected subgraph is also l-edge-connected. □

3.3. Approximations to the edge-connected subgraph problem. The optimization version of l-ECS asks, What is the maximum l such that there is an induced subgraph of edge connectivity l? Let max-ECS(G) denote this largest l. As in the case of vertex connectivity, an approximate solution to this problem is to find an l' such that

$$\text{max-ECS}(G) \geq l' \geq c(\text{max-ECS}(G))$$

for some fixed $c < 1$. It will be shown that this approximation problem cannot be solved in NC for $c > \frac{1}{2}$ unless P = NC. However, max-ECS(G) can be approximated in NC for $c < \frac{1}{2}$. To show this we first prove an extremal graph result (Theorem 8 below). Of course, since by Theorem 6 the l-ECS(G) problem can be answered in time polynomial with respect to $|G|$ and l, computing max-ECS(G) is polynomial in $|G|$.

THEOREM 8. *If a graph has n vertices and m edges $(n > 1)$, then it has an induced subgraph that is $\lfloor m/(n-1) \rfloor$-edge-connected.*

The above theorem follows from the following lemma.

LEMMA 4. *Let G be a graph with n vertices and m edges $(n \geq 2)$. For any integer p if $pn \leq m + p$, then G has an induced p-edge-connected subgraph.*

Proof. The proof is by induction on the number of edges. Assume without loss of generality that $\lambda(G) < p$, and let F be a minimum edge separating set ($|F| = \lambda(G)$). Also let $\{A_1, A_2\}$ be a partition of the set of vertices of G in two sets so that no vertex in A_1 is connected to a vertex in A_2 by a path that avoids the edges in F, and let n_i (respectively, m_i) be the number of vertices (respectively, edges) of the subgraphs induced by A_i. Then $n_1 + n_2 = n$ and $m_1 + m_2 + \lambda(G) \geq m$. Now if $pn_1 \leq m_1 + \lambda(G)$, then $pn_1 \leq m_1 + p$ and the result follows from the induction hypothesis. So assume that $pn_1 > m_1 + \lambda(G)$. Then $m_2 \geq m - (m_1 + \lambda(G)) > m - pn_1$; therefore, $m_2 > m - p(n - n_2)$, and so

$$pn_2 < m_2 + pn - m \leq m_2 + p.$$

From the last inequality the required follows by the induction hypothesis. □

THEOREM 9. *For any $c < \frac{1}{2}$ an approximation of* max-ECS(G) *can be found in* NC *with approximation factor c.*

Proof. As in the case of vertex connectivity, we define a routine Test(l) that returns an answer that is either "the graph has no subgraph of edge connectivity $\geq l$" or "the graph has a subgraph of edge connectivity $\lfloor cl \rfloor$." The routine Test(l) discards all vertices of degree $<l$ from the graph until either the graph is empty of fewer than $(1-2c)n'$ vertices have degree $<l$, where n' is the number of vertices currently in the graph. Analogously to the case of vertex connectivity, it is easy to check that this routine can be implemented on an EREW PRAM with n processors in $O(\log^2 n)$ time (n is the number of nodes originally in the graph). If the routine stops with an empty graph, then G cannot have a subgraph of edge connectivity at least l. If, on the other hand, Test(l) stops with n' vertices, where $n' \neq 0$, then at least $2cn'$ of these vertices have degree $\geq l$ and therefore the remaining graph will have at least cln' edges. Therefore, by Theorem 8, G will have a subgraph of edge connectivity at least $\lfloor cln'/(n'-1) \rfloor \geq \lfloor cl \rfloor$.

Now by applying Test(l) in parallel for all $l = 1, \ldots, m$, an l_0 will be found such that G has no subgraph of edge connectivity $<l_0$ but has a subgraph of edge connectivity at least $\lfloor cl_0 \rfloor$. Therefore,

$$l_0 \geq \text{max-ECS}(G) \quad \text{and} \quad \lfloor cl_0 \rfloor \leq \text{max-ECS}(G).$$

It suffices now to choose as an approximation to max-ECS(G) the number $\lfloor cl_0 \rfloor$. The approximation factor is then c, and the complexity of the algorithm depends on it. □

THEOREM 10. *If* P $\neq$ NC, *then* max-ECS(G) *cannot be approximated in* NC *by any factor $c > \frac{1}{2}$.*

Proof. For reasons that were outlined in the proof of Theorem 7, it turns out that a proof completely analogous to the proof of Theorem 5 works for this case as well. □

4. Discussion.

We showed that the problem of approximating the largest l for which there is an l-edge-connected subgraph of a given graph is a problem that exhibits a threshold: for approximation factors strictly less than $\frac{1}{2}$ the problem is in NC, whereas for approximation factors strictly greater than $\frac{1}{2}$ it is P-complete. The case where the approximation factor is equal to $\frac{1}{2}$ is an open problem.

The situation is as follows for the corresponding problem for vertex connectivity: we have shown that the problem is P-complete for approximation factors strictly greater than $\frac{1}{2}$, whereas we have given an NC algorithm only for factors strictly less than $\frac{1}{4}$. We conjecture that for factors as great as (but not including) $\frac{1}{2}$, the approximation is in NC. This is so because we believe that Lemma 2 could be improved to cover these factors as well.

However, we believe it is improbable that the same techniques would yield an NC algorithm for the case in which the approximation factor is equal to $\frac{1}{2}$. This is so because then by continuity arguments we could find an NC algorithm for factors infinitesimally greater than $\frac{1}{2}$, thus contradicting the P-completeness result.

Acknowledgments. We wish to thank H. Jung, C. Papadimitriou, V. Ramachandran, J. Reif, and M. Yannakakis for their prompt feedback on preliminary work on the parallel complexity of the vertex-connected subgraph problem. Also, we thank the referees, whose remarks helped considerably in improving the paper.

REFERENCES

[1] R. ANDERSON AND E. W. MAYR [1984], *A P-complete problem and approximations to it*, Tech. Rep., Department of Computer Science, Stanford University, Stanford, CA.
[2] J. BRUNO, K. STEIGLITZ, AND L. WEINBERG [1970], *A new planarity test based on* 3-*connectivity*, IEEE Trans. Circuit Theory, 17, pp. 197–206.
[3] S. EVEN [1979], *Graph Algorithms*, Computer Science Press, New York.
[4] A. GIBBONS AND W. RYTTER [1988], *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, England.
[5] L. GOLDSCHLAGER [1977], *The monotone and planar circuit value problems are log space complete for P*, SIGACT News, 9 (2), pp. 25–29.
[6] F. HARARY [1972], *Graph Theory*, Addison-Wesley, Reading, MA.
[7] J. E. HOPCROFT AND R. E. TARJAN [1973], *Dividing a graph into triconnected components*, SIAM J. Comput., 2, pp. 135–158.
[8] J. JAJA AND J. SIMON [1982], *Parallel algorithms in graph theory: Planarity testing*, SIAM J. Comput., 11, pp. 314–328.
[9] D. W. MATULA [1978], *k-blocks and ultrablocks in graphs*, J. Combin. Theory Ser. B, 24, pp. 1–13.
[10] G. L. MILLER AND V. RAMACHANDRAN [1987], *A new graph triconnectivity algorithm and its parallelization*, Proc. 19th Annual ACM Symposium on Theory of Computing, ACM Press, New York, pp. 335–349.
[11] G. L. MILLER AND J. REIF [1985], *Parallel tree contraction and its applications*, in Proc. 26th IEEE Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 478–489.
[12] W. T. TUTTE [1966], *Connectivity in Graphs*, University of Toronto Press, Toronto, Canada.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 587–616, June 1993

FINDING TRICONNECTED COMPONENTS BY LOCAL REPLACEMENT*

DONALD FUSSELL†, VIJAYA RAMACHANDRAN‡, AND RAMAKRISHNA THURIMELLA§

Abstract. A parallel algorithm for finding triconnected components on a CRCW PRAM is presented. The time complexity of the algorithm is $O(\log n)$, and the processor-time product is $O((m+n)\log\log n)$, where n is the number of vertices and m is the number of edges of the input graph. The algorithm, like other parallel algorithms for this problem, is based on open ear decomposition, but it uses a new technique, local replacement, to improve the complexity. Only the need to use the subroutines for connected components and integer sorting, for which no optimal parallel algorithm that runs in $O(\log n)$ time is known, prevents the algorithm from achieving optimality.

Key words. parallel algorithm, triconnectivity, PRAM, graph, vertex connectivity

AMS(MOS) subject classifications. 05C40, 68Q22, 90B12

1. Introduction. A connected graph $G = (V, E)$ is k-vertex connected if it has at least $(k+1)$ vertices and removal of any $(k-1)$ vertices leaves the graph connected. Designing efficient algorithms for determining the connectivity of graphs has been a subject of great interest in the last two decades. Applications of graph connectivity to problems in computer science are numerous. Network reliability is one application: algorithms for edge and vertex connectivity can be used to check the robustness of a network against link and node failures, respectively. In spite of all the attention this subject has received, $O(m+n)$-time sequential algorithms for testing k-edge and k-vertex connectivity of an n-node, m-vertex graph are known only for $k \leq 3$ [5], [11]. Recently, Gabow has devised a very nice algorithm for edge connectivity. His algorithm, unlike previous algorithms for connectivity, does not appeal to Menger's theorem. It runs in $O(km\log(n^2/m))$ time [9]. The algorithms for vertex connectivity for $3 < k \leq \sqrt{n}$ currently require $O(k^2n^2)$ time [2], [13], [19].

The subject of this paper is the parallel complexity of 3-vertex connectivity. The importance of 3-vertex connectivity stems from the face that if a planar graph is 3-vertex connected (triconnected), then it has a unique embedding on a sphere. Hence an efficient algorithm that divides a graph into triconnected components is sometimes useful as a subroutine in problems such as planarity testing and planar graph isomorphism.

We present in this paper an algorithm, based on open ear decomposition, for dividing a biconnected graph into triconnected components. The model of computation used in this paper is a concurrent-read-concurrent-write (CRCW) parallel random-access machine (PRAM) in which write conflicts are resolved arbitrarily (the arbitrary-CRCW PRAM model). See [14] for a discussion on the PRAM model. Our algorithm runs

*Received by the editors March 15, 1990; accepted for publication (in revised form) March 6, 1992. A preliminary version of this paper was presented at the 16th International Colloquium on Automata, Languages, and Programming, Stresa, Italy, July 1989; cf. [8].

†Department of Computer Sciences, University of Texas, Austin, Texas 78712. The work of this author was supported by the Office of Naval Research under contracts N00014-86-K-0763 and N00014-86-K-0597.

‡Department of Computer Sciences, University of Texas, Austin, Texas 78712. The work of this author was supported in part by the Joint Services Electronics Program under contract N00014-85-C-0149 at the Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801, and by National Science Foundation contract CCR 89-10707.

§Department of Mathematics and Computer Science, University of Denver, Denver, Colorado 80208. The work of this author was supported in part by Office of Naval Research contracts N00014-86-K-0763 and N00014-86-K-0597. This work was done while the author was with the University of Texas at Austin.

in $O(\log n)$ time performing at most $O((m+n)\log\log n)$ work, where m and n are the number of edges and the number of vertices of the input graph, respectively.

The first optimal sequential triconnected component algorithm (based on depth-first search (DFS)) was given by Hopcroft and Tarjan in 1973 [11]. Several parallel algorithms (e.g., [12], [16]) have been developed since then for addressing the triconnected component problem by using techniques other than DFS since the question of finding a DFS spanning tree efficiently in parallel remains one of the major open problems in the area of parallel algorithm design. The algorithms in [12] and [16] use parallel matrix multiplication as a subroutine; hence their processor complexity is far from optimal. Significant progress has been made in recent years: first, Miller and Ramachandran [18] derived an $O(\log^2 n)$ parallel algorithm; later, Ramachandran and Vishkin [21] derived an algorithm with $O(\log n)$ parallel time for the more restricted problem of finding separating pairs. A drawback with both of these algorithms is that the work performed by them in the worst case is a $O(\log^2 n)$ factor off the optimal. Independent of [18] and [21], Fussell and Thurimella [7] came up with a parallel algorithm for finding separating pairs whose time complexity is $O(\log n)$ while the work performed is only a $O(\log n)$ factor off the optimal; detecting separating pairs forms the central part of any triconnected component algorithm.

The chief method used by [18] and [21] can be broadly classified as divide-and-conquer. Additional complexity improvements are unlikely with this approach because of the end-node sharing problem: the difficulty arising from two or more ears sharing an end vertex. A novel technique known as *local replacement* was introduced in [7] as a method for obtaining efficient parallel reductions.

Using the local replacement technique and building on the algorithm in [7], we obtain an algorithm for triconnected components. A new linear-time sequential algorithm, an alternative to the algorithm of Hopcroft and Tarjan, for finding triconnected components can be easily extracted from our paper. We remark that a different presentation of the results of this paper is available in [20].

2. Preliminaries. Let $V(G)$ and $E(G)$ stand, respectively, for the vertex set and the edge set of a graph G. Assume that $|V(G)| = n$ and $|E(G)| = m$. We denote an edge between x and y as (x, y) or simply xy. A connected graph G is *k-vertex connected* if $|V(G)| > k$ and at least k vertices must be removed to disconnect the graph. A *biconnected graph* (or a *block*) is a 2-vertex connected graph. A pair of vertices $\{x, y\}$ of a biconnected graph is a *separating pair* if the number of components of the subgraph induced by $V(G) - \{x, y\}$ is more than one. An *ear decomposition starting with a vertex* P_0 of an undirected graph G is a partition of $E(G)$ into an ordered collection of edge disjoint simple paths $P_0, P_1, \ldots, P_k$ such that P_1 is a simple cycle starting and ending at P_0, and for P_i, $1 \le i \le k$, each end vertex is contained in some P_j for some $j < i$ and no internal vertex of P_i is contained in any P_j, $j < i$. Each of these paths P_i is an *ear.* P_0 is called the *root* of the decomposition and is referred to as r. If the two end vertices of a path P_i do not coincide, then P_i is an *open ear.* In an *open ear decomposition* every ear P_i, $1 < i \le k$, is open.

THEOREM 2.1 (Whitney [26]). *A graph has an open ear decomposition if and only if it is biconnected.*

From the above theorem we can conclude that the subgraph induced by the vertices of the ears of $P_0, P_1, \ldots, P_i$ for all i, $1 \le i \le k$, is biconnected.

An ear is a *nontrivial ear* if it consists of more than one edge; otherwise, it is a *trivial ear.* For an ear P and two vertices x and y of P, $P[x, y]$ (respectively, $P(x, y)$) denotes the segment of P that is between x and y, inclusive (respectively, exclusive) of x and y.

The segments $P(x, y]$ and $P[x, y)$ of P are defined similarly. A vertex is *internal* to an ear if it is not one of the end vertices of that ear. For two vertices v and w on P, $P - P[v, w]$ refers to the segment(s) of P formed by $V(P) - V(P[v, w])$.

Because an ear decomposition is a partition on the edge set of a graph, each edge (v, w) belongs to a unique ear (denoted by $ear(vw)$). Notice that, except for the root, each v is internal to exactly one ear; call it $ear(v)$. Refer to Fig. 1 for an example of a biconnected graph and an open ear decomposition for it. The following definition labels each vertex v depending on the *position* of v on $ear(v)$.

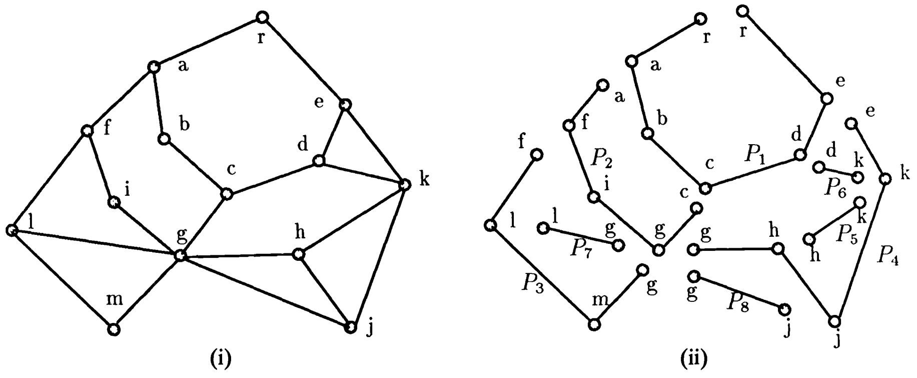


FIG. 1. (i) *A biconnected graph G with* 14 *vertices and* 21 *edges.* (ii) *Its open ear decomposition, where* $P_0 = \langle r\rangle$, $P_1 = \langle r, a, b, c, d, e, r\rangle$, $P_2 = \langle a, f, i, g, c\rangle$, $P_3 = \langle f, l, m, g\rangle$, $P_4 = \langle g, h, j, k, e\rangle$, $P_5 = \langle h, k\rangle$, $P_6 = \langle d, k\rangle$, $P_7 = \langle l, g\rangle$, *and* $P_8 = \langle g, j\rangle$.

Starting with an arbitrary end vertex p of P, define the position of p on P, $pos(p, P)$, to be zero. For every vertex v of P, $v \neq r$, the position of v on P, $pos(v, P)$ is equal to the number of edges between p and v on P. When a vertex v is an internal vertex of P, we omit the second argument and simply write $pos(v)$. The value of $pos(x, P)$ for $x \notin V(P)$ is undefined. Some example pos values for the graph of Fig. 1 are $pos(g) = pos(g, 2) = 3$, $pos(d, 6) = 0$, $pos(r, 1) = 0$, $pos(4, e) = 4$, and $pos(1, j)$ is undefined. For a pair of vertices u, v of P, u is to the *left* (respectively, *right*) of v if $pos(u, P) < pos(v, P)$ (respectively, $pos(u, P) > pos(v, P)$). The vertex of P that has no vertices of P to its left (respectively, right) is called the *left end vertex of P* (respectively, *right end vertex of P*).

For the sake of completeness we include the definition of bridges. Let $G = (V, E)$ be a biconnected graph, and let Q be a subgraph of G. We define the *bridges of Q in G* as follows (see, e.g., [5, p. 148]): Let V' be the vertices in $G - Q$, and consider the partition of V' into classes such that two vertices are in the same class if and only if there is a path connecting them that does not use any vertex of Q. Each such class K defines a *nontrivial bridge* $B = (V_B, P_B)$ of Q, where B is the subgraph of G with $V_B = K \cup \{$vertices of Q that are connected by an edge to a vertex in $K\}$ and P_B contains the edges of G incident on a vertex in K. The vertices of Q that are connected by an edge to a vertex in K are called the *attachments* of B, and these edges are called the *attachment edges* of B. An edge (u, v) in $G - Q$ with both u and v in Q is a *trivial bridge* of Q with attachments u and v. The trivial and nontrivial bridges together constitute the bridges of Q.

Remark. Throughout the paper we will address only how to detect pairs where r, the root of an ear decomposition, does not belong to $\{x, y\}$. The pairs in which one of the vertices is r can be detected as a special case by finding the articulation vertices of the graph induced by $V(G) - \{r\}$ within the claimed resource bounds.

LEMMA 2.2. *If $\{x, y\}$ is a separating pair of a graph G, then there exists a nontrivial ear P in any open ear decomposition of G such that $\{x, y\}$ is a pair of nonadjacent vertices on P.*

Proof. Let C be a connected component induced by $V(G) - \{x, y\}$ such that $r \notin V(C)$. Then P is the minimum$\{ear(v) | v \in V(C)\}$ since $\{x, y\}$ separates r from $v \in P - P(x, y)$. □

DEFINITION 1. An ear P *is separated by* $\{x, y\}$ if x and y are nonadjacent vertices of P and $P(x, y)$ is separated from the lower-numbered ears in $G - \{x, y\}$.

Notice that the number of ears separated by a pair of vertices $\{x, y\}$ is one fewer than the number of connected components of $G - \{x, y\}$. The location of separating pairs on an ear P can be stated precisely in terms of the attachments of the bridges of P.

THEOREM 2.3. *$\{x, y\}$ is a separating pair that separates ear P if and only if* (i) *there exists a nontrivial ear P containing x and y as nonadjacent vertices,* (ii) *the bridge B_r of P in G that contains r has no attachments on $P(x, y)$, and* (iii) *for all other bridges B of P in G, if B has an attachment on $P(x, y)$, then all attachments of B are on $P[x, y]$.*

Proof. ($\Longrightarrow$) Let P be an ear separated by $\{x, y\}$. Assume, for contradiction, that the forward implication is not true. From Definition 1 we know that x and y are not adjacent on P. For P either B_r has an attachment $P(x, y)$ or there is a B with one attachment on $P(x, y)$ and one on $P - P[x, y]$. In either case there is a path from one of the vertices of $P(x, y)$ to r in $G - \{x, y\}$, which contradicts the assumption that P is separated by $\{x, y\}$.

($\Longleftarrow$) There must be at least one vertex v of P between x and y since they are not adjacent on P. For all such v there cannot be a path in $G - \{x, y\}$ from v to r since that would imply a bridge of P in G with at least two attachments—one on $P(x, y)$ and the other on $P - P[x, y]$. Therefore, the segment $P(x, y)$ is disconnected from the components containing lower-numbered ears when x and y are deleted from G. □

Theorem 2.3 ensures that each ear P together with the bridges of P are sufficient for extracting separating pairs and that each ear with its bridges can be considered in isolation. But if we were to consider bridges of all ears in their entirety, the number of edges involved could be far more than $O(m)$. However, notice that for an ear P the information about its bridges that is of relevance in finding separating pairs is contained only in those edges of the bridges that are incident on P. We can succinctly encode the information about separating pairs by building a collection of simple graphs as shown in the following.

DEFINITION 2. The collection $\mathcal{H}$ consists of simple graphs, one for each nontrivial ear of G. The graph $H_P \in \mathcal{H}$ for the ear P is as follows. Suppose that x and y are the end vertices of P and that $pos(x) < pos(y)$. Each H_P is such that $V(H_P) = V(P) \cup \{r_P\}$ and $E(H_P)$ is as follows. (i) For each vertex $v \in V(P)$, an edge (v, r_P) is added if the bridge of P that contains the root r has an attachment at v. Otherwise, (ii) at most two edges are added to $E(H_P)$ by considering the bridges (possibly trivial) of P that have an attachment at v. Let a be the leftmost attachment of one such bridge where a is further to the left than the leftmost attachment for any other bridge that has an attachment at v. The edge (v, a) is added if a belongs to $P[x, v)$. Similarly, an edge (v, b) is added where b is the rightmost vertex that can be reached from v through a bridge of P.

An example of an ear P with its bridges and its corresponding H_P is illustrated in Fig. 2. Suppose that v and w are on P with $pos(v) < pos(w)$. From the definition of H_P it follows that there is a bridge of P in G with one attachment on $P(v, w)$ and another on $P - P[v, w]$ if and only if there is a bridge of P in H_P with one attachment on $P(v, w)$ and another on $P - P[v, w]$. Hence by Theorem 2.3 we have the following corollary.

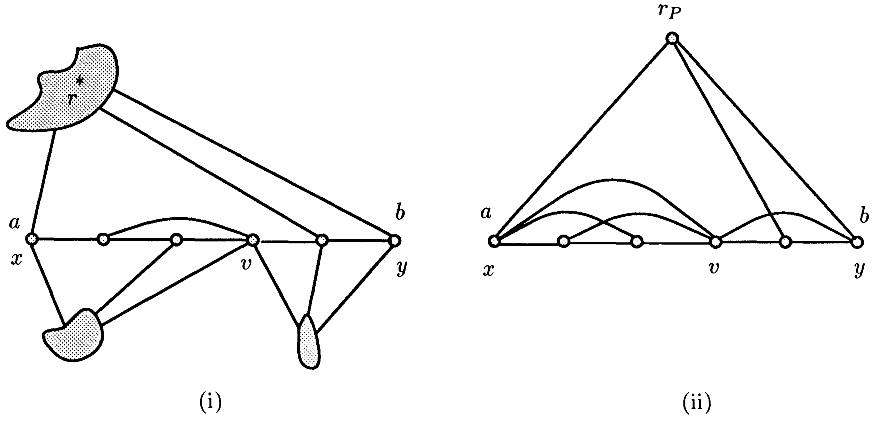


FIG. 2. (i) *An ear P and the bridges of P in G.* (ii) *The graph H_P.*

COROLLARY 2.4. *$\{v, w\}$ is a separating pair that separates P if and only if $\{v, w\}$ is a separating pair in H_P.*

3. Algorithm for separating pairs. Lemma 2.2 tells us that in our search for separating pairs we do not have to consider those vertex pairs for which there is no single ear containing them. If we further assured somehow that *every* separating pair belongs to the *internal* vertices of *some* ear, then we can efficiently reduce the problem of finding separating pairs to that of finding biconnected components. The idea is to build a multigraph by collapsing the internal vertices of every nontrivial ear to a single vertex. We elaborate more on this reduction in §3.2. But before we proceed to this reduction, we need to address a more serious difficulty: for a separating pair $\{x, y\}$ there *need not* be any ear in an open ear decomposition, for which x and y are internal. Section 3.1 shows how to circumvent this problem. In §3.2 we show how to find a collection of graphs similar to $\mathcal{H}$ that succinctly encode separating pairs. Finally, in §3.3 we address the extraction and the output representation of separating pairs.

3.1. Making separating pairs internal to an ear. In this subsection we show how to build a graph G', the *local replacement graph.* This graph is such that for every separating pair $\{x, y\}$ that separates P in G there is a corresponding separating pair $\{x_P, y_P\}$ and an ear P' in G' such that x_P and y_P are internal vertices of P'. Furthermore, for every separating pair $\{x_P, y_Q\}$ of G', $\{x, y\}$ is a separating pair in G. Roughly, the strategy is to divide the graph G into a set of paths by splitting the end vertices of the nontrivial ears of an open ear decomposition. This would result in making several copies of a vertex, i.e., one copy for each nontrivial ear that contains it. Next we add new edges to connect up the different copies of a vertex. The main difficulty is in figuring out an efficient way of connecting the split vertices without jeopardizing the primary goal of preserving the overall structure of separating pairs.

DEFINITION 3. (i) Define $\vec{G}$ to be a directed acyclic version of G that is obtained by the following construction. Suppose (a, b) is one of the end edges of P_1 and that (b, c) is next to (a, b) on P_1. Then give a direction to (a, b). Orient the rest of P_1 in the opposite direction. Now $\vec{G}$ is obtained by giving directions to the remaining ears of G so that the resulting digraph is acyclic.

(ii) Define $\overrightarrow{T}$ to be the directed spanning tree rooted at a obtained by removing the last edge in each directed ear P_i, $i > 1$, in $\overrightarrow{G}$ and deleting (b, c) in the case of P_1.

(iii) Define $\overleftarrow{T}$ to be the directed spanning tree rooted at b obtained as shown below. Let $\overleftarrow{G}$ be the graph resulting from reversing the directions of the edges of $\overrightarrow{G}$. Then $\overleftarrow{T}$ is obtained by deleting the last edge of each ear P_i, $i > 1$, in $\overleftarrow{G}$ and deleting the end edge of P_1 other than (a, b), in the case of P_1.

Figure 3 illustrates the above definitions. One efficient way of obtaining $\overrightarrow{G}$ for a given G, is by st-numbering the vertices of G. (Recall that for a biconnected graph $G(V, E)$ and an edge (s, t) of G, a 1–1 function $g : V \to \{1, 2, \ldots, |V|\}$ is called an *st-numbering* if (i) $g(s) = 1$, (ii) $g(t) = |V|$, and (iii) for every $v \in V - \{s, t\}$ there are adjacent vertices u and w such that $g(u) < g(v) < g(w)$.) For example, $\overrightarrow{G}$ can be obtained by assuming that (s, t) is an edge of P_1 with $s = r$ and directing each edge of an ear from the lower-numbered vertex to the higher-numbered one. Since each end vertex of every nontrivial ear belongs to a lower-numbered ear, we have the following proposition.

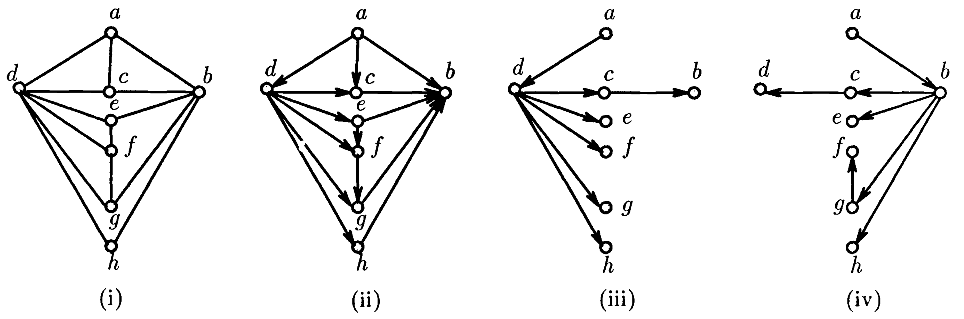


FIG. 3. (i) *A biconnected graph* G *with* $P_1 = \langle a, b, c, d, a\rangle$, $P_2 = \langle b, e, d\rangle$, $P_3 = \langle b, g, d\rangle$, P_4 $\langle b, h, d\rangle$, $P_5 = \langle d, f, g\rangle$, $P_6 = \langle e, f\rangle$, *and* $P_7\langle a, c\rangle$. (ii) $\overrightarrow{G}$. (iii) *An out-tree* $\overrightarrow{T}$ *with* a *as the root.* (iv) *An out-tree* $\overleftarrow{T}$ *with* b *as the root.*

PROPOSITION 3.1. *The ear numbers of the edges of the tree path from any vertex* v *to the root* r *in* $\overrightarrow{T}$ *decrease monotonically.*

DEFINITION 4. P_{xy} refers to an ear P with $\{x, y\}$ as the end vertices. If P is directed from x to y in $\overrightarrow{G}$, then we denote it as $P_{\overrightarrow{xy}}$.

The following useful lemma relates the attachments of a bridge and $\overrightarrow{G}$.

LEMMA 3.2. *Let* B *be a bridge of* $P_{\overrightarrow{xy}}$ *that contains no edge with an ear number less than* P. *Then if* B *has an attachment edge* (z, y) (*respectively,* (x, z)) *at* y (*respectively,* x), *then* zy (*respectively,* xz) *is directed from* z *to* y (*respectively,* x *to* z) *in* $\overrightarrow{G}$, *i.e.,* $xz \notin E(\overrightarrow{T})$.

Proof. We will prove only the case for which B has an attachment at y. A similar proof can be derived if B has an attachment at x. Figure 4 illustrates the statement of the lemma. Let $ear(z) = Q$, where the edges of Q belong to $E(B)$. Suppose that the lemma does not hold, i.e., that Q is directed such that the edge zy is directed from y to z. Trace a path $\overrightarrow{G}$ as shown in the following. Start from y, and traverse until the last vertex of Q, say, w is encountered. Suppose $ear(w) = S$ for some S that belongs to B. Since an end vertex of each ear belongs to a lower-numbered ear in an ear decomposition, we have $S < Q$. Now trace the edges of the ear S along the direction given to S until the last vertex of S is reached. This process, when continued in this fashion, uses edges of B with monotonically decreasing ear labels. Therefore, we must eventually encounter a vertex of P. From that vertex of P we can reach y by going along the direction given to P. In other words, if the lemma does not hold, we can trace a cycle starting and ending at y in $\vec{G}$, which contradicts the fact that $\vec{G}$ is acyclic. □

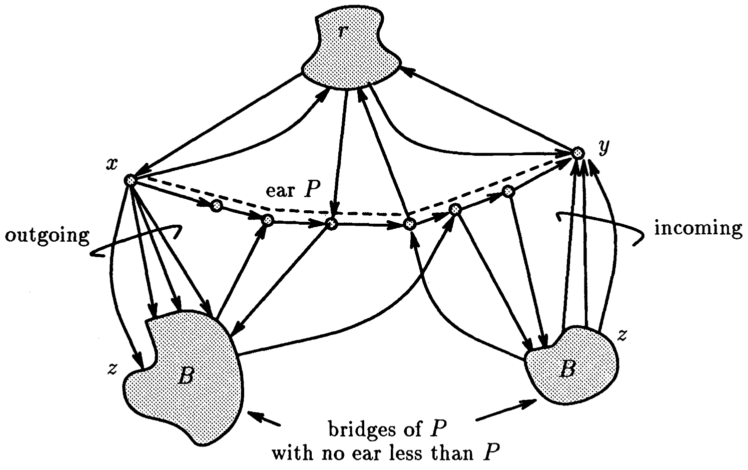


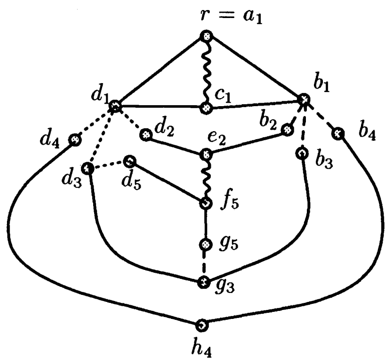
FIG. 4. *Illustration of Lemma* 3.2.

Using these directed graphs, we show how to build the local replacement graph in the following.

ALGORITHM. *Build* G'.
Input: A graph G, an open ear decomposition of G, and the directed graph $\vec{G}$ and its associated spanning trees $\vec{T}, \overleftarrow{T}$.
Output: A graph G' (local replacement graph) in which each separating pair of G is internal to some ear of G'.

1. *Construction of* $V(G')$
   $V(G') = \{v_P | v \in V(P)$ for some nontrivial ear $P\}$. Refer to v_P as a copy of v.
2. *Construction of* $E(G')$
   (a) Initialize $E(G')$ to $\{(u_P, v_P) | (u, v) \in E(P)\}$.
   (b) For every nontrivial ear $P_{\vec{vw}}$ add an edge as follows. If the least common ancestor of v and w (henceforth denoted as $lca(v, w)$) in $\vec{T}$ is v, then let Q be the ear number of the first tree edge in the path from v to w in $\vec{T}$. Then add (v_P, v_Q) to $E(G')$. If $lca(u, w) \neq v$ in $\vec{T}$, then add (v_P, v_Q) to $E(G')$, where Q is such that v is internal to Q.
   (c) Repeat Step 2(b) with $\vec{T}$ and $\vec{G}$ replaced by $\overleftarrow{T}$ and $\overleftarrow{G}$, respectively.
   (d) For all trivial ears uv, if u and v are internal to ears P and Q, respectively, then add (u_P, v_Q) to $E(G')$.

Figure 5 shows various stages of Step 2 on an example graph. The above algorithm does not quite suffice for our purposes, i.e., it need not be the case that there is a separating pair $\{x, y\}$ in G that separates an ear P if and only if $\{x_P, y_P\}$ is a separating pair in G'. This happens if the input graph G contains two or more ears with the same end vertices. Consider an example. Refer to the graph G of Fig. 3(i), and consider the subgraph $G - \{h\}$ obtained by deleting h and edges incident on h. In $G - \{h\}$ the separating pair $\{b, d\}$ separates the ear 2 but does not separate 1. In the local replacement graph corresponding to $G - \{h\}$, i.e., in $G' - \{h_4, b_4, d_4\}$ of Fig. 5, the pair $\{b_2, d_2\}$ is not a separating pair. Before we give the final step of *Build* G', we need the following definitions.



edge type	caused by
—	Step 2(a)
......	Step 2(b)
- - -	Step 2(c)
∿	Step 2(d)

FIG. 5. *Stages in the execution of Step* 2 *of Build G′ when the input graph is G of Fig.* 3.

DEFINITION 5. (i) An ear is a *parallel* ear if there is another ear with the same end vertices.

(ii) Consider the following partition on a set of parallel ears with $\{x, y\}$ as the end vertices. Denote the connected component of $G - \{x, y\}$ that contains the root by C_r. For each connected component C of $G - \{x, y\}$, $C \neq C_r$, denote the minimum of $\{ear(v) \mid v \in V(C)\}$ by P. Each such P is in a different partition. (Notice that every P has end vertices at $\{x, y\}$.) Additionally, for each P the partition that contains P contains exactly (no more than or no fewer than) those ears parallel to P whose internal vertices belong to C. Finally, the ears with $\{x, y\}$ as the end vertices that belong to C_r can be put in any (or additional) partition arbitrarily. Each partition is called a *bundle* of parallel ears.

(iii) The ear with the smallest label of a bundle is called the *representative* for that bundle.

In the example graph of Fig. 3, ears 2 and 3 are in one bundle and ear 4 is in a bundle by itself. Now we describe a method to handle parallel ears.

Algorithmically, partitioning a collection of ears into sets of parallel ears is easy because it is just grouping ears according to the end vertices. However, further classifying them into bundles is nontrivial. In the following we give an alternative definition for a bundle of parallel ears that is equivalent to Definition 5(ii).

DEFINITION 6. Define, recursively, when an ear Q *depends* on an ear P as follows. (a) P *depends* on P. (b) An ear Q depends on P if Q is not parallel to P and if for each of the end vertices v of Q there exists an ear R that depends on P such that $v \in V(R)$.

DEFINITION 7. Define, recursively, a *bundle* of parallel ears as follows. Two parallel ears P and Q are in the same bundle if one of the following holds. (a) There is a path from an internal vertex of P to an internal vertex of Q such that for every edge uv of that path, if $ear(uv) = R_{ab}$, then there exist ears that depend on either P or Q and that contain $ear(a)$ and $ear(b)$. (b) There is another parallel ear R such that P, R and R, Q are in the same bundle.

Suppose that $\{x, y\}$ separates a parallel ear P and that the component C of $G - \{x, y\}$ containing $P(x, y)$ also contains the internal vertices of Q_{xy}. Then notice that for all $v \in V(C)$, $ear(v)$ depends on either P or an ear parallel to P (such as Q) whose internal vertices belong to C. Therefore, it follows that the bundles resulting from this definition conform to the rules stated in Definition 5(ii).

In the following the first step shows how to partition parallel ears into bundles efficiently. Informally, it is as follows. First, we build an auxiliary graph G_p based on the parallel ears of G. G_p is such that P and Q belong to the same bundle if and only if the corresponding pair of vertices p and q are in the same connected component in G_p. The graph G_p is built by making use of the local replacement graph that is available after Step 2(b). Observe that after Step 2(b) each ear is hooked at only one of its ends.

Therefore, the partial G' after Step 2(b) is a tree. Denote it by $\vec{T_l}$. Now we describe the method.

ALGORITHM. *Build* G' (continued)

3. *Adjust* $E(G')$ *for parallel ears*
   (a) *Identify bundles.*
   Let G_p be an auxiliary graph whose vertices p correspond to the parallel ears P of G. The edges of G_p are added by making use of the tree $\vec{T_l}$. Consider two parallel ears $P_{\vec{xy}}$ and $Q_{\vec{xy}}$. An edge $(p,q) \in E(G_p)$ if and only if there exists an ear $R_{\vec{ab}}$ in G (possibly, a trivial ear) that satisfies the following two properties. (i) $\{a,b\} \cap \{x,y\} = \emptyset$. (ii) Let a and b are internal to the ears U and W in G, respectively. Then the tree paths in $\vec{T_l}$ from the $lca(a_U, b_W)$ to a_U and b_W start with the edges added by Step 2(b) for P and Q.
   (b) *Adjust* $E(G')$ *for parallel ears.*
   For each connected component C of G_p do the following. Find a spanning tree T and root it at the vertex with the smallest label (say, p). Let P be the ear in G that corresponds to p. For each ear $Q_{\vec{vw}}$ in G, $Q \neq P$, such that q belongs to T; (i) delete the end edges of Q added to $E(G')$ in Steps 2(b) and 2(c), and (ii) add (v_S, v_Q) and (w_S, w_Q) to $E(G')$, where s is the parent of q in T.

At the end of Step 3 of *Build* G' we still have a partition of the edge set into disjoint paths. However, because of the rearrangement of edges in Step 3(b), the end vertices of an ear may not lie on a lower-numbered ear if we continue to use the old ear labels for G'. But this deficiency is inconsequential to the rest of our algorithm, and hence we continue to use old ear labels. Notationally, if P is an ear of G, the path in G' consisting of P together with two new end edges created by local modifications will be referred to as P'. The following two propositions can be proved by induction and Definitions 6 and 7.

PROPOSITION 3.3. *If an ear* Q *depends on* $P_{\vec{xy}}$, *then both end vertices of* Q' *belong to the subtree of* $\vec{T_l}$ *rooted at* x_P.

PROPOSITION 3.4. *If an ear* Q *depends on* P, *then there is a path from any (internal or end) vertex of* Q *to an internal vertex of* P *such that if* uv *belongs to this path, then* $ear(uv)$ *depends on* P.

THEOREM 3.5. *Two parallel ears* $P_{\vec{xy}}$ *and* Q *are in the same bundle if and only if* p *and* q *are in the same connected component in* G_p.

Proof. In what follows we use the notation of Step 3. Since P and Q are parallel, Q is directed from x to y in $\vec{G}$. Furthermore, from the construction in Step 2 it follows that they have the same parent in $\vec{T_l}$; call it x_S. Denote the subtrees of $\vec{T_l}$ rooted at x_P and at x_Q by T_p and T_q, respectively. Recall that a is an end vertex of R that is internal to U. Assume without loss of generality that a_U is in T_p.

($\Rightarrow$) Assume $(p, q) \in E(G_p)$. If an ear R satisfies the condition (ii) of Step 3(a), then by Proposition 3.3 R depends neither on P nor on Q. Conversely, if R is the ear with smallest label that depends on neither P nor Q, then by Definition 6 R cannot have both its end vertices in one of T_p, T_q. This follows from the fact that if an ear has both its endpoints in T_p (respectively, T_q), then it depends on P (respectively, Q). Hence R satisfies the conditions of Step 3(a). Assume without loss of generality that R is the ear with the smallest label that satisfies the conditions of Step 3(a). Because U and W contain the end vertices of R, $U < R$ and $V < R$. Since R is the ear with smallest label that satisfies the conditions of Step 3(a), U depends on P and W depends on Q. Then by condition (i) of Step 3(a) and Proposition 3.4 there is a path from an internal vertex of P to an internal vertex of Q that satisfies the condition (a) of Definition 7.

($\Rightarrow$) Assume that there are no ears parallel to P except Q in the bundle containing P. Because the condition (b) of Definition 7 is transitive, it suffices to prove just this case. Suppose that $P < Q$. We are required to show that there exists an ear R that satisfies the conditions of Step 3. Consider the subgraph D of G formed by P, Q, and all those ears of G that depend on either P or Q. In this subgraph P and Q are clearly two different bridges of $\{x, y\}$. Now an inductive proof in which the induction step consists of adding an ear that depends on either P or Q shows that in the new subgraph D, P and Q are in different bridges of $\{x, y\}$. That is, there is no path from an internal vertex of P to that of Q in D. Denote the bridge of $\{x, y\}$ in D containing P and Q by D_p and D_q, respectively. If P and Q are in the same bundle, then by Definition 6 and the fact that there are no other parallel ears in D there must exist an ear R that connects a nonattachment vertex of D_p, say, a, to a nonattachment vertex of D_q, say, b. Because a and b are nonattachment vertices of D_p and D_q, neither of them is from $\{x, y\}$. Therefore, R satisfies condition (i) of Step 3(a). Since a and b are internal to ears that depend on P and Q, respectively, the end vertices of R' belong to T_p and T_q by Proposition 3.3. That is, R satisfies condition (ii) of Step 3(a). □

The reverse direction of the above theorem can be stated equivalently as follows.

COROLLARY 3.6. *If $(p, q) \in G_p$, then there is a path from an internal vertex of P to an internal vertex of Q such that this path does not use any vertices of P or of ears parallel to P.*

Next we prove the correctness of *Build G'*. The following simple facts are useful in the proofs. Observe that the number of copies of a vertex v in G' is one more than the number of nontrivial ears for which v is an end vertex, i.e., it is $1 + (\text{degree}(v) - 2)$. The algorithm *Build G'* adds an edge only between two copies of the same vertex, and it adds one edge for every end vertex of every nontrivial ear (see Fig. 6). Therefore, we have the following proposition.

PROPOSITION 3.7. (i) *The edges that connect different copies of a single vertex v form a tree in G'. Therefore, there is a path between any two copies of v in G' that uses only other copies v. As a consequence, for a vertex $v \notin V(P)$ all copies of v belong to a single bridge of P' in G'.* (ii) *The graph G can be obtained from G' by collapsing, for each v, all copies of v into one.*

DEFINITION 8. For an ear P we say a bridge of P is a *relevant* bridge if for each ear Q that belongs to it (a) $Q > P$ and (b) if Q is parallel to P, then Q is in the same bundle as P. If a bridge is not relevant, then it is said to be *irrelevant.*

Observe that for a set of parallel ears with x and y as the end vertices, if one of the ears in $\vec{G}$ is directed from x to y, then all of them are directed from x and y. After Step 2

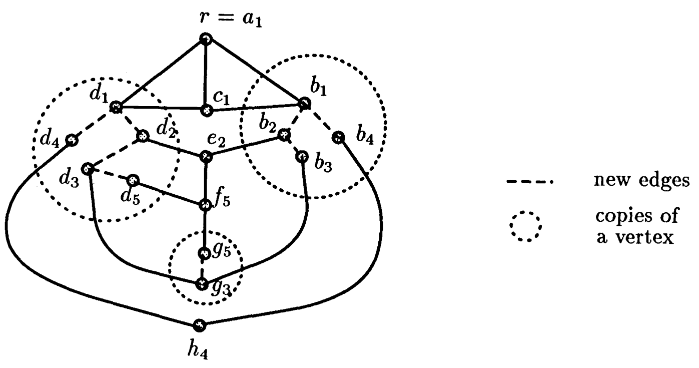


FIG. 6. *Final G' after Step* 3.

(but before Step 3), if P and Q are parallel in G, then P' and Q' are parallel in G' by the construction of Step 2. Because Step 3 does not change the end edges created in Step 2 for the representatives, we have the following proposition.

PROPOSITION 3.8. *If P and Q are parallel ears such that each is a representative of its bundle, then P' and Q' have the same end vertices in G'.*

For a subgraph D of G let $\mathcal{E}(D) = \{ear(uv) | uv \in E(D)\}$. The following can be derived from Definition 8 and the definition of open ear decomposition.

PROPOSITION 3.9. *Suppose that B is a relevant bridge of P. If B contains an edge of an ear Q, then B contains all edges of Q, i.e., if $Q \in \mathcal{E}(B)$, then all edges of Q belong to B.*

Another useful fact can be derived from the definition of open ear decomposition by using the minimum of $\mathcal{E}(B)$.

PROPOSITION 3.10. *If B is a bridge of an ear P that contains an edge whose ear label is less than P, then B has attachments at the end vertices of P.*

LEMMA 3.11. *Consider a relevant bridge B of $P_{\overrightarrow{xy}}$ (respectively, $P_{\overleftarrow{xy}}$) in G. Suppose it contains an ear Q with one end vertex at x (respectively, y) and the other at $v \notin V(P)$. Then the $lca(x, v)$ (respectively, $lca(y, v)$) in $\overrightarrow{T}$ (respectively, $\overleftarrow{T}$) is x (respectively, y).*

Proof. We will prove the lemma when Q has an attachment at x. A similar proof works when Q has an attachment at y. We will show that x is an ancestor of v in $\overrightarrow{T}$. Observe that all paths from v to r in G must go through a vertex of P because v belongs to a bridge of P and this bridge does not contain r. Specifically, the tree path from v to r must have a vertex, say, w, of P. This vertex w of P cannot be y because, by Lemma 3.2, y is not reachable in $\overrightarrow{T}$ from v. Hence the tree path from v to r must encounter a vertex w from $P[x, y)$. Now the last vertex of P on this tree path is x because of the direction given to P. Therefore, the tree path from v to r in $\overrightarrow{T}$ contains x. In other words, x is an ancestor of v in $\overrightarrow{T}$. □

LEMMA 3.12. *Consider a relevant bridge B of $P_{\overrightarrow{xy}}$ in G. Suppose it contains an ear Q with one end vertex at $u \in V(P)$ and the other at $v \notin V(P)$. If $lca(u, v) = u$, then the edges on the tree path from v to u in $\overrightarrow{T}$ belong either to P or to those ears of B whose label is less than Q.*

Proof. Assume $ear(v)$ is U for some $U \in \mathcal{E}(B)$. By the definition of a relevant bridge, $U > P$. We traverse the tree path from v to u. As argued in the proof for Lemma 3.11, we must encounter a vertex z of P from $P[x, y)$ in this traversal. Now u is an ancestor of v because $lca(u, v) = u$. Therefore, z is a descendant of u. Finish the traversal of the tree path from z to u by taking the edges of P from z to u. In this traversal U initially belongs to a relevant bridge, namely, B. Also, whenever the ear labels change from, say, P_1 to P_2 for $P_1, P_2 \neq P$, the switch occurs at the end vertex of P_1. This end vertex is a nonattachment vertex of B because otherwise we would have moved from P_1 to P instead of to P_2. Hence P_1 and P_2 are in the same bridge of P. Since U belongs to P, P_1 and hence P_2 belong to B. In summary, we showed that the traversal of the tree path from v to u uses the edges of the ears of B and possibly some of P. □

THEOREM 3.13. *Let B_q be the bridge of $P_{\overrightarrow{xy}}$ in G containing the ear Q. Let BL_q be the bridge of P' in G' containing the ear Q'. Then* (i) *for every ear S if $S \in \mathcal{E}(B_q)$, then $S' \in \mathcal{E}(BL_q)$. Additionally,* (ii) *if B_q is relevant bridge, then for every S' if $S' \in \mathcal{E}(BL_q)$, then $S \in \mathcal{E}(B_q)$.*

Proof. If two ears Q and S are in the same bridge B of P in G, then there must be a path between a vertex of Q and a vertex of S consisting of only the nonattachment vertices of B. Then by Proposition 3.7 it follows that there is a single bridge of P' in G' containing Q' and S'. That proves part (i).

Next we will prove by induction that if two ears belong to two different bridges B_1 and B_2 in G, then they belong to two different bridges of P' in G' provided that at least one of B_1, B_2 is a relevant bridge. For the base of the induction we start with the subgraph D_1 of G' formed by P' plus $\{Q' \mid$ where $Q < P$, or Q is the minimum labeled ear of a bridge of P, or Q is parallel to P and Q is the representative of its bundle$\}$. The ith induction step consists of building D_i from D_{i-1} by adding the smallest ear from $\mathcal{E}(G) - \mathcal{E}(D_{i-1})$. This ear is added in a manner that conforms with the construction rules of Steps 2 and 3, and thus we maintain the invariant that D_i is a subgraph of G' at all times.

To show that D_1 satisfies the base case, we claim that each ear of a relevant bridge of P in G is in a bridge (of P') by itself in D_1. First, observe these ears have their end vertices on P in G. In *Build G'* the end edges for the minimum labeled ears of a relevant bridge of P are decided in Step 2, and these are not changed later in Step 3. To prove the claim, it suffices to show that each of these ears is attached in Step 2 to some vertices of P'. All parallel ears are attached to the end vertices of P' by Proposition 3.8. For any other ear $Q_{\overrightarrow{vw}}$, $v, w \in V(P)$, if $w = y$, then $v \neq x$ because we assumed that Q is not parallel to P. In that case $lca(v, w) \neq v$ in $\overrightarrow{T}$ (because the end edge of P incident on y is not present in $\overrightarrow{T}$) and v_Q is attached to v_P. The other end vertex w is attached to w_P because $lca(w, v) = w = y$ in $\overleftarrow{T}$ and the tree path from w to v starts with an edge of P. The cases arising from the other positions for v and w on P can be analyzed similarly, and we conclude that v_Q and w_Q are attached to v_P and w_P, respectively.

Now we prove the induction step. Assume, inductively, that the theorem holds when the smallest $(i-1)$ ears from $\mathcal{E}(G) - \mathcal{E}(D_1)$ are added. Consider the ith smallest ear $Q_{\overrightarrow{uv}}$.

Suppose there is an ear (possibly P) parallel to $Q_{\overrightarrow{uv}}$. Then from the construction of Step 3(b) we notice that Q is attached to the copies of u and v, say, u_S and v_S, where s is (in the notation of Step 3(b)) the parent of q in T. The case in which $S = P$ results in creating a new bridge when Q' is added to D_{i-1}, and the induction step is trivially true. But if $s \neq p$ and $(s, q) \in E(G_p)$, then by Corollary 3.6 there is a path between an internal vertex of Q to an internal vertex of S that does not use any vertices of ears parallel to Q. Specifically, this path does not use any vertices of P. Hence S and Q are in the same bridge of P in G.

Next, assume that $Q_{\overrightarrow{uv}}$ has no parallel ears. Let $ear(u)$ and $ear(v)$ be S and U, respectively. We will analyze the cases resulting from assuming different positions for u and v.

Assume that neither u nor v belongs to P. Then by Proposition 3.7 Q' is connected to a nonattachment vertex of the bridge of P' (in D_{i-1}) containing S' (call it BL_s) to a nonattachment vertex of the bridge containing U' (call it BL_u). Clearly, S and U (and hence the ears of G corresponding to the ears of BL_s and BL_u) are in one bridge of P in G because of the path (namely, Q) between them that uses no vertices of P.

Now consider the case in which both u and v belong to P, i.e., $P = S = U$. Then it cannot be that $u = x$ and $v = y$ since that would make Q have an ear (namely, P) parallel to it. Hence one of the end vertices of Q must be internal to P. In this case, by a proof similar to the one used for the base case, Q' is attached to two of the vertices of P'. In other words, if both u and v belong to $V(P)$, then a new bridge is created and the induction step holds.

Next, assume that one of u, v belongs to P and the other does not. Assume without loss of generality that u belongs to P. Clearly, U and Q are in a single bridge of P in G because they are connected at v. Therefore, if Q' attaches itself to the bridge of P' in D_{i-1} that contains U' (denote this bridge by BL_u), then the induction step holds because U (and all the ears corresponding to the ears of BL_u) and Q are in a single bridge of P in G. We will show that this is indeed the case, i.e., that Q' attaches itself to BL_u. Notice that by Proposition 3.7 since $v \notin V(P)$, there exists a bridge of P' in D_{i-1} that contains all copies of v. Therefore, v_Q is connected to a copy of v that belongs to BL_u. It remains to be shown that v_Q is not connected to a nonattachment vertex of a bridge other than BL_u. Consider the $lca(u, v)$ in $\overrightarrow{T}$. If $lca(u, v) \neq u$, then u_Q is connected to u_P by Step 2. Otherwise, by Lemma 3.12 the tree path in $\overrightarrow{T}$ from v to u consists of edges of either P or the ears from $\mathcal{E}(BL_u)$. Therefore, u_Q is connected to u_P or u_W, where $W \in \mathcal{E}(BL_u)$. □

3.2. Reduction to biconnected components. We briefly alluded to reducing triconnectivity to biconnectivity at the beginning of §3. We elaborate more on this reduction here. In this subsection we show how to find a collection similar to $\mathcal{H}$ that encodes separating pairs. There are two reasons for finding a collection that is only similar to $\mathcal{H}$ and that is not $\mathcal{H}$ itself as defined before. The first reason is that the new collection suffices for our purposes, and the second reason is that it can be computed by an easy reduction to any biconnected component algorithm.

Recall the definition of $\mathcal{H}$ from Definition 2; it consists of a graph H_P for each nontrivial ear P. We slightly change the definition of H_P because it is difficult to efficiently check for $v \in V(P)$ if the bridge of P containing the root r is adjacent to v.

DEFINITION 9. The graph H_P is defined as in Definition 2 with the following modification to part (i) of that definition. We say that (v, r_P) is added to $E(H_P)$ if v is adjacent to w and w belongs to an irrelevant bridge of P.

Even though the resulting collection is slightly different, we will continue to denote it by $\mathcal{H}$. Let us see whether this new definition of graph H_P also encodes the set of separating pairs of G. If the bridge containing w also contains the root r, then the new rule results in the same H_P. Otherwise, the bridge containing w must necessarily have attachments at x and y by Proposition 3.10. Therefore, we would have added (v, x) and (v, y) to $E(H_P)$ instead of (v, r_P). A pair of vertices of P that is not a separating pair before would not be a separating pair now. The converse also holds except for the pair $\{x, y\}$. In this case notice that we would detect $\{x, y\}$ as a separating pair on H_Q, where

$\{x, y\}$ separates Q. This is true because by Definition 1 all vertices of $Q(x, y)$ are adjacent to vertices of bridges of Q relevant to Q. Therefore, the edges added to $E(H_Q)$ for the vertices of $Q(x, y)$ are identical irrespective of whether the old or new rule is used.

Next we show how to build $\mathcal{H}$ by a reduction to biconnected components. The biconnected component algorithm is run on a multigraph G_e (the subscript e to indicate that G_e is built from the ears of G') constructed from G' and its ears.

DEFINITION 10. Let P' be a nontrivial ear $\langle v_0, v_1, \ldots, v_k\rangle$ of length greater than two, i.e., $k \geq 3$. The graph G_e is obtained from G' by contracting all such ears P' by merging the internal vertices $v_1, v_2, \ldots, v_{k-1}$ into a single vertex p.

Figure 7 shows the multigraph G_e for the local replacement graph and its ear decomposition shown in Fig. 6. Recall that the representative (say, P) of a bundle of parallel ears is the ear of that bundle with the smallest label. Observe that the end vertices P belong to a lower-numbered ear that is not parallel to P. This observation together with the definition of open ear decomposition imply the following simple fact.

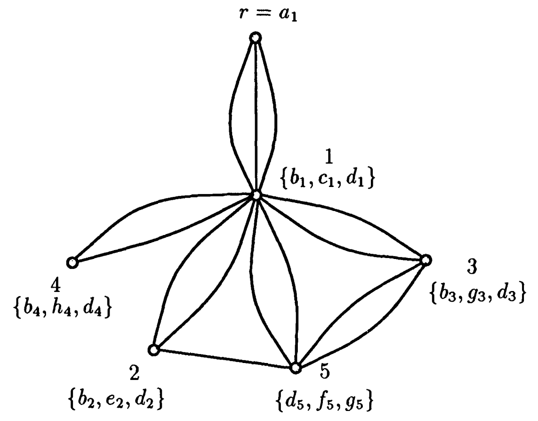


FIG. 7. *The multigraph G_e based on G' of Fig. 6.*

PROPOSITION 3.14. *If D is a biconnected component of G_e, then the vertex of D with the minimum label is either r or an articulation vertex of G_e.*

There is a correspondence between the relevant bridges of P_{xy} in G, the bridges of $P'[x_P, y_P]$ in G', and the connected components resulting from deleting p from G_e. The relation between the latter two is simple: from the definition of G_e it follows that the ears Q and R are in the same bridge of $P'[x_P, y_P]$ in G' if and only if q and r are in the same component in $G_e - \{p\}$. The relation between the attachments of the bridges of P in G and those of the bridges of $P'[x_P, y_P]$ in G' is stated below.

THEOREM 3.15. *Let B_q denote the bridge of $P_{\vec{xy}}$ in G that contains Q. Similarly, denote the bridge of P' in G' that contains Q' by BL_q.*

(i) *B_q is relevant to P if and only if BL_q has no attachments at the end vertices of P' in G'.*

(ii) *Assume that B_q is relevant to P. Then B_q has an attachment at v if and only if BL_q has an attachment at v_P.*

Proof. (i) Assume that B_q is a relevant bridge for P. Then by an inductive proof similar to the one used to show part (ii) of Theorem 3.13 the attachments of BL_q are from $P'[x_P, y_P]$. The invariant that should be maintained at all times is that if Q belongs to a relevant bridge, then the bridge of P' containing Q' in D_i has all its attachments on $P'[x_P, y_P]$.

Assume that B_q is not a relevant bridge. We will show that BL_q has attachments at the end vertices of P' in G'. There are two kinds of irrelevant bridges: the ones that have an ear whose number is less than P and the ones that have an ear parallel to P from a different bundle in G.

Case 1. Assume that B_q has an ear whose number is less than P. Then BL_q contains an ear whose number is less than P' by Theorem 3.13(i).

Case 1a. Assume, in addition, that BL_q contains an ear parallel to P from the same bundle as P. Then from the construction of Step 3(b) it follows that BL_q has attachments at the end vertices of P'.

Case 1b. Assume that ears of BL_q parallel to P are not from the same bundle as P. Consider the subgraph of G' consisting of P', the ears of BL_q that have no other ears parallel to them, plus the representative from each bundle of parallel ears that belongs to BL_q. Clearly, this subgraph is biconnected and the labels on the ears define an open ear decomposition. Therefore, by Proposition 3.10 BL_q has attachments at the end vertices of P'.

Case 2. Assume that B_q is an irrelevant bridge because it contains an ear S parallel to P such that P and S are in different bundles. Assume without loss of generality that S is the representative of its bundle. By Proposition 3.8 the end vertices of S' and the end vertices of the representative of the bundle of P' are the same. Therefore, BL_q has attachments at the end vertices of P'.

(ii) Assume that BL_q has an attachment at v_P and that this attachment belongs to an ear S' of BL_q. Since B_q is a relevant bridge, by part (ii) of Theorem 3.13, $S \in \mathcal{E}(B_q)$. By Proposition 3.7, $v \in V(S)$. Therefore, B_q has an attachment at v.

Consider the "only if" direction of the theorem. Let $S_{\overrightarrow{vw}}$ denote the ear from the bridge B_q with the least label that has an attachment at v. Then if $lca(v,w) \neq v$ in $\vec{T}$, then by Lemma 3.11 v is an internal vertex of P. Therefore, by Step 2(c), v_S is attached to v_P. Otherwise, i.e., if $lca(v,w) = v$, then by Lemma 3.12 the tree path from u to v consists of the edges from either P or the ears of B_q with a label less than S. Because there are no ears of B_q less than S incident on v, the tree path must end with the edges of P. Hence, by Step 2 v_S is attached to the vertex v_P. □

Part (ii) of Theorem 3.15 implies the following.

COROLLARY 3.16. *$\{x,y\}$ is a separating pair that separates P in G if and only if $\{x_P, y_P\}$ separates P' in G'.*

Proof. Consider the bridges BL_q of $P'[x_P, y_P]$ in G'. If $\{x_P, y_P\}$ separates P' in G', then the corresponding bridges B_q in G are relevant to P by the reverse direction of (i). Therefore, the reverse direction of the corollary follows from part (ii) of Theorem 3.15 and from Theorem 2.3. The argument for the forward direction is similar. □

We exploit this correspondence in building $\mathcal{H}$. The nontrivial part in building H_P is in identifying which edges, if any, to add for a vertex v of $P[x,y]$. This involves knowing the extreme attachments of the relevant bridges of P adjacent to v. If there are any bridges of P that are relevant, then p would be an articulation vertex in G_e; each block D of G_e attached to p that does not contain r' corresponds to a relevant bridge of P. Therefore, we give a common label, called α label, to all vertices (except p) of each such block D of G_e. This α label is a 3-tuple $\langle P, a, b\rangle$: a and b are the *pos* labels of the extreme attachments of the bridge of P that correspond to D, and the first tuple reflects the fact that these attachments are on P. We will denote the first, second, and the third tuples of $\alpha(q)$ by $\alpha(q).1$, $\alpha(q).2$, and $\alpha(q).3$, respectively.

It turns out that the α labeling can be computed efficiently by a slight modification of any biconnected component algorithm as shown below. Given α labeling, the edges that need to be added to build H_P can be figured out quite easily.

ALGORITHM. *Build $\mathcal{H}$.*
Input: A graph G, an open ear decomposition of G, *pos* labeling for each ear, and the local replacement graph G' for that decomposition.
Output: A collection of graphs $\mathcal{H}$ (built on the top of each nontrivial ear of G) that encode separating pairs succinctly.

1. Build a multigraph G_e from G' by merging the internal vertices of each nontrivial ear P' into a single vertex p. Discard the self loops from G_e.
2. Find the biconnected components $D_1, D_2, \ldots, D_k$ of G_e.
3. Label the vertices of G_e with 3-tuples:
   For each D_i do the following. Let p be the vertex of D_i with the smallest label. For each $q \in V(D_i) - \{p\}$ set the first component of $\alpha(q)$ to P. Of all edges ps of D_i incident on p, consider those that have an image, say, cd, in G. Assume $c \in V(P)$ (see Fig. 8). Let a and b the minimum and the maximum, respectively, of the *pos* labels of all such c. Then a and b are the second and third components of $\alpha(q)$.

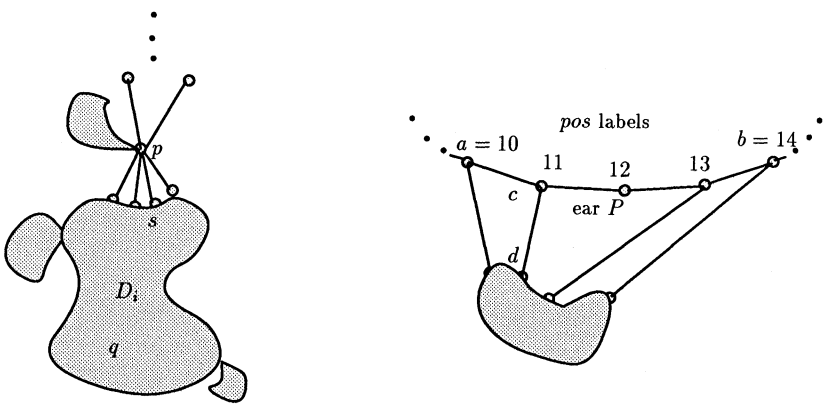


FIG. 8. *Illustration of Step* 3 *of Build $\mathcal{H}$.*

4. Build H_P for each nontrivial ear P by using the α labels:
   (a) Assign $V(H_P) = V(P) \cup \{r_P\}$.
   (b) Initialize $E(H_P) = E(P)$. For every vertex $v \in V(P)$ add the following edges. For an edge $vw \in E(G)$, $w \notin V(P)$, denote $ear(w)$ by Q.
      (i) If $\alpha(q).1$ is less than P, then add (v, r_P) to $E(H_P)$. Otherwise, add two edges as shown in the next step.
      (ii) Let a be such that $pos(a) = \min\{\{\alpha(q).2 | ear(w) = q \text{ and } vw \in E(G), w \notin V(P)\} \cup \{pos(w) | vw \in E(G) \text{ and } v, w \in V(P)\}\}$. Add (v, a) to $E(H_P)$. Also, add (v, b), where b is obtained similarly by using max and $\alpha(q).3$.

LEMMA 3.17. *Consider an ear P and an edge $vw \in E(G)$, $w \notin V(P)$, where v is an internal vertex of P. Let $ear(w)$ be Q. Then if the bridge B_q of P containing Q contains no ear number less than P, then $\alpha(q).1 = P$.*

Proof. If B_q contained ears parallel to P, then they would have to be in the same bundle with P because of the edge vw. Therefore, B_q is a relevant bridge of P. Now it follows from part (i) of Theorem 3.15 that the bridge containing Q' would have all its attachments on the internal vertices of P'. Then p would be an articulation vertex that is on every path between q and the root vertex r in G_e. Therefore, $\alpha(q).1 \geq P$. But p and q are in the same block because the edge vw of G causes an edge between p and q in G_e. Hence, $\alpha(q).1 = P$. □

THEOREM 3.18. *Algorithm Build $\mathcal{H}$ computes $\mathcal{H}$ as defined in Definition* 9.

Proof. Consider an ear $P_{\overrightarrow{xy}}$ and an internal vertex v of P. We need to argue that the construction carried out in Step 4 is correct. Suppose there is an edge $vw \in E(G)$, $w \notin V(P)$. Denote $ear(w)$ by Q. Denote the bridge of P containing Q by B_q and the $\min\{\mathcal{E}(B_q)\}$ by N. Note that regardless of whether vw is a trivial ear or one of the end edges of Q, it causes an edge to be added between p and q in G_e. Therefore, there is a block that contains both p and q.

According to Definition 9 the edge (v, r_P) is added to $E(H_P)$ if and only if N is less than P. Consider the construction in Step 4. We claim that $N < P$ if and only if $\alpha(q).1$ is less than P. To prove the forward direction of the claim, we need to prove that the block D of G_e containing p and q contains an articulation vertex m such that $M < P$. By Proposition 3.14 it is sufficient to show that p is not the articulation vertex in D with the smallest label. Because $N < P$, the bridge B_q is irrelevant to P. Now if p is an articulation vertex that appears on every path from q to the root r', then BL_q of P' containing Q' in G' has all its attachments on $P[x_P, y_P]$. This contradicts part (i) of Theorem 3.15. Hence p cannot be the articulation vertex with the smallest label in D. Consider the reverse direction. Assume for contradiction that $\alpha(q).1$ less than P but that $N \geq P$. Clearly, $N \neq P$ because N belongs to a bridge of P. But if $N > P$, then, from Lemma 3.17, $\alpha(q).1$ is exactly P and is not less than P.

Next consider the case in which there is an edge $vw \in E(G)$, $w \notin V(P)$, but for all such edges $\alpha(q).1$ is not less than P. By the claim of the previous paragraph if $\alpha(q).1$ is not less than P, then $N > P$. Therefore, by Lemma 3.17, $\alpha(q).1 = P$. That is, every bridge of P' with an attachment at v has all its attachments on $P'[x_P, y_P]$. Because of the edge vw, each such bridge B_q is a relevant bridge of P. Therefore, by part (ii) of Theorem 3.15 the attachments of BL_q on P' are identical to the attachments of B_q on P. From the definition of G_e the attachments of B_q become the edges of G_e incident on p where these edges belong to the block containing q. The attachments of B_q that are closest to x and y are reflected in the second and third tuples of $\alpha(q)$, as computed in Step 3. Therefore, Step 4(b)(ii) computes the $E(H_P)$ as defined in part (ii) of Definition 2. □

3.3. Extraction and output representation of separating pairs. Observe that in $O(m+n)$ time we cannot possibly list all separating pairs of a graph because there could be $O(n^2)$ of them. For example, in a simple cycle every pair of nonadjacent vertices is a separating pair. That is, a cycle of n vertices has $n(n-3)/2$ separating pairs. Our output representation is a set of paths referred to as candidate lists such that a pair of vertices $\{u, v\}$ separates P if and only if u and v are nonadjacent on P and there is a candidate list generated from H_P that contains u and v. Such a representation of separating pairs is sufficient to divide a graph into triconnected components.

Let us reexamine the graph H_P of $\mathcal{H}$ from §3.2. Assume that H_P happens to be planar and that it is drawn in the plane as shown in Fig. 9, i.e., P appears as a horizontal line and all its bridges in H_P are drawn on the top. (This embedding will henceforth be referred to as a *canonical* embedding.) Then an important, but easy to see, observation that follows from Theorem 2.3 is that a pair of vertices x and y separates P if and only if a bounded region in the canonical embedding of H_P contains x and y. This fact can be used quite effectively in producing candidate lists as demonstrated below.

ALGORITHM. *Find Candidate Lists.*
Input: A collection of planar graphs encoding separating pairs.
Output: A set of paths, called candidate lists, encoding separating pairs. The output representation is a linked list.

```
1. For every bridge $B$ of $P$ in $H_P$ that does not contain $r_P$ let $u$ and $v$, respectively,
   be the attachments of $B$ with the minimum and maximum $pos$ value and do the
   following:
   (i) Output the edge $(u, v)$.
   (ii) If $v$ is not adjacent to $r_P$ in $H_P$ and if the furthest vertex to the left that is
        reachable from $v$ through a bridge is $u$, then perform this step. Let $w$ be
        the furthest vertex to the right that is reachable through a bridge adjacent
        to $v$. Add a pointer from the output location of $(u, v)$ to that of $(v, x)$,
        where $x = w$ if $w \neq v$; otherwise, $x$ is the successor of $v$ on $P$ (see Fig. 9).
2. For every edge $(a, b)$ of $P, pos(a) < pos(b)$, that is not part of a triangular region
   do the following:
   (i) Output the edge $(a, b)$.
   (ii) If the furthest vertex to the left that is reachable from $b$ through a bridge
        is $b$, then perform this step. Let $c$ be the furthest vertex to the right that is
        reachable through a bridge adjacent to $b$. Add a pointer from the output
        location of $(a, b)$ to that of $(b, d)$, where $d = c$ if $c \neq b$; otherwise, $d$ is the
        successor of $b$ on $P$ (see Fig. 9).
```

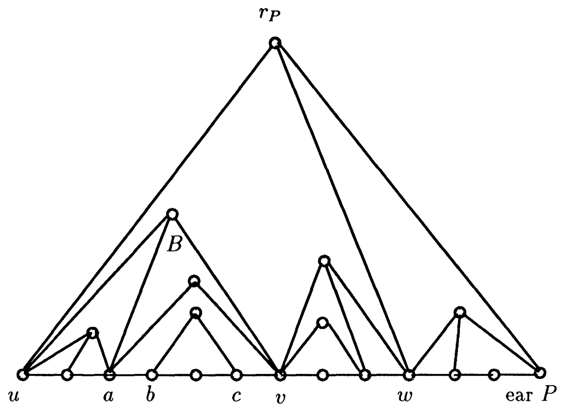


FIG. 9. *Generating candidate sets from a planar I_P.*

In the rest of this subsection we show how to take a nonplanar H_P and produce a planar graph I_P on the same set of vertices such that the set of separating pairs of H_P and that of I_P are identical. We will denote the resulting planar collection by $\mathcal{I}$. The process of planarizing H_P involves coalescing interlacing bridges. Call a pair of bridges B_1 and B_2 of P interlacing bridges if two of the attachments of B_1 and B_2 are at x, y and u, v, respectively, such that $pos(x) < pos(u) < pos(y) < pos(v)$. The operation of coalescing is to discard the bridges B_1 and B_2 and to put in a new, single bridge whose attachments are a union of the attachments of B_1 and B_2. A nice property of the operation of coalescing that follows from Theorem 2.3 is that it preserves the set of separating pairs. Now if bridges of H_P are coalesced until no more bridges are interlacing, then the resulting graph is planar; this fact can easily be proved by constructing a canonical embedding of the resulting graph. Denote the resulting planar graph by I_P.

In the following we give a fast parallel algorithm for finding $\mathcal{I}$. At a high level the algorithm reduces the problem of planarizing H_P to that of finding connected components of a graph G_b. The vertices of G_b correspond to bridges of P. Two vertices b_1 and b_2 of G_b are in the same connected components if and only if the bridges corresponding to the two vertices B_1 and B_2 are interlacing. In constructing G_b if we add edges between b_1 and all vertices whose corresponding bridges in H_P interlace with B_1, then there could be far too many edges in G_b (as many as $O(n^2)$). The trick is to add at most two edges per vertex. Add (b_1, b_2) (respectively, (b_1, b_3)) where B_2 (respectively, B_3) interlaces B_1 and furthermore, is the bridge with an attachment that is furthest to the left (respectively, right) with respect to the leftmost (respectively, rightmost) attachment of B_1. The following argument shows that this trick does not alter the connected components of G_b. Consider a proof by contradiction. Let $B_1 = (s, t)$ be the rightmost bridge (i.e., highest $pos(t)$ value) with an interlacing bridge $B_2 = (u, v), pos(u) < pos(s) < pos(v) < pos(t)$, such that $(b_1, b_2) \notin E(G_b)$. Then by our construction there must exist (b_2, r_1) and (b_1, l_1) in G_b where the bridges $R = (a, b)$ and $L = (c, d)$ interlace with B_2 and B_1, respectively. Furthermore, R and L are such that $pos(d) < pos(u)$ and $pos(b) > pos(t)$. Now since R interlaces with B_2, $a \in P(u, v)$. If $a \in P(s, v)$, then R and B_1 interlace. Because we assumed that B_1 is the rightmost bridge that belongs to a "bad" interlacement pair, we can conclude that $(r, b_1) \in E(G_b)$. On the other hand, if $a \in P(u, s)$, R and L interlace and $(r, l) \in E(G_b)$. In either case b_1 and b_2 belong to the same connected component in G_b, a contradiction.

We summarize the ideas below.

ALGORITHM. *Planarize $\mathcal{H}$.*
Input: The collection $\mathcal{H}$ of graphs encoding separating pairs of G.
Output: A collection $\mathcal{I}$ of planar graphs encoding separating pairs of G succinctly.

```
1. For each $H_P$ build a graph $G_b$ based on the bridges of $P_{xy}$ in $H_P$ that do not
   contain $r_P$. These bridges are all single edges $(u, v)$.
   (i) For each such $(u, v)$ create a vertex in $G_b$ denoted by the 2-tuple $\langle u, v\rangle$.
   (ii) Find the bridge $(a, b)$ (respectively, $(c, d)$) with one attachment on $P(u, v)$
        and the other furthest to the left (respectively, right) from $u$ (respectively,
        $v$) on $P[x, u)$ (respectively, $P(v, y]$).
   (iii) Add an edge between $\langle u, v\rangle$ and $\langle a, b\rangle$, and between $\langle u, v\rangle$ and $\langle c, d\rangle$ in $G_b$.
2. Find the connected components $C_1, C_2, \ldots, C_k$ of $G_b$.
3. Build $I_P$ by using the connected components of $G_b$.
   (i) $V(I_P) = V(H_P) \cup \{c_i | C_i \text{ is a component of } G_b\}$.
   (ii) $E(I_P) = E(P)$. Add the edges incident on $r_P$ to $E(I_P)$. In addition,
        include the following edges. Add $(c_i, u)$ if there a vertex in $C_i$ with $u$ as
        one of the 2-tuples in its label.
```

3.4. Complexity on a CRCW PRAM. The results of the previous subsections show that the following algorithm generates all separating pairs (as candidate lists) of a given biconnected graph G.

ALGORITHM. *Separating Pairs.*

```
1. Find an open ear decomposition of $G$.
2. Construct the local replacement graph $G'$ by executing *Build $G'$*.
```

3. Succinctly encode separating pairs by using *Build* $\mathcal{H}$.
4. Use *Planarize* $\mathcal{H}$ and obtain the collection $\mathcal{I}$.
5. Generate separating pairs as candidate lists by invoking *Find Candidate Lists* with $\mathcal{I}$ as its input.

The most expensive step of the above algorithm from the deterministic complexity point of view is providing the required input representations to some of the subroutines we use and building the adjacency lists of the auxiliary graphs. First, we show how to construct the needed representation of graphs by assuming that the input is a list of edges. This construction runs in $O(\log n)$ time with $((m+n)\log\log n)$ work. The needed representation is a special kind of adjacency list. In this representation every edge (u, v) appears as two directed edges $\langle u, v\rangle$ and $\langle v, u\rangle$, i.e., the vertices u and v appear in each other's adjacency lists. Given a list of edges, we can accomplish this by (1) sorting the edges to make sure that no edge appears twice initially, (2) creating $\langle v, u\rangle$ for every $\langle u, v\rangle$, (3) evaluating the degree of each vertex v by using the difference in the addresses in the sorted array of the first occurrence and last occurrence of v, (4) allocating in memory one array of size degree(v) for each v, and (5) making the processor allocated to (u, v) responsible for creating the entry in the list of u. We use the parallel bucket-sort algorithm of Hagerup [10], which runs in $O(\log n)$ time with $(n\log\log n)$ operations to sort n numbers to achieve the desired complexity.

In the remainder of this subsection we argue that the complexity of the rest of the algorithm is identical to that of the best-known parallel connected component algorithm.

Let G have n vertices and m edges. We say an algorithm has an "almost-optimal" processor–time bound if it runs in $O(\log n)$ parallel time with $O((m+n)\alpha(m,n)/\log n)$ processors on a CRCW PRAM, where α is the inverse Ackermann function. We first note the following results on optimal and almost-optimal parallel algorithms.

(A) List ranking on n elements can be performed optimally in $O(\log n)$ time on an Exclusive Read Exclusive Write (EREW) PRAM [3].

(B) Connected components and the spanning tree of an n-node, m-edge graph can be found in $O(\log n)$ time with $O((m+n)\alpha(m,n)/\log n)$ processors on an arbitrary-CRCW PRAM [3] provided that the input is presented as an adjacency list.

(C) Least common ancestors of k pairs of vertices in an n-node tree can be found in $O(1)$ time with k processors after $O(\log n)$ time preprocessing with $O(n/\log n)$ processors on an EREW PRAM by using the algorithm in [23].

(D) The Euler-tour technique on trees of [24] can be implemented optimally in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM by using (A).

(E) Using the above-mentioned results as subroutines, we obtain an almost-optimal parallel algorithm for finding an open ear decomposition from the algorithm in [15], [17], for finding biconnected components from the algorithm of [24], and for finding an st-numbering in a biconnected graph from the algorithm of [15].

We will refer to the above five results while describing the processor-time complexity of the *Separating Pairs* algorithm.

Step 1 can be performed by using (E).

Consider Step 2. The digraphs $\overrightarrow{G}$, $\overrightarrow{T}$, $\overleftarrow{G}$, and $\overleftarrow{T}$ can be constructed in $O(\log n)$ time with an almost-optimal processor–time bound by using the st-numbering algorithm of (E). Splitting and renaming can be achieved by making the vertex labels a 2-tuple: the first component representing the vertex label and the second representing the ear label of the edge that is incident on that vertex. We can implement this in constant time per split node. The processors assigned to the end edges of each ear can be made responsible for adding the edge to attach that ear. The lca values of nontree edges can be computed optimally by (C). Step 3(a) of *Build G'* identifies the bundles of parallel ears. Let us analyze its complexity. For building G_p the processor assigned to the last edge (a, b) of each ear examines the ear labels of the two edges incident on the $lca(a, b)$ in the fundamental cycle created by including (a, b) in $\overrightarrow{T_l}$. Let the ear labels be P and Q. Next, it checks to see if P and Q have identical pair of end vertices. If so, an edge is added in G_p between p and q. Now using the connected component algorithm of (B) we can implement Step 3(b) of *Build G'*. This leads to an algorithm for implementing *Build G'* that runs in $O(\log n)$ time and has an almost-optimal processor–time bound.

Let us examine the complexity of *Build $\mathcal{H}$*. The *pos* labeling can be computed optimally in $O(\log n)$ time with the Euler-tour technique by using (D). The auxiliary multigraph G_e can be constructed as follows. The vertex set is easy to create. The processor assigned to the first edge of an ear R does the following. (i) It finds the end vertices u and v of that ear. (ii) It also finds the ears P and Q, where $p = ear(u)$ and $Q = ear(v)$, and the values $pos(u, P)$ and $pos(v, Q)$. Assume that the vertices corresponding to P, Q, and R in G_e are p, q, and r, respectively. Finally, (iii) it creates edges (p, r) and (r, q) and labels these edge with a 2-tuple $\langle pos(u, P), pos(v, Q)\rangle$.

The α labeling of the vertices of G_e consists of the following steps. Find the blocks of G_e by using (E). Treat each block D separately, and construct a spanning tree T_b in each block by using (B). The articulation vertex q with the smallest label in a block D can be found optimally by using the Euler-tour technique. That gives us the first component of α, i.e., Q. The values a and b can be found by examining the 2-tuple labels of the edges of D incident on q. These values are broadcast to all the vertices in the block D by again using the Euler-tour technique. Finally, building H_P involves computing the minimum and maximum of the labels of edges incident on a vertex. This can be done optimally in $O(\log n)$ time by using (A).

In *Planarize $\mathcal{H}$* the only nontrivial step is the construction of the G_b. It involves the identification of the arcs (a, b) and (c, d) for each arc (x, y). An optimal algorithm for this problem is given in [1] (this problem is also known as the range-minima problem). I_P can be built by an easy reduction to the connected components.

Finally, *Find Candidate Lists* requires computing the minimum and the maximum of the labels of edges incident on a vertex. This can be done optimally in $O(\log n)$ time by using (A).

4. Algorithm for finding triconnected components.

We start with some definitions.

Let $G = (V, E)$ be a biconnected graph, and let Q be a subgraph of G. We define the *bridge graph of* $Q, S = (V_S, P_S)$ as follows (this is a little modified from the usual definition in [5], [18], and [21]). Let the bridges of Q in G be B_i, $i = 1, \dots, k$. Then $V_S = V(Q) \cup \{B_1, \dots, B_k\}$ and $P_S = E(Q) \cup \{\text{edge } (v, B_i) \text{ for each edge } (v, w) \in B_i \text{ with } w \in V(Q), 1 \le i \le k\}$. Note that S is a *multigraph*, i.e., a graph in which there can be several edges between the same pair of vertices. Each $B_i \in V_S$ together with the edges incident on B_i is a bridge of Q in S.

A *star* is a connected graph with a vertex v such that every edge in the graph is incident on v. A *star graph* $G(P)$ is a graph G consisting of a simple path P, each of whose bridges is a star. Thus if Q is a simple path in G, then S, the bridge graph of Q, is a star graph. Let B be a star in a star graph $G(P)$, where for convenience let $P = \langle 0, \dots, k-1\rangle$. Let the attachments of B on P be $v_0, \dots, v_j$, with $v_0 < v_1 < \cdots < v_j$. Then the vertices v_0 and v_j are the *end attachments* of B, and the remaining attachments are its *internal attachments.* We will also refer to v_0 as the *left attachment* of B and to v_j as its *right attachment.* The closed interval $[v_0, v_j]$ is the *span* of B, and it contains all of the vertices on P between v_0 and v_j (both vertices inclusive).

We now review some material from [11], [18], and [25] relating to triconnected components. This material deals with multigraphs. An edge e in a multigraph is denoted by (a, b, i) to indicate that it is an edge between a and b; here i is the label that distinguishes e from the other edges between a and b. The third entry in the triplet may be omitted for one of the edges between a and b.

A pair of vertices a, b in a multigraph $G = (V, E)$ is a separating pair if and only if there are either two nontrivial bridges or at least three bridges, one of which is nontrivial, of $\{a, b\}$ in G. If G has no separating pair, then G is triconnected. The pair a, b is a *nontrivial* separating pair if there are two nontrivial bridges of $\{a, b\}$ in G.

Let $\{a, b\}$ be a separating pair for a biconnected multigraph $G = (V, E)$. For any bridge X of $\{a, b\}$ let $\bar{X}$ be the induced subgraph on $(V - V(X)) \cup \{a, b\}$. Let B be a bridge of G such that $|E(B)| \geq 2$, $|E(\bar{B})| \geq 2$, and either B or $\bar{B}$ is biconnected. We can apply a *Tutte split* [11], [25] $s(a, b, i)$ to G by forming G_1 and G_2 from G, where G_1 is $B \cup \{(a, b, i)\}$ and G_2 is $\bar{B} \cup \{(a, b, i)\}$. Here i is an index that distinguishes this split from other splits that may be performed at the separating pair $\{a, b\}$. Note that we consider G_1 and G_2 to be two separate graphs. Thus it should cause no confusion that there are two edges (a, b, i) since one of these edges is in G_1 and the other is in G_2. The graphs G_1 and G_2 are called the *split graphs of G with respect to* $\{a, b\}$. The *Tutte components* of G are obtained by successively applying a Tutte split to split graphs until no Tutte split is possible. Every Tutte component is one of three types: (i) a triconnected simple graph, (ii) a simple cycle (a *polygon*), or (iii) a pair of vertices with at least three edges between them (a *bond*); the Tutte components of a biconnected multigraph G are the unique *triconnected components* of G. In this section we give an $O(\log n)$-time parallel algorithm whose processor-time bound is almost optimal to find the triconnected components of G corresponding to triconnected simple graphs and polygons. The bonds can be inferred, if necessary, by counting the number of triconnected components with respect to each separating pair.

Let $G = (V, E)$ be a biconnected graph with an open ear decomposition $D = [P_0, \ldots, P_{r-1}]$. When referring to vertices on a specified ear P_i or on a path P, we assume for convenience that they are numbered in sequence from one end vertex of the path (its *left-end vertex*) to the other (its *right-end vertex*). Let $\{a, b\}$ be a pair separating P_i. Let $B_1, \ldots, B_k$ be the bridges of P_i with no attachments outside the interval $[a, b]$ on P_i, and let $T_i(a, b) = (\cup_{j=1}^{k} B_j) \cup P_i(a, b)$, where $P_i(a, b)$ is the segment of P_i between and including vertices a and b. Then the *ear split* $e(a, b, i)$ consists of forming the *upper split graph* $G_1 = T_i(a, b) \cup \{(a, b, i)\}$ and the *lower split graph* $G_2 = \bar{T}_i(a, b) \cup \{(a, b, i)\}$. An ear split $e(a, b, i)$ is a Tutte split if either $G_1 - \{(a, b, i)\}$ or $G_2 - \{(a, b, i)\}$ is biconnected.

Let S be a nontrivial candidate list for ear P_i. Two vertices u, v in S are an *adjacent separating pair for P_i* if u and v are not adjacent to each other on P_i and S contains no vertex in the interval (u, v) on P_i. Two vertices a, b in S are an *extremal separating pair for P_i* if $|S| \geq 3$ and S contains no vertex in the interval outside $[a, b]$. An ear split on an adjacent or extremal separating pair is a Tutte split, and the Tutte components of G are obtained by performing an ear split on each adjacent and extremal separating pair [18].

With each ear split $e(a, b, i)$ corresponding to an adjacent or extremal pair separating P_i, we can associate a unique Tutte component of G as follows. Let $e(a, b, i)$ be such a split. Then by definition $T_i(a, b) \cup \{(a, b, i)\}$ is the upper split graph associated with the ear split $e(a, b, i)$. The *triconnected component of the ear split* $e(a, b, i)$, denoted by $TC(a, b, i)$, is $T_i(a, b) \cup \{(a, b, i)\}$ with the following modifications: call a pair $\{c, d\}$ separating an ear P_j in $T_i(a, b)$ a *maximal pair for* $T_i(a, b)$ if there is no e, f in $T_i(a, b)$ such that $\{e, f\}$ separates some ear P_k in $T_i(a, b)$ and c and d are in $T_k(e, f)$. In $T_i(a, b) \cup \{(a, b, i)\}$

replace $T_j(c,d)$ together with all two-attachment bridges with attachments at c and d, for each maximal pair $\{c,d\}$ of $T_i(a,b)$, by the edge (c,d,j) to obtain $TC(a,b,i)$. We denote by $TC(0,0,0)$ the unique triconnected component that contains a specified edge on P_0.

We note that $TC(a,b,i)$ as defined above is a triconnected component of G since each split of $T_i(a,b)$ in the above definition is a valid Tutte split, and the final resulting graph contains no unprocessed separating pair. Furthermore, we also note that every triconnected component of G appears as $TC(a,b,i)$ for some adjacent or extremal separating pair. This is seen as follows. Let T be a triconnected component of G. By the results in [18] we know that T can be obtained by a sequence of ear splits at adjacent and extremal pairs separating ears in the open ear decomposition D of G. Since the order of processing these ear splits is arbitrary, let us consider a sequence in which these splits are performed in nonincreasing order of ear number. In this case every upper split graph formed at the end of processing ear P_i must be a triconnected component since it will contain no unprocessed separating pairs. Let P_i be the lowest-numbered ear that contains a separating pair whose copies are present in T, and let $e(a,b,i)$ be the last ear split performed in the generation of T. Then clearly $T = TC(a,b,i)$.

In our parallel algorithm we will make the collection of splits S_1 corresponding to adjacent separating pairs simultaneously, followed by the collection of splits S_2 for extremal separating pairs. We will call each component present after completion of splits in S_1 an *adjacent triconnected component* and will denote it by $TC_A(a,b,i)$. Since the virtual edges corresponding to the splits will be inserted by concurrent writes, we will have only one copy of each such edge between a given pair of vertices. Hence we will not generate the triconnected components corresponding to bonds. These can be inferred, if necessary, by counting the number of triconnected components of the other two types that are present at each separating pair.

The rest of this section is devoted to describing an algorithm for performing these operations. We first review some further material from [18] and [21].

Let G be a biconnected graph with an open ear decomposition $D = [P_0, \dots, P_{r-1}]$. Let $B_1, \dots, B_l$ be the bridges of P_i that contain a nonattachment vertex on an ear numbered lower than i; we call these the *anchor bridges of* P_i. The *ear graph of* P_i, denoted by $G_i(P_i)$, is the graph obtained from the bridge graph of P_i by the following operations:

(a) Replace all of the anchor bridges by a new star whose attachment edges are the union of the internal attachment edges of all anchor bridges, delete the attachments of anchor bridges to the end vertices of P_i, and replace them by one new edge to each end vertex. We will call this new star the *anchoring star* of $G_i(P_i)$.

(b) Remove any multiple two-attachment bridges with the same two attachments, and also remove any two-attachment bridge with the end vertices of P_i as attachments.

Note that $G_i(P_i)$ is a multigraph. (This definition of ear graph is slightly modified from that in [18] and [21] to reflect the change made in the definition of a bridge graph.) $G_i(P_i)$ is also a star graph.

Two stars S_j and S_k in a star graph $G(P)$ *interlace* (see also [5, p. 149]) if one of the following holds:

1. There exist four distinct vertices a, b, c, d in increasing order in P such that a and c belong to $S_j(S_k)$ and b and d belong to $S_k(S_j)$.

2. There are three distinct vertices on P that belong to both S_j and S_k.

The operation of *coalescing* two stars S_j and S_k is the process of forming a single new star S_l from S_j and S_k by combining the attachments of S_j and S_k and deleting S_j and S_k. Given a star graph $G(P)$, the *coalesced graph* $G_c(P)$ of $G(P)$ is the graph obtained from G by coalescing all pairs of stars that interlace. Note that $G_c(P)$ is a star graph with respect to P and that $G_c(P)$ has a planar embedding with P on the outer face since no pair of stars interlace on P.

Let $G(P)$ be a star graph in which no pair of stars interlace. If $G(P)$ contains no star that has attachments to the end vertices x and y of P, then add a virtual star X to $G(P)$ with attachments to x and y. The *star embedding* $G^*(P)$ of $G(P)$ is the planar embedding of (the possibly augmented) $G(P)$ with P on the outer face. A star B is the *parent-star* of star B' and B' a *child-star* of B if there is a face in the star embedding $G^*(P)$ that contains the left and right attachments x and y of B' as well as an attachment edge of B in each of the intervals $[l,x]$ and $[y,r]$, where l and r are the left and right end vertices of P.

The following lemma is shown in [18].

LEMMA 4.1. *A pair $\{a,b\}$ separates P_i in the coalesced graph $G_{i_c}(P_i)$ if and only if $\{a,b\}$ separates P_i in G.*

We will use the following corollary to Lemma 4.1.

COROLLARY 4.2. *An edge (x,y) incident on P_i is in $TC(a,b,i)$ if and only if (x,y) is in the triconnected component associated with pair $\{a,b\}$ separating P_i in $G_{i_c}(P_i)$.*

Proof. Let $C_i(P_i)$ be the bridge graph of P_i, and let $C_{i_c}(P_i)$ be its coalesced graph. A straightforward extension of the proof of Lemma 4.1 given in [18] shows that an edge incident on P_i is in $TC(a,b,i)$ if and only if it is in the triconnected component associated with the pair $\{a,b\}$ separating P_i in $C_{i_c}(P_i)$. We then observe that the edges of $C_i(P_i)$ that are deleted in the ear graph $G_i(P_i)$ cannot appear in $TC(x,y,i)$ for any pair x,y separating P_i. □

For convenience of notation we denote $G_{i_c}(P_i)$ by $G_c(P_i)$. Reference [21] includes an algorithm for forming the coalesced graph of a star graph $G(P)$ that runs in logarithmic time on a CRCW PRAM with an almost-optimal processor-time bound. This algorithm has the same processor-time complexity as that of finding connected components.

LEMMA 4.3. *In the coalesced graph $G_c(P_i)$, for each adjacent pair $\{a,b\}$ separating P_i there is at most one bridge of P_i with attachments on a,b and a vertex in (a,b), the portion of P_i between a and b.*

Proof. Suppose not, and let B_1 and B_2 be two bridges of P_i in $G_c(P_i)$ that have attachments on a,b and a vertex in (a,b). Then B_1 and B_2 must interlace, which contradicts the fact that $G_c(P_i)$ is the coalesced graph of the ear graph $G_i(P_i)$. □

LEMMA 4.4. *Let B be a two-attachment bridge of P_i in $G_c(P_i)$ with attachments a and b. Then*

(a) *If the span $[a,b]$ is degenerate (i.e., (a,b) is an edge in P_i) or if there is a bridge B' of P_i with attachments on a and b and at least one other vertex, then $G_c(P_i)-\{B\}$ defines the same set of polygons and simple triconnected components $TC(x,y,i)$, for i fixed, as $G_c(P_i)$.*

(b) *If part (a) does not hold, then $\{a,b\}$ is an extremal pair separating P_i as well as an adjacent pair separating P_i.*

Proof. Let P_j be the lowest-numbered ear in B. Then $j>i$, and a and b are the end vertices of P_j. Hence the ear split $e(a,b,j)$ separates B from P_i, and thus B is not part of $TC(x,y,i)$ for any pair $\{x,y\}$ separating P_i. So a 2-attachment bridge on P_i is never a part of a triconnected component associated with a pair separating P_i, although it may define some adjacent and extremal separating pairs as in case (b) of the lemma.

We now prove parts (a) and (b) of the lemma.

(a) Suppose span $[a,b]$ is degenerate. Then the triconnected component associated with split $e(a,b,i)$ is the single edge (a,b), which is a bond. Otherwise, if there is a bridge B' with attachments on a,b and at least one other vertex v, then the triconnected component associated with split $e(a,b,i)$ contains a portion of P_i between a and b together with B' if v is in the interval (a,b) and is a polygon if v is not in $[a,b]$. Both of these situations can be inferred without the presence of B. Note that it is not possible for B' to have an attachment v in the interval (a,b) and another attachment w that is not in $[a,b]$ since the bridge B would interlace with B' in such a case.

(b) Let the span $[a,b]$ be nondegenerate, and let the portion of P_i between a and b be $\langle a=a_1,a_2,\ldots,a_k=b\rangle$. Since there is no k-attachment bridge, $k>2$, with span $[a,b]$, there must exist an a_i, $1<i<k$, such that a, a_i, and b are in the same candidate list C and no vertex outside $[a,b]$ is in C. Hence $\{a,b\}$ is an extremal separating pair. Also, since there is no bridge with attachments on a,b and some other vertex c outside $[a,b]$, there must be some vertex c on P_i such that either $c<a<b$ or $a<b<c$ and a, b, and c are in the same candidate list C'. Furthermore, no vertex in the interval (a,b) can belong to C'. Hence $\{a,b\}$ is an adjacent pair in the candidate list C'. □

Let $\{a,b\}$ be an adjacent separating pair for ear P_i. The pair a,b is a *nonvacuous adjacent separating pair for* P_i if there is a bridge of P_i in $G_c(P_i)$ with attachments on a,b and one other vertex in the interval (a,b) on P_i; otherwise, the pair $\{a,b\}$ is a *vacuous adjacent separating pair.* We leave it as an exercise to verify that if $\{a,b\}$ if a nonvacuous adjacent separating pair, then $TC(a,b,i)$ is a simple triconnected graph and if $\{a,b\}$ is a vacuous adjacent separating pair, then $TC(a,b,i)$ is a bond; if $\{a,b\}$ is an extremal separating pair, then $TC(a,b,i)$ is a polygon.

Lemmas 4.3 and 4.4, in conjunction with Corollary 4.2, tell us that we can compute the triconnected components of G by the following method. Make the splits corresponding to the adjacent separating pairs by performing, for each star B in $G_c(P_i)$, an ear split $e(a,b,i)$, where $[a,b]$ is the span of B. Then break off chains of degree-2 vertices on the paths in the resulting star graphs to perform the splits corresponding to the extremal separating pairs.

There are two problems with using the above approach in an efficient logarithmic-time algorithm for forming the triconnected components of a graph. One is that we are working with the ear graphs of the ears and the total size of these graphs need not be linear in the size of G. The second is that this approach will not work if a vertex a appears in an ear split for two different ears. In particular, two-attachment bridges corresponding to adjacent separating pairs would be separated on two different ears, and this would cause processor conflicts.

We now turn to G', the local replacement graph of G that we defined in §3.1, in order to develop an efficient method of identifying the associated triconnected components.

Let G' be the local replacement graph of G, and let $D'=[P'_0,\ldots,P'_{r-1}]$ be the corresponding open ear decomposition. By Corollary 3.16 a pair $\{a,b\}$ separates P_i in G if and only if the pair $\{a_P,b_{P_i}\}$ separates P'_i in G'. Furthermore, neither a_{P_i} nor b_{P_i} is an end vertex of P'_i. The following lemma shows that in G' we can efficiently identify any bridge B of an ear P'_i that has no attachment to an end vertex of P'_i.

LEMMA 4.5. *Let G_e be the graph obtained from G' by collapsing all internal vertices of each ear into a single vertex. Let vertex v_i represent ear P'_i in G_e. Then the edges incident on v_i in each block of G_e whose lowest-numbered vertex is v_i correspond to the attachment edges of a bridge of P'_i in G', and, conversely, each bridge of P'_i in G' that has no attachments to the end vertices of P'_i corresponds to a block of G_e.*

Proof. Let e_1 and e_2 be any pair of edges incident on v_i that lie in the same block B of G_e whose lowest-numbered vertex is v_i. Then there is a path between e_1 and e_2 in B that avoids v_i, and hence in G' there is a path between e_1 and e_2 that avoids internal vertices of P'_i. But since the lowest-numbered vertex in B is v_i, the path between e_1 and

e_2 in B does not contain any vertex on an ear numbered lower than i, and hence e_1 and e_2 must lie in a connected component in $G' - \{P_i'\}$.

Conversely, let B be a bridge of P_i' in G' that has no attachments to the end vertices of P_i'. Then when the internal vertices of P_i' are collapsed into v_i, all of the attachments of B on P_i' become incident on v_i. Thus B becomes a block in G_e with articulation vertex v_i. Furthermore, since B has no attachments to the end vertices of P_i', B is not an anchor bridge of P_i' and hence v_i is the minimum-numbered vertex in B in G_e. □

Recall that (Proposition 3.7) the copies of a vertex v in G' are connected in the form of a tree. For the following lemma assume this local tree that replaces v is rooted at v_S, where $ear(v) = S$.

LEMMA 4.6. *Let $\{x, y\}$ be a separating pair that separates P in G. Let C be a connected component in $G - \{x, y\}$ that contains $P(x, y)$. If Q is an ear label of one of the edges of C and if x is one of the end vertices of Q, then x_P is an ancestor of x_Q in the local tree that replaces x.*

Proof. If Q if parallel to P, then as $\{x, y\}$ separates P it has the smallest ear label among the labels of edges of C. Hence P would have to be the representative of the bundle containing Q, and the lemma is clearly true. Otherwise, notice that $V(C) - V(P)$ consists of nonattachment vertices of a relevant bridge of P. If the *lca* of the end vertices of Q is not x, then x_Q is a child of x_P by Step 2 of *Build G'*. Otherwise, by Lemma 3.12, x_Q is a descendant of x_P. □

LEMMA 4.7. *Any bridge of P_i' in G' with an attachment to an end vertex of P_i' must be either part of the anchoring star of $G_i'(P_i')$ or a bridge of P_i' with attachments only to the end vertices of P_i'.*

Proof. Let B be a bridge of P_I' in G' with an attachment to one of its end vertices x_{P_j}.

We first show that the internal vertices on P_j' are part of the anchoring star of P_i'. If P_j is not parallel to P_i, then $j < i$ and the result follows directly. If P_j is parallel to P_i, then let C be the connected component constructed in Step 3 of *Build G'* that contains P_i and P_j and let P_l be the root of the spanning tree of C constructed in that step. Hence $l \leq j$ and x_{P_l} is an ancestor of x_{P_j} in LT_x, where LT_x is the local tree that replaces the vertex x in G'. Furthermore, by the construction in Step 3 of *Build G'* there is a path in G' between an internal vertex of P_j' and an internal vertex of P_l' that avoids all vertices on P_i'. Hence the vertices on P_j' belong to an anchor bridge of P_i'.

Let $e = (y, x_{P_j})$ be an attachment edge of bridge B of P_i'. We will show that B is an anchor bridge of P_i'. Let e belong to ear P_k'.

If $y \neq x_{P_k}$, then e is an edge on P_j'. Hence e, and thus B, are parts of the anchoring star of P_i'. If $y = x_{P_k}$, then consider the fundamental cycle completed by the nontree edge (u, v) in P_k' in the tree $\vec{T}$ in which (x_{P_j}, x_{P_k}) is a tree edge. If P_k is not parallel to P_i, then the presence of edge (x_{P_k}, x_{P_j}) in G' implies either that this fundamental cycle contains an edge on P_j' and no vertex on P_i', or that there is a path from v to the root s of G' that avoids all vertices in P_i'. In either case, e is part of a bridge of P_i' that contains a nonattachment vertex on an ear numbered lower than i.

If P_k and P_i are parallel to each other, then if P_j is not parallel to P_i, each of P_i and P_k correspond to the root of the spanning tree of a connected component constructed in Step 3 of *Build G'*. Hence by Lemma 4.6 B is a bridge of P_i' with no internal attachment on P_i'. Finally, if P_i, P_j, and P_k are all parallel to each other, then since x_{P_j} is the parent of x_{P_k} in LT_x, there is a path in G' between an internal vertex of P_k' and an internal vertex of P_j' that avoids all vertices in P_i'. Hence e is part of the bridge of P_i' that contains the internal vertices of P_j'. This bridge was shown to be an anchor bridge of P_i'. □

Lemmas 4.5 and 4.7 tell us that the following algorithm generates the ear graph of each ear in G'.

ALGORITHM. *Ear Graphs of G'.*
Input: A local replacement graph G' with its associated open ear decomposition $D' = [P'_0, \dots, P'_{r-1}]$.

```
1. Form $G_e$.
2. For each block $B$ in $G'$ do
   (a) Let the minimum-numbered vertex in $B$ be $v_i$. make the image $e$ in $G'$ of
       each edge $e'$ in $B$ incident on $v_i$ an attachment edge of nonanchor bridge
       $B$ in the ear graph of $P'_i$.
   (b) For each vertex $v_j \neq v_i$ in $B$ make the image $e$ in $G'$ of each edge $e'$ of $B$
       incident on $v_j$ an attachment edge of the anchoring star of the ear graph if
       $P'_j$.
3. For each ear $P'_i$ add attachment edges to the end vertices of $P'_i$ for the anchoring
   star created in Step 2(b).
```

Step 1 is the same as Step 1 of *Build $\mathcal{H}$* applied to G' (§3.2). Steps 2 and 3 can be implemented in constant time per edge by using the α-values of each vertex computed in Step 3 of *Build $\mathcal{H}$*. The total size of all of the ear graphs is $O(m)$, where m is the number of edges in G', since each edge in G' appears in at most two ear graphs (corresponding to the ears containing the two end vertices of the edge).

Having obtained the ear graph $G'_i(P'_i)$ of each ear in G', we can obtain the coalesced graph $G'_c(P'_i)$ of each of the ear graphs by using the algorithm of [21]. By Corollary 3.16 and Lemma 4.1 a pair $\{x_{P_i}, y_{P_i}\}$ is an adjacent (extremal) pair separating P'_i in $G'_c(P'_i)$ if and only if $\{x, y\}$ is an adjacent (extremal) pair separating P_i in $G_c(P_i)$. It turns out that the relation between G and G' extends beyond separating pairs to triconnected components. The following two lemmas allow us to related the bridges of ears in G' with the bridges of ears in G and hence develop an efficient algorithm for finding the triconnected components of G.

LEMMA 4.8. *Let x be a vertex in P_i in G (possibly its end vertex), and let $e_1 = (u_1, x) \in P_j$ and $e_2 = (u_2, x) \in P_k$ be two edges incident on x that belong to different bridges (B_1 and B_2, respectively) of P_i, each of which has an internal attachment on P_i. Then the least common ancestor (lca) of x_{P_j} and x_{P_k} in LT_x is x_{P_p}, where x_{P_p} is an ancestor of x_{P_i} in LT_x.*

Proof. Suppose not, and let x_{P_l} be $lca(x_{P_j}, x_{P_k})$, where x_{P_l} is a proper descendant of x_{P_i}.

Case 1. There are no parallel ears incident on x in G. Let the vertices on the path from x_{P_l} to x_{P_j} in LT_x be $x_{P_{j1}} = x_{P_j}, x_{P_{j2}}, \dots, x_{P_{jr}} = P_l$. Then by construction the fundamental cycle of P_{j_h} contains an edge in $P_{j_{h+1}}$, $h = 1, \dots, r-1$ and contains no edge in P_i in G. A similar situation holds for x_{P_k}. But then all of these ears would lie in the same bridge of P_i.

Case 2. There are some parallel ears incident on x in G. Again by construction a pair of parallel ears have an ancestor–descendant relationship in LT_v only if they are connected to each other by a path that avoids P_i and their two end vertices. Hence again by a combination of this observation and the argument in Case 1 we deduce that P_j and P_k must be in the same bridge of P_i. □

LEMMA 4.9. *Let $\{a, b\}$ be an adjacent or extremal pair separating P_i in G, and let $\mathbf{B} = \{B_1, \dots, B_r\}$ be the bridges of P_i with an internal attachment in (a, b). Similarly, let $\mathbf{B}' = \{B'_1, \dots, B'_{r'}\}$ be the bridges of P'_i in G' with an internal attachment in (a_{P_i}, b_{P_i}).*

Then $r = r'$, and there is a one-to-one correspondence between the bridges in $\mathbf{B}$ and the bridges in $\mathbf{B}'$ (without loss of generality we assume that the correspondence is between B_i and B_i' for $i = 1, \ldots, r$) such that an edge e is in B_i if and only if the corresponding edge e' is in B_i'.

Proof. We need only to verify the connectivity at LT_v in $G' - \{P_i'\}$, for $v \in P_i$, since the connectedness in the rest of the graph remains unaltered when a vertex u in G is replaced by the tree LT_u in G'. But we note from Lemma 4.8 that if two edges $e_1 = (u, v) \in P_j$ and $e_2 = (u_2, v) \in P_k$, $v \in P_i$, are in different bridges of P_i, then e_1 and e_2 are separated from each other in $G' - \{v_{P_i}\}$. The lemma follows. □

From Lemma 4.9 we see that given an adjacent pair $\{a, b\}$ separating P_i, the bridges of P_i' with no attachments outside the interval $[a_{P_i}, b_{P_i}]$ on P_i, together with the path from a_{P_i} to b_{P_i} on P_i', will correspond to the upper split graph of the ear split $e(a, b, i)$ in G. Now we can further apply Corollary 4.2 to G' and can work with $G_c'(P_i')$ to directly identify the triconnected components of G. This is done in the following algorithm.

ALGORITHM. *Triconnected Components.*

Input: A biconnected graph G with an open ear decomposition $D = [P_0, \ldots, P_{r-1}]$ and its local replacement graph G' together with its associated open ear decomposition $D' = [P_0', \ldots, P_{r-1}']$ and the coalesced graph $G_c(P_i')$ of the ear graph of each ear in D'.

```
1. For each ear $P_i'$ do
      For each vertex $v$ on $P_i'$ make a copy $v_B$ of $v$ for each star $B$ in $G_c(P_i')$ that
      has an attachment on $v$. If there is no star with an internal attachment on
      $v$, then make an additional copy $v_P$ of $v$ to represent the lower split graph
      formed when all adjacent pairs containing $v$ have been processed.
2. Assign vertices to edges on $P_i'$.
   (a) For $j = 0, 1, \ldots, k-1$ do
          If there is no bridge with its leftmost attachment on $j$, then replace
          edge $(j, j+1)$ on $P_i'$ by an edge incident on $j_C$, where $C$ is $B$ is there
          is a bridge $B$ with an internal attachment on $j$ and is $P$ otherwise.
   (b) For $j = 1, \ldots, k$ do
          If there is no bridge with its rightmost attachment on $j$, then replace
          edge $(j-1, j)$ on $P_i'$ by an edge incident on $j_D$, where $D$ is $B'$ if there
          is a bridge $B'$ with an internal attachment on $j$ and is $P$ otherwise.
3. Make the splits corresponding to adjacent separating pairs:
      For each star $B$ in $G_c(P_i')$ do
          Let the end attachments of $B$ on $P_i'$ be $v$ and $w, v < w$.
          (a) Replace all edges in $B$ incident on $v$ by edges incident on $v_B$. Sim-
              ilarly, replace all edges in $B$ incident on $w$ by edges incident on
              $w_B$.
          (b) If $B$ has no child-star with an attachment at $v$, then replace edge
              $(v, v+1)$ on $P$ by an edge incident on $v_B$. Similarly, if $B$ has no
              child-star with an attachment at $w$, then replace edge $(w-1, w)$
              by an edge incident on $w_B$.
          (c) Add a virtual edge $(v_B, w_B, i)$ and another virtual edge $(v_C, w_D, i)$,
              where $C$ (respectively, $D$) is the parent-star of $B$ is the parent-star
              of $B$ has an attachment at $v$ (respectively, $w$) and is $P$ otherwise.
          (d) Replace each internal attachment edge of $B$ on a vertex $u$ in $P_i'$ by
              an edge incident on $u_P$.
4. Process extremal pairs:
      For each star $B$ in $P_i'$ do
```

```
        Let the attachments of $B$ on $P_i'$ be $v_0 < v_1 < \cdots < v_l$.
        For $j = 0, \ldots, l-1$ do
            If $(v_{j_B}, v_{j+1_B})$ is not an edge in the current component containing
            $B$, then
            For convenience of notation let $x$ denote $v_j$ and let $y$ denote $v_{j+1}$.
            (a) Make a copy $x_{B_r}$ of $x$ and a copy $y_{B_l}$ of $y$.
            (b) Replace the edge on $P_i'$ connecting $x_B$ to the next larger vertex
                in the current graph by an edge incident on $x_{B_r}$.
            (c) Replace the edges on $P_i'$ connecting $y_B$ to the next smaller
                vertex in the current graph by an edge incident on $y_{B_l}$.
            (d) Add virtual edges $(x_B, y_B, i)$ and $(x_{B_r}, y_{B_l}, i)$.
    5. Convert the vertices in $G'$ into vertices in $G$.
        In each of the components formed collapse all vertices that correspond to
        a given vertex $v$ in $G$ into a single copy of $v$ to construct the triconnected
        components of $G$.
```

THEOREM 4.10. *Algorithm Triconnected Components correctly finds the simple triconnected components and the polygons of G.*

Proof. Consider a bridge B' in $G_c(P_i')$ with span $[x_{P_i}, y_{P_i}]$. By Corollary 3.16 and Lemma 4.9 we can map each edge e' in B' (that is not in any LT_v) to an edge e in a bridge B of $G_c(P_i)$ with span $[x, y]$. A similar argument holds for the bridges of P_i' in $G_c(P_i')$ corresponding to the maximal pairs in $T_i(x, y)$. Finally, any two-attachment bridge B'' with attachments c_{P_i} and d_{P_i} on P_i' is split off in P_j' at c_{P_j} and d_{P_j}, where P_j' is the minimum-numbered ear in B''. Hence when we make the split corresponding to B' in Step 3 of *Triconnected Components,* the edges in the component formed must correspond to the edges in the adjacent triconnected component $TC_A(x, y, i)$. Finally, the polygons generates in Step 4 are clearly the polygons of the triconnected components of G since all vertices on a polygon are local to a given ear.

Thus when we implement Step 5 of the algorithm in a component to get back original vertices of G, we get back a triconnected component of G. □

For the processor–time complexity of *Triconnected Components* we note that steps 1, 2, and 4 can be performed optimally in logarithmic time, as can all of the steps in Step 3 except Step 3(c), which requires identifying the parent-star of a star in a star embedding. This step can be performed by using the bucket-sort algorithm of Hagerup [10]. It can also be performed optimally in logarithmic time by using list ranking and making use of the fact that $G_c(P_i')$ is planar. The details of this implementation are given in [8]. They are omitted here since the overall complexity of the algorithm is dominated by the need to perform bucket sort in order to obtain the adjacency lists of the various graphs. Step 5 can be performed with the same bounds as those used in finding the connected components of a graph.

Hence *Triconnected Components* runs in $O(\log n)$ time deterministically on a CRCW PRAM while performing $O((m + n) \log \log n)$ work.

5. Conclusion.

We have presented an efficient parallel algorithm for dividing a graph into triconnected components. We conclude the paper with the following remarks.

1. Our algorithm can be adapted to test 3-edge connectivity within the same bounds. For this we use an ear decomposition instead of an open ear decomposition, and we look for separating pairs of edges. It turns out that in this case it is not necessary to construct the local replacement graph since each edge of the graph is contained in exactly one

ear. Hence the resulting algorithm is simpler than the one we have presented for testing (vertex) triconnectivity.

2. Our parallel algorithm is slightly suboptimal in the work it performs because of the suboptimality of the currently known parallel algorithms for finding connected components and performing bucket sort. It will be interesting to find improvements in these parallel algorithms, which in turn will lead to improvements in the bounds for our algorithm.

REFERENCES

[1] N. ALON AND B. SCHIEBER, *Optimal preprocessing for answering on-line queries*, Tech. Rep. 71/87, Tel Aviv University, Tel Aviv, Israel, 1987.

[2] J. CHERIYAN AND R. THURIMELLA, *Algorithms for parallel k-vertex connectivity and sparse certificates*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 391–401.

[3] R. COLE AND U. VISHKIN, *Approximate parallel scheduling,* II: *Applications to optimal parallel graph algorithms in logarithmic time*, Inform. Comput., 91 (1991), pp. 1–47.

[4] S. EVEN, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.

[5] D. FUSSELL AND R. THURIMELLA, *Separation pair detection*, in VLSI Algorithms and Architectures, Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988, pp. 149–159.

[6] D. FUSSELL, V. RAMACHANDRAN, AND R. THURIMELLA, *Finding triconnected components by local replacements*, in Proc. ICALP '89, Lecture Notes in Computer Science 372, Springer-Verlag, Berlin, New York, 1989, pp. 379–393.

[7] H. N. GABOW, *A matroid approach to finding edge connectivity and packing arborescences*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 112–122.

[8] T. HAGERUP, *Towards optimal parallel bucket sorting*, Inform. and Comput., 75 (1975), pp. 39–51.

[9] J. E. HOPCROFT AND R. E. TARJAN, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.

[10] J. JÁJÁ AND J. SIMON, *Parallel algorithms in graph theory: Planarity testing*, SIAM J. Comput., 11 (1982), pp. 314–328.

[11] A. KANEVSKY AND V. RAMACHANDRAN, *Improved algorithms for graph four-connectivity*, J. Comput. System Sci., 42 (1991), pp. 288–306.

[12] R. M. KARP AND V. RAMACHANDRAN, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[13] Y. MAON, B. SCHIEBER, AND U. VISHKIN, *Parallel ear decomposition search* (EDS) *and ST-numbering in graphs*, Theoret. Comput. Sci., 47 (1986), pp. 277–298.

[14] G. L. MILLER AND J. REIF, *Parallel tree contraction, part 2: Further applications*, SIAM J. Comput., 20 (1991), pp. 1128–1147.

[15] G. L. MILLER AND V. RAMACHANDRAN, *Efficient parallel ear decomposition with applications*, Manuscript, MSRI, Berkeley, CA, January 1986.

[16] ———, *A new triconnectivity algorithm and its applications*, Combinatorica, 12 (1992), pp. 53–76.

[17] H. NAGAMOCHI AND T. IBARAKI, *Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[18] V. RAMACHANDRAN, *Parallel open ear decomposition and its application to graph biconnectivity and triconnectivity*, in Synthesis of Parallel Algorithms, J. H. Reif, ed., Morgan-Kaufmann, to appear.

[19] V. RAMACHANDRAN AND U. VISHKIN, *Efficient parallel triconnectivity in logarithmic time*, in VLSI Algorithms and Architectures, Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, New York, 1988. pp. 33–42.

[20] B. SCHIEBER AND U. VISHKIN, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[21] R. E. TARJAN AND U. VISHKIN, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1984), pp. 862–874.

[22] W. T. TUTTE, *Connectivity in Graphs*, University of Toronto Press, Toronto, Canada, 1966.

[23] H. WHITNEY, *Non-separable and planar graphs*, Trans. Amer. Math. Soc., 34 (1932), pp. 339–362.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 617–626, June 1993

IMPROVED PARALLEL POLYNOMIAL DIVISION*

DARIO BINI† AND VICTOR PAN‡

Abstract. The authors compute the first N coefficients of the reciprocal $r(x)$, $r(x)p(x) = 1 \bmod x^N$ (given a natural N and a polynomial $p(x)$), ($p(0) \neq 0$), by using $O(h \log N)$ arithmetic steps and $O(\,(N/h)(1+2^{-h}\log^{(h)} N)\,)$ processors, for any h, $h = 1, 2, \ldots, \log^* N$, under the PRAM arithmetic models, provided that $O(\log m)$ steps and m processors suffice to perform discrete Fourier transforms on m points and that $\log^{(0)} N = N$, $\log^{(h)} N = \log_2 \log^{(h-1)} N$, $h = 1, \ldots, \log^* N$, $\log^* N = \max\{h : \log^{(h)} N > 0\}$. The same estimates apply to some other computations, such as the division with a remainder of two polynomials of degrees $O(N)$ and the inversion of an $N \times N$ triangular Toeplitz matrix. This improves the known estimates of Reif–Tate and Georgiev. The presented techniques are extended to parallel implementation of other recursive processes, such as the evaluation modulo x^N of the mth root $p(x)^{1/m}$ of $p(x)$ (for any fixed natural m), for which we need $O(\log N \log\log N)$ timesteps and $O(N/\log\log N)$ processors. The paper demonstrates some new techniques of supereffective slowdown of parallel algebraic computations combined with the technique of stream contraction.

Key words. polynomial division, parallel algorithms, computational complexity, triangular Toeplitz matrices, Newton's iteration for algebraic computing

AMS(MOS) subject classifications. 68C20, 68C25, 68C05

1. Introduction.

In this paper we will improve the known upper estimates for the parallel arithmetic complexity of computing the reciprocal of a polynomial and, consequently, of the equivalent computations, such as polynomial and power series division and triangular Toeplitz matrix inversion (compare [BP], [BPa] and the appendix). This improvement relies on the techniques of *stream contraction* and *supereffective slowdown of parallel computations*. The first technique contracts the stream of successive steps of some recursive parallel algorithms in a pipelined fashion, by modifying these algorithms so as to start each new recursive step without completing the original preceding step. In [PR] such an approach improved parallel computation of paths in graphs; in our paper, it works (in a different form) for parallel polynomial division. The second technique applies to parallel recursive algebraic computations, such as Newton's iteration for algebraic problems, where each recursive step increases by the factor c (for a fixed $c > 1$) the number of computed output values. This technique *recursively stops and restarts* the computations, which enables us to decrease by the factor q the processor bound for (a large class of) parallel algebraic computations in the result of their slowdown by $s = o(q)$ times. We call such a slowdown *supereffective* since a variant of Brent's scheduling principle only implies an effective, by $O(s)$ times, decrease of the processor bound, in this case (see [KR], [PP]).

We focus on the application of our approach to polynomial division, but our techniques can be immediately extended to some other polynomial computations based on

*Received by the editors June 8, 1990; accepted for publication (in revised form) January 29, 1992. The results of this paper were presented at the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '92), Pittsburgh, Pennsylvania, 1992.

†Department of Mathematics, University of Pisa, 56100 Pisa, Italy. The work of this author was supported by National Science Foundation grants CCR 8805782 and CCR 9020690, and by Ministero dell'Università e della Ricerca Scientifica e Tecnologica (40% funds).

‡Mathematics and Computer Science Department, Lehman College, City University of New York, Bronx, New York 10468. The work of this author was supported by National Science Foundation grants CCR 8805782 and CCR 9020690 and by Professional Staff Congress of the City University of New York Awards #661340, #668541, #669210, and #662478.

recursive processes, such as Newton's iteration. In particular, in the appendix we show an extension to computing (modulo x^N) the $(\frac{1}{m})$th power (or the mth root), $p(x)^{1/m}$, of a polynomial $p(x)$, summarize the techniques used, and state a more general theorem on the resulting bounds on the complexity of parallel computations.

We will state our estimates in the form $O_A(t, P)$, which means that our computations can be performed by using $O(st)$ arithmetic steps and $O(P/s)$ processors (for any fixed s, $1 \leq s \leq P$), under the PRAM arithmetic models of computing (see [KR]). We will assume the bounds $O_A(\log m, m)$ on the cost of performing (forward and inverse) discrete Fourier transforms on the mth roots of 1 [hereafter referred to as $DFT(m)$]. This bound holds, in particular, over the complex field of constants. Over any commutative ring (with unity), the bound $O_A(\log m, m \log\log m)$ holds [CK], so that all our estimates apply if their processor bounds are increased by the factor of $\log\log N$ provided that the ring supports Bluestein's generalized fast Fourier transform (FFT) ([Kn]). In § 6 we will comment on the latter assumption. Hereafter, log stands for $\log_2$ (all logarithms are binary).

Given a natural N and the coefficients of a polynomial $p(x)$, $p(0) \neq 0$, we will compute the first N coefficients of the reciprocal power series,

$$r(x) = \sum_{i=0}^{\infty} r_i x^i, \qquad r(x)p(x) = 1, \qquad r_0 = \frac{1}{p(0)}, \tag{1.1}$$

yielding the complexity estimates

$$c_N = O_A\left(h \log N, \frac{N}{h}(1 + 2^{-h} \log^{(h)} N)\right), \quad \text{for all } h, \qquad h = 1, 2, \ldots, \log^* N. \tag{1.2}$$

Here,

$$\log^{(0)} N = N\ , \ \log^{(h)} N = \log_2 \log^{(h-1)} N, \quad h = 1, \ldots, \log^* N,$$

$$\log^* N = \max\{h : \log^{(h)} N > 0\}.$$

In particular, for $h = \log^* N$, (1.2) turns into

$$c_N = O_A(\log^* N \log N,\ N/\log^* N). \tag{1.3}$$

This improves the previous record bounds of [G],

$$c_N = O_A(\log N,\ N \log^2 N),$$

and of [RT],

$$c_N = O_A(\log N \log\log N,\ N/\log\log N).$$

We refer the reader to [BP] and [BPa] on other previous works on parallel algorithms for polynomial reciprocals, in particular, on Reif's results of [R], including the bound $c_N = O_A(\log N, N^2)$, and Bini's results of [B], where $c_N = O_A(\log N, N)$ is proven for any precision approximation to the reciprocal of a polynomial modulo x^N.

2. Newton's iteration and its contraction. It is well known that, given a polynomial $p(x)$, $p(0) \neq 0$, the first m coefficients of the reciprocal power series $r(x)$ of (1.1) can be computed by performing $\lceil \log m \rceil$ steps of Newton's iteration, (2.1) $z_0(x) = 1/p(0), \quad z_{i+1}(x) = z_i(x)\,(2 - p(x)z_i(x)\,), \quad i = 1, \ldots, \lceil \log m \rceil.$

In fact, since

(2.2) $$r(x) - z_{i+1}(x) = \big(r(x) - z_i(x)\,\big)^2 p(x),$$

the number of coefficients shared by $r(x)$ and $z_i(x)$ is doubled in each iteration step i, so that, for all i,

(2.3) $$z_i(x) = r(x) \bmod x^{2^i}.$$

The algorithm of Sieveking–Kung is based on this scheme, where, however, both polynomials $p(x)$ and $z_i(x)$ of (2.1) are reduced modulo x^{2^i}. It follows that step i of Newton's process outputs a polynomial of degree at most $3(2^i-1)$, whose coefficients can be computed by means of forward and inverse DFTs on $m(i)$ points at the cost $O_A(i, 2^i)$, where $m(i) > 3(2^i - 1)$. This implies the bound $O_A(\log^2 n, n/\log n)$ on the overall cost of computing the coefficients of $r(x) \bmod x^n$, which turns into the bound

(2.4) $$c_{n,k} = O_A\left(\log n \log\left(\frac{n}{k}\right),\ n/\log\left(\frac{n}{k}\right)\right)$$

if we initially know the k coefficients of $r(x) \bmod x^k$.

In [G] it is pointed out that the relation (2.3) still holds even if $p(x)$ in (2.1) is replaced by $p(x) \bmod x^{2^{i+1}}$. Then the degrees of the polynomials $z_i(x)$ are bounded by $i2^i$, $i = 0, 1, \ldots$, and thus, the coefficients of $z_d(x)$, for $d = \lceil \log n \rceil$, can be computed by means of a single inverse DFT on $N = O(n \log n)$ points (that is, on the Nth roots of 1) once the values of $z_d(x)$ on these points have been computed. To compute the values of $z_d(x)$ on a set of points, we may compute on this set, at first, the values of $p_i(x) = p(x) \bmod x^{2^{i+1}}$, for all $i < d$, and then, recursively, the values of $z_{i+1}(x)$, by applying the equation (2.1) for $i = 0, 1, \ldots, d-1$. This enables us to *contract the stream* of Newton's recursive steps so as to avoid the application of DFT until we compute the values of $z_d(x)$. In such a way, the first n coefficients of $r(x) \bmod x^n$ can be computed at the cost $O_A(\log n, n \log^2 n)$ [G].

3. A basis algorithm for new improvements and its recursive application. In this and in the next two sections, we will combine the two ways of applying Newton's iteration (2.1), that is, the coefficientwise way (Sieveking–Kung) and the pointwise way [G], applied *with recursive restarting*. This policy enables us to decrease the overall parallel computational cost. Specifically, we will consider Newton's iteration that starts with

(3.1) $$y_0(x) = r(x) \bmod x^k$$

for any fixed natural k (not necessarily for $k = 1$) and that continues as follows:

(3.2) $$p_i(x) = p(x) \bmod x^{k2^{i+1}},$$

(3.3) $$y_{i+1}(x) = y_i(x)\,\big(2 - p_i(x)\,y_i(x)\,\big),$$

$i = 0, 1, \ldots$. Then (2.2) immediately implies that

(3.4) $$y_i(x) = r(x) \bmod x^{k2^i},$$

and we verify by induction on i that

$$\deg y_i(x) \le (i+1)k\,2^i - 2^{i+1} + 1. \tag{3.5}$$

Equations (3.4) and (3.5) immediately imply the correctness of our basis algorithm shown below.

With no loss of generality, we assume that n is large enough, say, $n \ge 16$. Furthermore, let s denote an integer, $4 \le s \le n$; let $\log\log s$ and $\log n$ be integers.

Basis algorithm.

Input: Integers n and s and the coefficients of $p(x)$ and $r(x) \bmod x^{n/s}$.

Output: The coefficients of $r(x) \bmod x^{n/\log s}$.

Initialize: Set $k = n/s$, $j = \log s - 2\log\log s$, $q = n/\log s$, $y_0(x) = r(x) \bmod x^k$.

Computation:

1. $j+1$ times apply DFT on q points, to evaluate the polynomials $y_0(x)$ of (3.1) and $p_i(x)$ of (3.2), for $i = 0, \dots, j-1$, on the qth roots of 1.
2. Use (3.3), for $i = 0, 1, \dots, j-1$, in order to evaluate $y_{i+1}(x)$, for $i = 0, 1, \dots, j-1$, on the qth roots of 1.
3. Apply the inverse DFT on q points in order to evaluate the coefficients of $y_j(x)$. [$\deg y_j(x) \le (j+1)k2^j - 2^{j+1} + 1$; $y_j(x) = r(x) \bmod x^{k2^j}$, due to (3.4), and $k2^j = n/\log^2 s$.]
4. Update the initialized parameters by first setting $y_0(x) = y_j(x) \bmod x^{k2^j}$, $q = n(1 + \log\log s)/\log s$, and then $k = n/\log^2 s$, $j = \log\log s$; then redefine $p_i(x)$ and $y_{i+1}(x)$, for $i = 0, 1, \dots, j-1$, by using the equations (3.2) and (3.3); then again apply steps 1–3 to compute and to output the coefficients of the polynomial $y_j(x)$ (which equals $r(x) \bmod x^{k2^j}$, where $k2^j = n/\log s$ for the updated k and j).

The cost of the computation by the basis algorithm is clearly bounded by $O_A(\log n, n)$, and we will recursively apply this algorithm for a fixed n and for s successively taking the values

$$s_0 = n, \qquad s_1 = \log n, \qquad s_2 = \log\log n, \ \dots, \ s_{h-2} = \log^{(h-2)} n, \tag{3.6}$$

for $2 \le h \le \log^* n$ (for simplicity, let $\log^{(g)} n$ be integers for $g = 1, \dots, h$). This way, we will compute the first $m(h,n) = n/\log^{(h-1)} n$ coefficients of $r(x)$ at the cost $O_A((h-1)\log n, n)$. Such a policy of recursive restarting enables us to perform the computation fast, even by using fewer processors, which cannot be achieved with a single application of the algorithm of [G].

4. Final refinement of the algorithm. Let $N = n2^h$. In particular, this choice of N ensures that the above cost bound $O_A((h-1)\log n, n)$ is not greater than the desired bound (1.2). We still need, however, to compute the $N - m(h,n)$ coefficients of $r(x) \bmod x^N$ that remain unknown after the recursive application of the basis algorithm. We start with computing $r(x) \bmod x^n$ [note that $m(h,n) < n$ for $h \le \log^* n$]. We set

$$y_0(x) = r(x) \bmod x^{m(h,n)}, \quad k = m(h,n), \qquad j = \log^{(h-2)} n, \tag{4.1}$$

$$q = (j+1)k2^j - 2^{j+1} + 1, \tag{4.2}$$

[so that $k2^j = n$, $q \le (1 + \log^{(h-1)} n)n$], and apply steps 1–3 of the computation of the basis algorithm. The complexity of these steps is bounded from above by

$O_A(\log n, n(\log^{(h-1)} n)^2)$, which we may replace by the weaker bound $O_A(h\log n,$ $(n/h)(\log^{(h-1)} n)^2)$. The overall cost of computing the coefficients of $r(x) \bmod x^n$ is thus bounded by

$$O_A\left(h\log n,\ n\left(1+\frac{(\log^{(h-1)} n)^2}{h}\right)\right).$$

For $N = n2^h$ and for $h = \log^* n - O(1)$, in particular, for $h > \log^* n - 2$, we have $\log^{(h-1)} n = O(1)$, and the latter cost estimate implies the bound

$$c^*_{N,h} = O_A\left(h\log N,\ N2^{-h}\left(1+\frac{\log^{(h)} N}{h}\right)\right), \tag{4.3}$$

not exceeding the desired bound (1.2).

If $h \le \log^* n - 2$, then we will proceed similarly, but will extend the computation of $r(x) \bmod x^{n/s_i}$ for $s_0, s_1, \ldots$ of (3.6) to $s_{h-1} = \log^{(h-1)} n$ and $s_h = \log^{(h)} n$. This way we will compute all the coefficients of $r(x) \bmod x^n$ at the cost bounded by $O_A(h\log n,\ n(1+$ $(\log^{(h+1)} n)^2)/h)$. We replace this bound by the weaker bound

$$O_A\left(h\log n, n\left(1+\frac{\log^{(h)} n}{h}\right)\right)$$

on the overall cost of computing all the coefficients of $r(x) \bmod x^n$. We now substitute $N = n2^h$ and arrive at (4.3) also for $h \le \log^* n - 2$.

Finally, we make the transition from $r(x) \bmod x^n$ to $r(x) \bmod x^N$. We apply h steps of the Sieveking–Kung algorithm to compute the $N-n$ coefficients of $r(x) \bmod x^N$ that have not yet been computed; the cost of this stage, $O_A(h\log N,\ N/h)$ [see (2.4)], is smaller than the bound (1.2), at which we now arrive for $h \le \log^* n$. Surely, $\log^* N \le 1 + \log^* n$, and $\log^* N \to \infty$ as $N \to \infty$, so that we extend (1.2) to all $h \le \log^* N$ and then also deduce (1.3).

To relax the assumption about the integrality of $\log\log s$ and $\log^{(g)} n$ for $g = 1, \ldots, h$, it suffices: to replace the values of k in the basis algorithm (chosen at the initialization step and at step 4) and in (4.1), as well as the values of n/s_i [for $i = 1, \ldots, h$ and for $s_1, s_2, \ldots, s_h$ of (3.6)] and $m(h,n)$, by their ceilings, $\lceil\ \rceil$, that is, by the minimum integers not exceeded by these values; to define q by (4.2), and to let j denote the minimum integer such that $k2^j$ is not less than the number of the coefficients or $r(x)$ that should be computed at the current stage, that is, $\lceil n/\log^2 n\rceil$ in step 3 of the basis algorithm, $\lceil n/\log s\rceil$ in its step 4, and n in the final refinement stage. It is easy to verify that such a replacement preserves (4.3).

5. A modification of the main algorithm. Let us show a simple extension of our computation that leads to a simple extension of (4.3) [and, therefore, to (1.2) and (1.3)]. Denote

$$\log^{(2,0)} n = n,\ \log^{(2,a)} n = \log\left(\left(\log^{(2,a-1)} n\right)^2\right) = 2\log(\log^{(2,a-1)} n),$$
$$a = 1, 2, \ldots, \log^{(2,*)} n,$$

$$\log^{(2,*)} n = \max\{a : \log^{(2,a)} n > 0\}.$$

Fix G, $2 \le G \le \log^{(2,*)} n$, and recursively apply steps 1–3 of the basis algorithm, for $g = 0, 1, \dots, G^*$ (where either $G^* = G$ if $G \le (\log^{(2,*)} n) - 2$ or $G^* = G - 2$ otherwise), replace, at the gth call for these steps, the input values s, k, q, and j by $s(g)$, $k(g)$, $q(g)$, and $j(g)$, respectively, where we set

$$s(g) = \log^{(2,g)} n, \; k(g) = n/s(g), \; q(g) = (j(g)+1)\; k(g)\, 2^{j(g)} - 2^{j(g)+1} + 1,$$

$j(g) = \log(s(g)/s(g+1))$. (Assume for simplicity that these values, as well as s, k, q, and g below are integers.) Then again apply steps 1–3 of the basis algorithm for

$$s = \log^{(2,G-1)} n, \quad k = n/s, \quad q = (j+1)k\,2^j - 2^{j+1} + 1\,, \quad j = \log s,$$

set $N = n\,2^G$, and finally, conclude the evaluation of $r(x) \bmod x^N$ by following the line of § 4 and, in particular, performing G steps of the Sieveking–Kung algorithm. We immediately extend all the results of §§ 3 and 4 and the estimates of (1.2), (1.3), replacing $\log^{(h)}$ by $\log^{(2,h)}$ (for all h) and $\log^*$ by $\log^{(2,*)}$.

6. Application over abstract rings. Our algorithms can be applied over any ring $\mathbf{R}$ of constants that contains unity and an element g such that $g^{-1} \in \mathbf{R}$ and $1, g, g^2, \dots, g^{N-1}$ are distinct in $\mathbf{R}$. In this case we may replace FFT by its Bluestein extension (where g replaces the principal root of 1) [Kn]. Such an extended FFT is performed at the cost $O_A(\log N, N \log\log N)$ over any ring $\mathbf{R}$ [CK].

Furthermore, there exists a desired element g in $\mathbf{R}$ if the number $|\mathbf{R}|$ of distinct elements that belong to $\mathbf{R}$ together with their reciprocals exceeds $N-1$. Otherwise, there must exist such an element g in the extension $\mathbf{E}$ formed by all polynomials over $\mathbf{R}$ modulo any irreducible polynomial of degree d if $d \ge \log N / \log |\mathbf{R}|$. In the latter case, each univariate polynomial of degree at most K in $\mathbf{E}$ turns into a bivariate polynomial in $\mathbf{R}$ of degrees at most K and $d-1$ in its two variables, respectively. Multiplication of two such polynomials has computational cost $O_A(\log(Kd), Kd \log\log(Kd))$ [CK]. This implies that the transition from application of our algorithms in $\mathbf{E}$ (where they are supported by Bluestein's extension of the FFT) to their application in the original ring $\mathbf{R}$ will require increasing the resulting processor bound by the factor of d, and we may assume that $d = O(\log N / \log |\mathbf{R}|)$.

As an alternative, we may compute $p^{-1}(x) \bmod x^{\lceil N/d \rceil}$ by using our algorithms and then make the transition to $p^{-1}(x) \bmod x^N$ by means of the Sieveking–Kung algorithm at the overall cost $O_A(\log N \log\log N, N/\log\log N)$ over any ring of constants.

The same comments apply to our algorithm for the evaluation modulo x^N of $p^{1/2}(x)$, to be presented next.

Appendix. Extensions. We will demonstrate two ways of extending the results of this paper.

1. First we will combine the techniques of the previous sections to improve the parallel computations of a polynomial $v(x) \bmod x^N$ by means of a recursive process of the form

$$v_{i+1}(x) = h(v_i(x), x) \bmod x^N, \tag{A.1}$$

where $h(v, x)$ is a polynomial in v and x; $\deg_v h$ is small, and $v_i(x) = v(x) \bmod x^{d_i}$ for d_i that rapidly grows as i grows. We will repeat the patterns of §§2–4, according to the following procedure:

(a) Contract the stream of the recursive steps of the original (slower but processor-efficient) algorithm to arrive at a faster, although processor-inefficient, algorithm;

(b) Recursively apply this resulting algorithm (with stopping and restarting to achieve processor efficiency); and

(c) Apply the original (slower but processor efficient) algorithm at the final refinement stage.

For demonstration purposes, we will choose the problem of computing modulo x^N the square root, $v(x) = p(x)^{1/2} \bmod x^N$, of a polynomial $p(x)$, where $p(0) = 1$. Since

$$f(v, x) = v^{-2}(x)\, p(x) - 1 = 0,$$

we apply Newton's iteration with $v_0(x) = 1$ and recursively compute

$$\text{(A.2)} \qquad \begin{aligned} v_{i+1}(x) &= v_i(x) - f(v_i, x) \,/\, f'_v(v_i, x) \\ &= 0.5\, v_i(x)(3 - p^{-1}(x) v_i^2(x)\,), \quad i = 0, 1, \ldots. \end{aligned}$$

We observe that

$$\begin{aligned} v_{i+1}(x) - p^{1/2}(x) &= 0.5\, v_i(x)\left(3 - p^{-1}(x) v_i^2(x)\,\right) - p^{1/2}(x) \\ &= v_i(x) - p^{1/2}(x) + 0.5 v_i(x) p^{-1}(x)\big(p(x) - v_i^2(x)\big) \\ &= \big(v_i(x) - p^{1/2}(x)\big)\left(1 - 0.5 v_i(x)\, p^{-1}(x)\big(p^{1/2}(x) + v_i(x)\big)\,\right) \\ &= \big(v_i(x) - p^{1/2}(x)\big)\left[\left(1 - v_i(x)\, p^{-1/2}(x)\,\right)\right. \\ &\quad \left. -0.5 v_i(x)\, p^{-1}(x)\big(v_i(x) - p^{1/2}(x)\big)\right] \\ &= -\big(v_i(x) - p^{1/2}(x)\big)^2 \left(p^{-1/2}(x) + 0.5 v_i(x)\, p^{-1}(x)\right) \\ &= O\left(\big(v_i(x) - p^{1/2}(x)\big)^2\right), \end{aligned}$$

so that

$$v_j(x) - p^{1/2}(x) = 0 \bmod x^J, \quad J = 2^j, \quad j = 0, 1, \ldots.$$

As in the Sieveking–Kung algorithm, substitute $v_j(x) \bmod x^J$, $J = 2^j$, for $v_j(x)$, $j = i, i+1$, in (A.2), substitute $p^{-1}(x) \bmod x^{2^i}$ for $p^{-1}(x)$ in (A.2), and arrive at the complexity bound $O_A(\log^2 N,\ N/\log N)$ for this computational problem.

As was the case in the algorithm of [G], replace $p^{-1}(x)$ by $p_i^{-1}(x) = p^{-1}(x) \bmod x^{2^{i+1}}$ in (A.2). Observe that in this case the degrees of the polynomials $v_i(x)$ are bounded from above by $2\sum_{g=1}^{i}(2/3)^g 3^i < 3^{i+2}$, and deduce the bound $O_A(\log k,\ k^d \log k)$, $d = \log 3 = 1.5849\ldots$, $1/d = 0.6309\ldots$, on the complexity of the evaluation of $p^{1/2}(x) \bmod x^k$.

We apply the latter results for $k = n^{0.63}$, where we will specify $n \le N$ later on [see (A.7)]. Then $k^d \log k = o(n^{0.999})$, and we obtain the k first coefficients of $p^{1/2}(x)$ at the cost of $O_A(\log n, n^{0.999})$.

Next, extend the basis algorithm of § 3, by using the equations

$$p_i^{-1}(x) = p^{-1}(x) \bmod x^{k\,2^{i+1}},$$

$$y_0(x) = v(x) \bmod x^k, \quad y_{i+1}(x) = 0.5\, y_i(x)\big(3 - p_i^{-1}(x)\, y_i^2(x)\big), \quad i = 0, 1, \ldots, j,$$

instead of (3.1)–(3.3). Observe that the degrees of the polynomials $y_i(x)$ are bounded from above by $3^i\left(k + 2\sum_{g=0}^{i}(2/3)^g\right) - \sum_{g=0}^{i} 3^g < 3^i(k+3) + 0.5$, so that the transition from $y_0(x) = p^{1/2}(x) \bmod x^k$ to $y_i(x) = p^{1/2}(x) \bmod x^{k2^i}$ costs

$$\text{(A.3)} \qquad O_A(\log n, k2^{id}(i + \log k)\,) = O_A(\log n,\ k(m/k)^d \log m),$$

for $d = \log 3$, $m = m(i, k) = k2^i \le n$.

Let us now choose an integer g and an increasing sequence of integers $k_0 = 1$, $k_1 = \lceil n^{0.63} \rceil$, $k_2, \dots, k_g$, $k_j = \lceil n^{1-0.37^j} \rceil$, $j = 0, 1, \dots, g$, and recursively apply the above transition from $y_0(x)$ to $y_i(x)$, for $k = k_j$, $m = k2^j = k_{j+1}$, $j = 0, 1, \dots, g-1$. Due to (A.3), the cost of this transition for any j is bounded by

$$O_A(\log n,\ n \log n), \tag{A.4}$$

since $\log m < \log n$ and since $k(m/k)^d = k_j(k_{j+1}/k_j)^d = n^{1-0.37^j(1-0.63d)} < n$. Denote $b_j = \log k_j / \log n \geq 1 - 0.37^j$, so that $1 - b_g \leq 1/\log n$ for

$$g = \lceil \log \log n / \log(1/0.37) \rceil. \tag{A.5}$$

In g recursive transitions we will arrive at $p^{1/2}(x) \bmod x^{k_g}$, where

$$k_g \geq n^{1-1/\log n} = n/2.$$

Due to (A.4) and (A.5), the overall cost of this computation is bounded by

$$O_A(\log n \log \log n, n \log n). \tag{A.6}$$

Now, let

$$n = \lceil N/(\log N \log \log N) \rceil, \tag{A.7}$$

apply $1 + \lceil \log(\log n \log \log n) \rceil$ Newton's steps of the Sieveking–Kung type and arrive at $p^{1/2}(x) \bmod x^N$ at the additional cost

$$O_A(\log N \log \log N,\ N/\log \log N), \tag{A.8}$$

where we may assume that $p^{1/2}(x) \bmod x^{k_g}$ is known for $k_g \geq n/2$. Equation (A.8) also bounds the overall cost of computing $p^{1/2}(x) \bmod x^N$ since (A.6) turns into (A.8) for n of (A.7).

On the other hand, for any fixed $\epsilon > 0$ and for $N = \lceil n^{1-0.37^j} \rceil$, we may choose j such that

$$n \log n < N^{1+\epsilon}.$$

Recall (A.4), and obtain $p(x)^{1/2} \bmod x^N$ at the cost of

$$O_A(\log N,\ N^{1+\epsilon}). \tag{A.9}$$

The complexity estimates (A.8) and (A.9) for computing the square root of a polynomial modulo x^N can be immediately extended to computing its mth root modulo N for any fixed positive integer m. Moreover, the techniques applied above enable us to extend (A.8) and (A.9) as follows.

THEOREM A.1. *Let P denote a given problem of computing a string $s(N)$ of N scalars. Suppose that for $c > 1$ and for a pair of integers k and m, $1 \leq k < m$, two alternate "black box" parallel algorithms,* A.1 *and* A.2, *are available; on the given substring of the k first scalars of $s(N)$ they compute its next $m - k$ scalars, at the cost bounded by* $O_A(\log m \log(m/k), m/\log(m/k))$ *and* $O_A(\log m,\ k(m/k)^c)$, *respectively. Then the recursive application of Algorithm* A.2, *for some appropriate recursive choice of the pairs $k = k_i, m = m_i, i = 0, 1, \dots,$ such that $k_{i+1} = m_i$, followed by the application of Algorithm* A.1, *supports the solution of the problem P of size N, at the cost bounded by* (A.8) *and* (A.9).

For a more narrow class of computational problems, exemplified by polynomial division, we may improve (A.8) and (A.9) to (1.2).

Somewhat similar techniques yield supereffective slowdown of some fundamental computations in linear algebra and in path algebras, where we recursively apply and restart fast but processor-inefficient algorithms to subproblems of a smaller size in order to decrease the size of the original problem (see [PP]).

Remark A.1. It is essential for the approach of this paper to have a polynomial $h(v, x)$ (rather than a rational function) in (A.1); for instance, we cannot combine this approach with Newton's iteration

$$w_0(x) = 1, \quad w_{i+1}(x) = 0.5\big(w_i(x) - 1/(w_i(x)\,p(x))\big), \quad i = 0, 1, \ldots$$

for the equation $w^2(x)\,p(x) - 1 = 0$.

2. The complexity estimates (1.2), (1.3), (A.8), and (A.9) can be extended to the computational problems readily reducible to computing the reciprocal of a polynomial $p(x)$ and its mth root, respectively. For demonstration, let us recall the well-known reduction of polynomial division with a remainder to computing the reciprocal of a polynomial (and to polynomial multiplication).

Indeed, given the coefficients of two polynomials $s(x) = \sum_{i=0}^{m} s_i x^i$ and $t(x) = \sum_{i=0}^{n} t_i x^i$, $t_n \neq 0$, we seek the quotient $q(x) = \sum_{i=0}^{m-n} q_i x^i$ and the remainder $r(x) = \sum_{i=0}^{n-1} r_i x^i$ such that

$$s(x) = t(x)\,q(x) + r(x). \tag{A.9}$$

Having computed $q(x)$ we immediately obtain $r(x)$ from (A.9). To compute $q(x)$, we define

$$\begin{aligned} S(z) &= z^m s(1/z) = \sum_{i=0}^{m} s_{m-i} z^i, \\ T(z) &= z^n t(1/z) = \sum_{i=0}^{n} t_{n-i} z^i, \\ Q(z) &= z^{m-n} q(1/z) = \sum_{i=0}^{m-n} q_{m-n-i} z^i, \end{aligned}$$

observe that

$$S(z) = T(z)\,Q(z) \bmod z^{m-n+1}, \tag{A.10}$$

and apply the algorithms of this paper to compute the reciprocal,

$$V(z) = \sum_{i=0}^{K-1} v_i z^i = T^{-1}(z) \bmod z^K, \quad K = m - n + 1.$$

It immediately follows from (A.10) that $Q(z) = V(z)\,S(z) \bmod z^K$, so that the coefficients of $q(x)$ are the trailing coefficients of the polynomial product $V(z) \sum_{i=0}^{m-n} s_{m-i} z^i$.

Acknowledgments. The authors thank the referees for several helpful comments and suggestions and also C. Engle for her excellent assistance with typing this paper.

REFERENCES

[B] D. BINI, *Parallel solution of certain Toeplitz linear systems*, SIAM J. Comput., 13 (1984), pp. 268–276.

[BP] D. BINI AND V. PAN, *Polynomial division and its computational complexity*, J.Complexity, 2 (1986), pp. 179–203.

[BPa] ———, *Numerical and Algebraic Computations with Matrices and Polynomials*, Vols. 1 and 2, Birkhäuser, Boston, 1993.

[CK] G. CANTOR AND E. KALTOFEN, *On fast multiplication of polynomials over arbitrary algebras*, Acta Inform., 28 (1991), pp. 693–701.

[G] R. E. GEORGIEV, *Inversion of triangular Toeplitz matrices by using the fast Fourier transform*, J. New Generation Comput. Systems, 2 (1989), pp. 247–256.

[KR] R. M. KARP AND V. RAMACHANDRAN, *A survey of parallel algorithms for shared memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[Kn] D. E. KNUTH, *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, Addison-Wesley, Reading, MA, 1981.

[PP] V.Y. PAN AND F. P. PREPARATA, *Supereffective slow-down of parallel computations*, Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 402–409.

[PR] V. Y. PAN AND J. H. REIF, *The parallel computation of the minimum cost paths in graphs by stream contraction*, Inform. Process. Lett., 40 (1991), pp. 79–83.

[R] J. H. REIF, *Logarithmic depth circuits for the algebraic problems*, SIAM J. Comput., 15 (1986), pp. 231–242.

[RT] J. H. REIF AND S. R. TATE, *Optimal size division circuits*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1989, pp. 264–270; SIAM J. Comput., 19 (1990), pp. 264–270.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 627–649, June 1993

THE COMPLEXITY OF PROCESSING HIERARCHICAL SPECIFICATIONS*

DANIEL J. ROSENKRANTZ†‡ AND HARRY B. HUNT III †§

Abstract. Hierarchical object descriptions consisting of a set of module descriptions are considered, where each module is either a primitive module or has a body that is an interconnection of submodules. The description represents a flattened object, whose size can be exponential in the size of the description. The complexity of processing and/or analyzing such hierarchically specified objects is considered. The simulation of hierarchically specified circuits is emphasized, but the results are applicable to other kinds of hierarchically specified objects.

It is shown that hierarchically specified acyclic circuits can be simulated deterministically in space linear in the size of the description, even when the description is not explicitly acyclic. $\Theta(n^2)$-size-bounded reductions are given from the languages in DSPACE(n) to the problem of simulating hierarchically specified acyclic monotone circuits. This implies that this simulation problem is PSPACE-complete and that any algorithm for it that operates faster than $2^{O(\sqrt{n})}$ deterministic time could be used to recognize all DSPACE(n) languages in less than $2^{O(n)}$ deterministic time. It is then shown that the simulation problem for hierarchically specified acyclic circuits (not necessarily monotone) can indeed be solved in $2^{O(\sqrt{n})}$ deterministic time. Moreover, every hierarchically specified acyclic circuit is shown to have an equivalent flat circuit of size $2^{O(\sqrt{n})}$. For binary circuits the size of the equivalent flat circuit is $O\left(n^{3/2}2^{1.53\sqrt{n}}\right)$. It is also shown that the problem of simulating hierarchically specified circuits is EXPSPACE-complete for cyclic circuits.

Key words. hierarchical specification, CAD systems, acyclic circuits, combinational circuits, cyclic circuits, sequential circuits, simulation, computational complexity

AMS(MOS) subject classifications. 68Q25, 94C10, 68M15

1. Introduction.

Hierarchical object design permits the overall design of an object to be partitioned into the design of a collection of modules, each of whose design is a more manageable task than producing a complete design in one step. It also facilitates the development of computer-aided design (CAD) systems since low-level objects can be incorporated into libraries and thus can be made available as submodules to designers of larger-scale objects. In particular, hardware description languages usually permit circuits to be hierarchically specified (see, e.g., [Cr], [LMS], [MC], [Ni]). Hierarchical description languages are also useful for describing the configuration of distributed software systems [MKS].

An important feature of hierarchical specifications is that they permit more concise descriptions of circuits than do flat nonhierarchical descriptions. A consequence of this is that the complexity of analyzing or otherwise processing a given hierarchically presented object can be different from that when the object is presented as a flat combination of primitives. For example, in VLSI mask specifications, analyzing a flat list of rectangles for overlap can be done in polynomial time, whereas this problem is NP-complete for hierarchically specified sets of rectangles [BO]. On the other hand, when restrictions are imposed on the hierarchical mask specifications, they can be processed more efficiently [BO], [Oea], [SS], [Wa]. Certain graph-analysis problems for hierarchically specified undirected graphs can be solved efficiently in polynomial time [Len2], [Len3], [LWan]. The effect of hierarchical specifications on VLSI design problems is investigated in [Len1], and their effect on more general combinatorial problems is investigated in [LWag]. In [Len1] and [LWag] it is shown that the problem of simulating

*Received by the editors November 13, 1989; accepted for publication (in revised form) February 18, 1992.
†Department of Computer Science, State University of New York at Albany, Albany, New York 12222.
‡This research was supported by National Science Foundation grants DCR 86-03184 and CCR 88-03278.
§This research was supported by National Science Foundation grants DCR 86-03184 and CCR 89-03319.

a hierarchically specified Boolean circuit, where the specification is explicitly acyclic at every level of the hierarchy, is a PSPACE-complete problem, where [Len1] covers the upper bound and [LWag] covers the lower bound. In [Len1] it is also shown that an explicitly acyclic Boolean circuit can be simulated within space that is linear in the size of the hierarchical description and a nonexplicitly acyclic circuit in quadratic space. The question of whether a nonexplicitly acyclic circuit can be simulated in linear space is left open, and it is conjectured that such an efficient deterministic space simulation is unlikely.

Here we study the complexity of processing and/or analyzing hierarchically specified objects, emphasizing the simulation of hierarchically specified circuits. A hierarchically specified object is presented as a set of modules, each of which is classified as either a primitive module or a composite module. Each composite module has a body, whose description consists of an interconnection of instances of lower-level modules. Each module has a corresponding flattened body, which is an interconnection of primitive modules that can be obtained by repeatedly replacing each instance of a composite module with its body. We are concerned with analyzing a hierarchically specified object for some given property, where the property is a property of the flattened object. An issue is, for which properties can the analysis be performed in a more efficient way than constructing the flattened object and then analyzing it? We focus on the problem of circuit simulation, which reflects the semantics of what circuits actually do.

In §2 we present definitions and terminology for hierarchically specified objects.

In §3 we show that hierarchically specified acyclic circuits can be simulated in deterministic space linear in the size of the description, even when the description is not explicitly acyclic at every level. This answers the open problem from [Len1] that was mentioned above.

In §4 we generalize and strengthen the PSPACE-hard lower bound in [LWag] for explicitly acyclic logic-circuit simulation to explicitly acyclic monotone logic-circuit simulation. We accomplish this in a very general algebraic setting by means of $\Theta(n^2)$-size bounded reductions from the languages in DSPACE(n). As a consequence, if there are languages in DSPACE(n) whose recognitions requires $2^{\Omega(n)}$ deterministic time, then this simulation problem requires $2^{\Omega(\sqrt{n})}$ deterministic time. In contrast, the reduction in [LWag] is a $\Theta(n^4)$-size bounded reduction from the languages in DSPACE(n); so even if there is such a language requiring $2^{\Omega(n)}$ time, the reduction only implies that the acyclic circuit simulation problem requires $2^{\Omega(n^{1/4})}$ deterministic time.

In §5 we show that hierarchically specified acyclic circuits can indeed by simulated in deterministic $2^{O(\sqrt{n})}$ time. This contrasts with the traditional approach of first flattening the hierarchically specified circuit and then simulating this flattened circuit [LM], [BS]. In the worst case the size of the flattened circuit is $2^{\Omega(n)}$ and thus the overall time required is $2^{\Omega(n)}$. However, we show that every hierarchically specified acyclic circuit has an *equivalent* flat circuit of size $2^{O(\sqrt{n})}$. Furthermore, this equivalent circuit can be constructed and thus simulated in time $2^{O(\sqrt{n})}$. Results on the size of acyclic circuit descriptions also appear in [KS], where the focus is on the effect of requiring that a description be explicitly acyclic at every level of the hierarchy. In [KS] it is shown that there are acyclic circuits that have a hierarchical description of size n such that any description that is explicitly acyclic at every level of the hierarchy must be of size $2^{\Omega(n)}$, and it is also shown that for every acyclic circuit that has a hierarchical description of size n there is an equivalent circuit that has a $O(n^3)$-size hierarchical description that is explicitly acyclic at every level of the hierarchy.

Sections 4 and 5 present lower and upper complexity bounds that match, given the current state of knowledge about computational complexity. A $\Theta(n^p)$-size-bounded reduction from the languages in DSPACE(n) to the simulation problem for acyclic circuits and an $2^{O(n^q)}$ algorithm for this simulation problem can be combined, implying that all languages in DSPACE(n) are recognizable in time $2^{O(n^{pq})}$. Thus an improvement either to the size bound of $O(n^2)$ in the reduction from §4 or to the time bound of $2^{O(\sqrt{n})}$ for the simulation algorithm from §5 would imply that all languages in DSPACE(n) are recognizable in less than $2^{\Theta(n)}$ deterministic time. This would be a surprising breakthrough in complexity theory.

In §6 we consider cyclic hierarchically specified logic circuits. Under fairly loose assumptions, the simulation problem for such logic circuits is shown to be EXPSPACE-complete by means of $\Theta(n)$-size-bounded reductions from the languages accepted by 2^n-space-bounded Turing machines. This suggests that to solve this type of problem, one might as well construct and then simulate the flattened circuit.

We assume that the reader is familiar with complexity classes and reductions; otherwise, see, e.g., [AHU], [GJ].

2. Definitions and terminology of hierarchical specifications.

We define the class of hierarchically specified objects as follows. An object is described as a set of *modules*, where each module is either a *primitive module* or a *composite module*. Each module (whether primitive or composite) has an *interface*, and each composite module has a *body*. A module interface specifies the *module name* and a set of *module ports*. Each module port has a *port name* and is specified to be either an *input port* or an *output port*. For a given module, the module port names are unique. A module body consists of an interconnection of *module instances*. A given module instance can be an instance of either a primitive module or a composite module. A given module body may contain more than one instance of the same module.

As an example, Fig. 1(a) shows an interface for a module named P with input ports $\{U, V, W\}$ and output ports $\{X, Y\}$. In the graphical representation used in our diagrams, input ports are denoted by incoming arrows and output ports are denoted by exiting arrows. Figure 1(b) shows a body for module P. The module body contains two instances of module B, an instance of module A, and an instance of module C. Each module instance in a body as an *instance name*. In Fig. 1(b) the instance names are $B1$, $B2$, A, and $CZAP$. For a given module body the instance names are unique.

A module body contains two kinds of ports: *instance ports* and *body ports*. An *instance port* is a port of an instance within the body. Examples of instance ports in Fig. 1(b) are port J of instance $CZAP$ and port Z of instance $B2$. A *body port* is a port of the module whose composition is described by the body. In Fig. 1(b), P has five body ports: U, V, W, X, and Y.

The interconnections within a module body are specified by *signals*. Each signal is given a *signal name* and is *connected* to a set of ports within the body. For example, in Fig. 1(b) signal $B2Z$ is connected to instance ports Z of $B2$, F of $B1$, and K of $CZAP$. Signal U is connected to body port U of P and instance ports I of A and D of $B1$. Note that Fig. 1(b) uses the convention of giving a signal connected to exactly one body port the same name as that body port. For a given module body the signal names are unique.

We assume that a set of modules satisfies certain restrictions. First, define the *DIRECTLY-WITHIN* relation on modules as follows: module α is *DIRECTLY-WITHIN* module β if the body of β contains an instance of α. We assume that the set of modules satisfies the *nesting restriction* that the *DIRECTLY-WITHIN* relation is acyclic. The nesting restriction ensures that the description of a set of modules is a meaningful hierarchical description of a finite object.

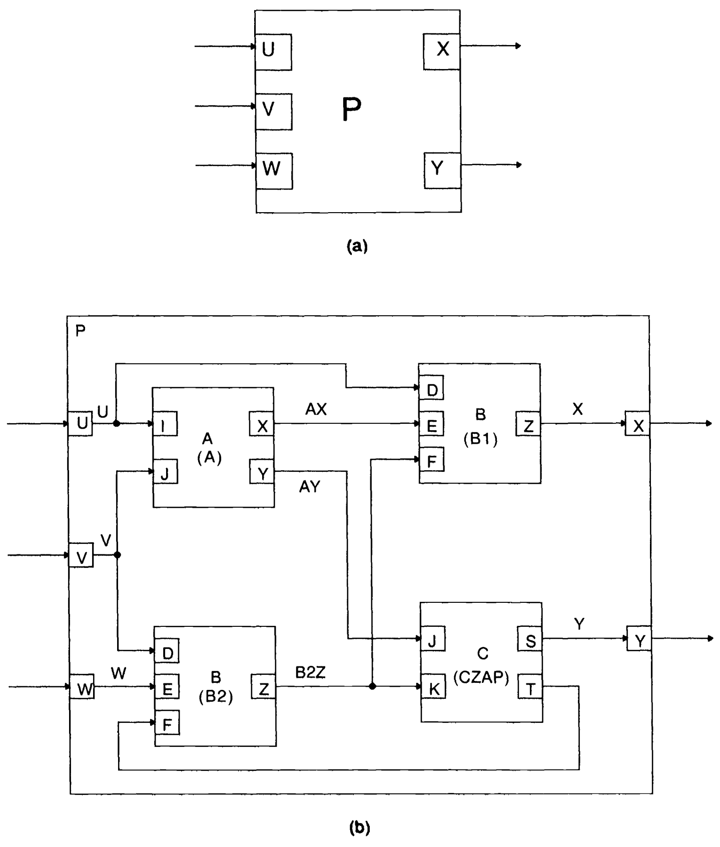


FIG. 1. (a) *Interface for module* P; (b) *body for module* P.

In a module body define a *driver port* to be either a body input port or an instance output port. Define a *drivable port* to be either a body output port or an instance input port. We assume that each module body satisfies the *wiring restriction* that no signal in the body is connected to two driver ports.

The wiring restriction prevents direct connections between drivers. However, real circuit technologies often permit drivers to be connected together, with a technology-dependent rule resolving the situation when connected drivers supply conflicting values. To model a circuit design that uses this capability, we would require the placement of a module instance having inputs connected to these drivers and having a single output. The semantics of this additional module would represent the conflict resolution function. For example, Fig. 2(a) shows a module body containing two violations of the wiring restriction. Figure 2(b) shows a corresponding body that satisfies the wiring restriction, with instances of module *RES* inserted to resolve conflicts.

For a given module in a hierarchically specified object, the *flattened body* for that module is the interconnection of instances of primitive modules, obtained by repeatedly replacing instances of composite modules in the body by their bodies.

A *module description* consists of a module interface, together with either an indication that the module is primitive or a body for the module. A *hierarchical object description* consists of a set of module descriptions, one of which is designated as the *root mod-*

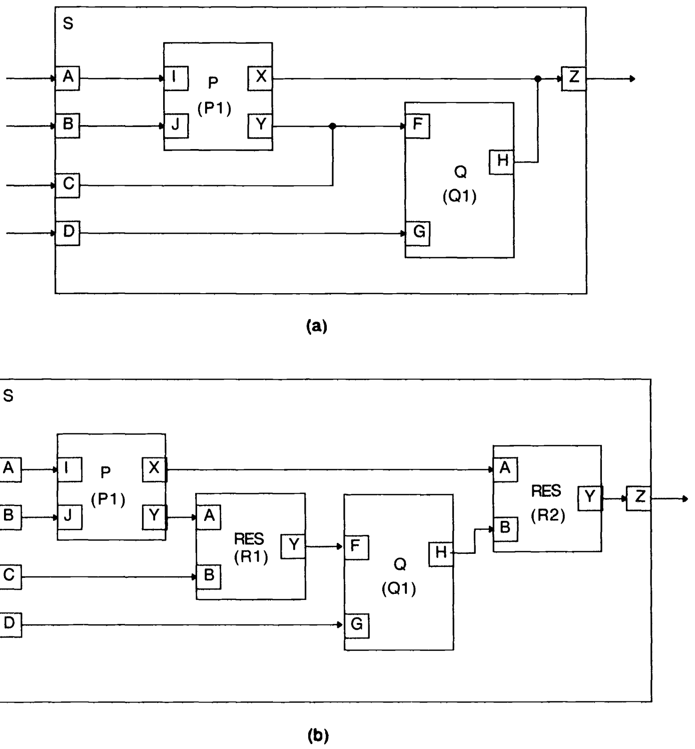


FIG. 2. (a) *Body with violations of wiring restriction*; (b) *body with extra instances to satisfy wiring restriction.*

ule, such that the set includes every module that occurs as a module instance within any of the module bodies. The root module is said to be *hierarchically specified*. The *hierarchically specified object* corresponding to a hierarchical object description is the flattened body of the root module. A *flat object description* is a hierarchical object description involving a single composite module. Thus the body of this module is an interconnection of instances of primitive modules.

The *size* of a module interface is 1 plus the number of module ports. The *size* of a module body is the sum of the size of the interface of the module, the number of signals in the body, and the sizes of the interface for each module instance within the body. Note that this is the same as the sum of 1, the number of signals within the body, the number of module instances within the body, and the number of ports (both instance ports and body ports) within the body. The *size* of a module description is the sum of the sizes of its interface and body. The *size* of a hierarchical object description is the sum of the sizes of the module descriptions occurring in the overall object description. This definition of size measures the number of symbols occurring in a typical representation of the object description. The number of bits in the representation would be larger since each symbol might be represented by a number of bits equal to the logarithm of the number of distinct symbols. We regard the number of symbol occurrences in the description to be the more practical measure of its size.

We now consider acyclic hierarchically described objects. An *instance dataflow graph* for a given composite module is a directed graph describing the dataflow within the body of the module. The graph has a vertex for each module instance within the body and a vertex for each body port. There are edges corresponding to connections from driver ports to drivable ports within the body. Specifically, there is a directed edge for each of the following four cases:

(a) Edge from instance A to instance B if an output port of A is connected to an input port of B.

(b) Edge from body input port X to instance B if X to connected an input port of B.

(c) Edge from instance A to body output port Y if an output port of A is connected to Y.

(d) Edge from body input port X to body output port Y if X is connected to Y.

As an example, Fig. 3 shows the instance dataflow graph for module P described in Fig. 1.

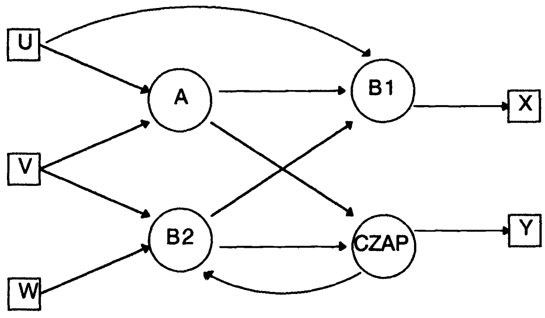


FIG. 3. *Instance dataflow graph.*

A composite module is said to be *locally acyclic* if its instance dataflow graph is acyclic. A composite module is said to be *weakly acyclic* if the instance dataflow graph for its flattened body is locally acyclic. A composite module M is said to be *strongly acyclic* if module M is locally acyclic and every composite module Q such that Q *DIRECTLY-WITHIN* M is strongly acyclic [Len1].

Strong acyclicity is a special case of weak acyclicity. Let the relation *SOMEWHERE-WITHIN* be the transitive closure of *DIRECTLY-WITHIN*. Then an equivalent definition of M being strongly acyclic is that M is locally acyclic and every composite module Q such that Q *SOMEWHERE-WITHIN* M is locally acyclic. Strong acyclicity requires not only that the flattened circuit be acyclic but also that at every level of the hierarchy each module body be described as an acyclic combination of the instances occurring directly within it. Given a hierarchical description, it is clear that strong acyclicity can be determined in polynomial time. In addition, weak acyclicity can be determined in polynomial time, using a bottom-up dataflow analysis [LWan].

3. Linear space simulation of weakly acyclic circuits. In this section we show that weakly acyclic hierarchically specified logic circuits can be simulated deterministically using only linear space. First, we define the simulation problem as follows: the *simulation problem* for a weakly acyclic module consists of computing the values of the output ports of the module, given a value for each input port of the module and a hierarchical circuit description for which it is the root module.

We make several assumptions to ensure that the simulation problem and the complexity analysis of simulation algorithms are well defined. First, we assume that the domain of signal values is finite. (Equivalently, the amount of space required to represent a signal value is fixed.) Second, we assume that each primitive module is a memoryless combinatorial element whose input values determine its output values. Third, we assume that the input–output behavior of each primitive module is polynomial-time computable, i.e., given input values, the output values can be computed in polynomial time. Finally, we assume that if the flattened body of the module being simulated contains a primitive module instance port that is unconnected to a driver, there is a rule for what value to use for that input port.

Reference [Len1] contains a sketch of how a strongly acyclic hierarchically specified circuit whose primitive modules are logic gates can be simulated deterministically in space that is linear in the size of the hierarchical circuit description. It is also pointed out that a weakly acyclic circuit can be simulated nondeterministically in linear space and deterministically in quadratic space. The problem of finding a deterministic linear space algorithm for this case is left as an open question, and Lengauer says that it is questionable whether such a method exists. The next result shows that weakly acyclic circuits can indeed be simulated deterministically in linear space.

THEOREM 3.1. *A weakly acyclic hierarchically specified object can be simulated in space linear in the size of the hierarchical circuit description.*

Proof. First, consider the case when the hierarchically specified object is strongly acyclic, as described in [Len1]. In this case, since each module is locally acyclic, its instance dataflow graph is acyclic. Each instance dataflow graph can be topologically sorted, and the instances within it can be simulated in this order. Consider a given module to be simulated, with given values for its input ports. If the given module is primitive, its simulation consists of computing the values of the output ports using the values of the input ports. If the given module is composite, then each of the instances in the body is simulated, in topological sort order. In any stage of simulating a given module body, the algorithm maintains a record of the input-port values of the body and of the output-port values of instances simulated thus far. Consequently, when a given instance in the body is to be simulated, all drivers of the input ports of the given instance have known values. When the last instance in the body of the given composite module has been simulated, the value of the drivers of the output ports of the given module are known and the simulation of the given module is complete.

Figure 4 shows an example in which A, B, and C are composite modules; left-to-right order in the figure corresponds to a topological sort of the instances within each body. Figure 4 shows a snapshot of the simulation of module A, where recorded values are indicated by darkening the appropriate driving ports. In the snapshot the simulation of instances $B1$ and $D1$ within A have been completed and the simulation of instance $B2$ within A is in progress. In this subsimulation, the simulation of $D1$ within module B has been completed and the simulation of $C1$ within B is in progress. In this subsimulation, the simulations of $D1$ and $E1$ have been completed and the simulation of $E2$ is about to begin.

Now consider a weakly acyclic hierarchically specified circuit. Consider the subsimulation problem of simulating a given composite module in the hierarchy, where a subset (not necessarily proper) of the input ports of the module have specified values. The subsimulation computes the values of all those output ports whose values are determined by the specified input-port values. More precisely, a given output-port value is computed if a value has been specified for every input port for which the instance dataflow graph for the flattened body of the module has a path to the given output port.

In the subsimulation of a given body we use two lists of module instances occurring in the body. We call these two lists *FULL* and *PARTIAL*. List *FULL* contains module instances for which all the instance input-port values are known and the module instance

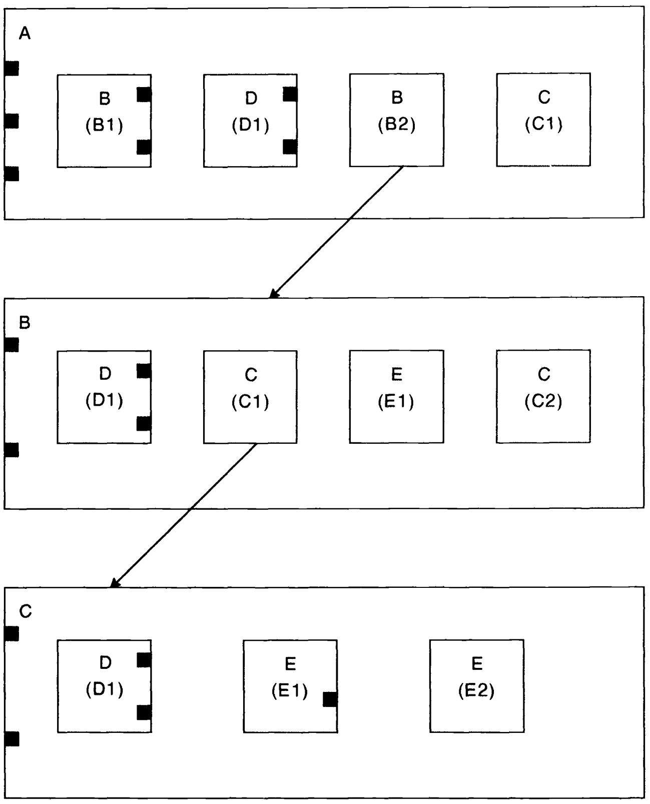


FIG. 4. *Snapshot of simulation of a strongly acyclic circuit.*

can be simulated to obtain all its instance output-port values. List *PARTIAL* contains module instances for which some (possibly none), but not all, of the instance input-port values are known and for which the module can be simulated to obtain those instance output-port values that do not depend on unknown input-port values.

Initially in the subsimulation some of the body input-port values are known. Each of these known driver values specifies the values of all drivable ports to which it is connected. Also, each unconnected instance input port is assigned a special driver value, say, UNCONNECTED, during the simulation. List *FULL* is initialized with those module instances for which all instance input-port values are known, based on the above information. List *PARTIAL* is initialized with all remaining module instances.

The subsimulation proceeds until lists *FULL* and *PARTIAL* are both empty. If both lists are nonempty, a module instance is deleted from *FULL* and is simulated. If *FULL* is empty and *PARTIAL* is nonempty, a module instance is deleted from *PARTIAL* and is simulated. When both lists are empty, the subsimulation returns all the output-port values for which the driving values have been computed during the subsimulation. As the subsimulation proceeds a record of known driver values is maintained, i.e., a record of the given body input-port values and the instance output-port values that have been obtained thus far.

Now consider what happens in a simulation upon completion of the subsimulation of a given module instance within the body being simulated. For each instance output-port value computed by the subsimulation such that this output-port value was not known prior to the subsimulation, this value is recorded. Furthermore, each module instance having an input port driven by this driver is checked. If the module instance does not yet have all its output values computed, then it is a candidate for subsimulation. If all its input-port values are now known, it is placed on list *FULL* (and removed from list *PARTIAL* if it is currently on that list); otherwise, it is placed on list *PARTIAL* if it is not already on that list.

An example is shown in Fig. 5, in which A, B, and C are composite modules. In this example, for both B and C the lower output port is determined solely by the lower input port and the upper output port is determined by both input ports. Figure 5 shows a snapshot of the simulation in which recorded values are indicated by darkening the appropriate driving ports. Within module A instance $B1$ was simulated with only its lower input port known, producing the value of its lower output port. The instance $B2$ was simulated with its lower input port known, producing the value of its lower output port. Then instance $C1$ was simulated with its lower input port known, producing the value of its lower output port. Then instance $B1$ was simulated again, this time with both input ports known, and the value of its upper output port was obtained. Then instance D was simulated. The simulation of instance $B2$ with both input ports known is in progress. In this subsimulation of B, $C1$ has already been simulated with its lower input port known, producing the value of its lower output port. The simulation of $C2$ with its lower input port known is in progress. In this subsimulation of C the simulation of instance $E1$ has been completed and the simulation of $E2$ is about to begin.

The data recorded at any point in the overall algorithm is a set of values for drivers within the bodies of modules occurring in a directed path of the *DIRECTLY-WITHIN* relation. Since the *DIRECTLY-WITHIN* relation is acyclic, each module occurs at most once in such a directed path. Thus the number of values to be remembered at any time does not exceed the size of the hierarchical description. $\square$

Although the algorithm of Theorem 3.1 requires only linear space, it may entail fruitless subsimulations. The subsimulation of a module instance from list *PARTIAL* is fruitful only if it produces an instance output-port value that was not previously known. Fruitless subsimulations can be avoided by using auxiliary information. For each module an interface dataflow graph can be computed in a bottom-up manner [LWan], where this graph indicates which input ports determine which output ports. Then a module is placed on list *PARTIAL* only if its known input-port values determine a not yet known output port. Since the size of an interface dataflow graph can be the product of the number of input ports and number of output ports, the use of this technique may require quadratic space.

4. Acyclic circuits: Lower bounds. In this section we investigate the inherent computational complexity of several analysis problems for hierarchically described acyclic circuits and focus on the simulation problem.

Reference [LWag] presents a sketch of a proof that the simulation problem is PSPACE-hard for strongly acyclic hierarchically specified logic circuits. The reduction used is from the quantified Boolean formula logical validity problem [SM] and is itself $\Theta(n^2)$-size-bounded. The reduction in [SM] used to prove that the quantified Boolean formula logical validity problem is PSPACE-hard is a $\Theta(n^2)$-size-bounded reduction from the membership problem for linear-bounded automata. Thus the reduction in [LWag] provides a $\Theta(n^4)$-size-bounded reduction from the membership problem for linear-bounded automata. If it is assumed that there are languages in DSPACE(n) whose recognition requires $2^{\Omega(n)}$ time, the reduction in [LWag] only provides evidence that

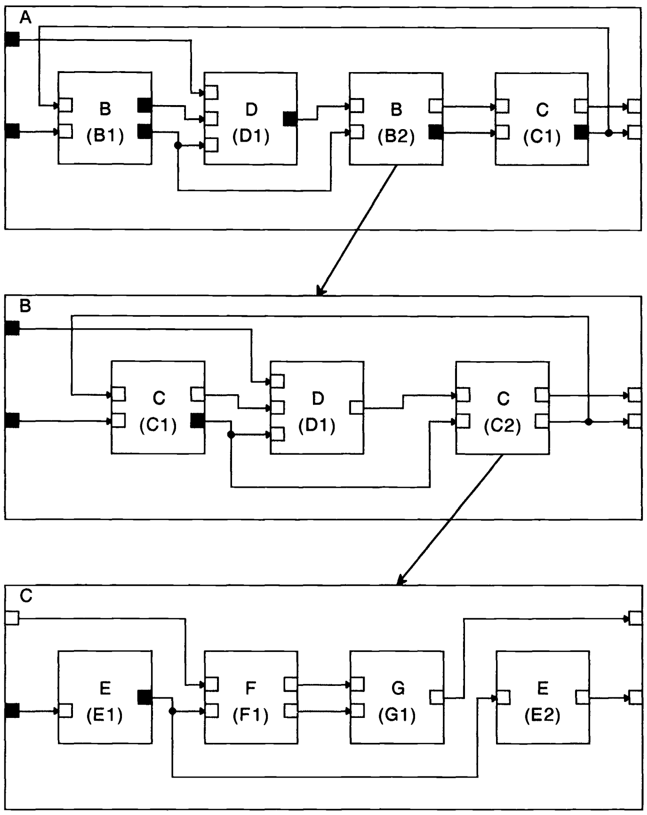


FIG. 5. *Snapshot of simulation of a weakly acyclic circuit.*

the simulation problem for strongly acyclic hierarchically specified logic circuits requires time $2^{\Omega(n^{1/4})}$. Also, it can readily be seen that the logical validity problem for quantified monotone Boolean formulas is decidable deterministically in polynomial time and thus cannot be used to prove the PSPACE-hardness of the simulation problem for hierarchically specified monotone logic circuits. (This contradicts an apparent claim in [LWag].)

Here we present a polynomial-time $\Theta(n^2)$-size-bounded reduction from the membership problem for deterministic linear-bounded automata to the simulation problem for explicitly acyclic hierarchically specified monotone circuits. If it is assumed that there are languages in DSPACE(n) whose recognition requires $2^{\Omega(n)}$ time, our reduction provides evidence that the simulation problem for strongly acyclic hierarchically specified logic circuits requires $2^{\Omega(\sqrt{n})}$ time. In §5 below we present a matching upper bound. Our reduction and lower bound apply not only to the simulation of monotone circuits but also to the simulation or evaluation of all classes of strongly acyclic hierarchically specified functions, for which the allowed primitive function modules can emulate monotone Boolean logic. Thus our reduction and lower bound apply to strongly acyclic hierarchical specified functions on many different algebraic structures with a 0 and 1. These include all of the following algebraic structures (provided that they have at least two elements): lattices, rings with a multiplicative identity, idempotent semirings with a multiplicative identity, finite semirings with a multiplicative identity that are not rings, etc. For examples of these structures see [BHR], [Ha], [MacLB], and [Zim]. In particular, our reduction and lower bound apply to the various lattice-theoretical structures used to simulate faults, errors, transients, unknown states, variable strength signals, etc., in digital logic both at the gate and transistor levels [Ha].

Before presenting our reduction, we need the following definition.

DEFINITION 4.1. We say that a set of primitive modules has *monotone logic expressibility* if there exist two values ϕ and τ, in the domain of values operated on by the primitive modules, and two modules *OR* and *AND*, either available as primitive modules or constructible as composite modules by an interconnection of primitive modules, with the following properties. Modules *OR* and *AND* each have two inputs and one output. If both outputs of *OR* equal ϕ, the output value is ϕ; if one input value is ϕ and the other is τ, the output is τ; and if both inputs are τ, the output is τ. If both inputs of *AND* are ϕ, the output is ϕ; if one input is ϕ and the other is τ, the output is ϕ; and if both inputs are τ, the output is τ.

THEOREM 4.2. *Let* Π *be a set of primitive modules with monotone logic expressibility. Then each language in* $DSPACE(n)$ *is polynomial time and* $\Theta(n^2)$*-size-bounded reducible to the following problem* Γ: *Given a strongly acyclic hierarchical object description whose primitive modules are in the set of primitive modules* Π, *an assignment of values to the input ports of the root module, and a specified value for one of the output ports of the root module, determine whether the specified object, given the specified input values, produces the specified output value.*

Proof. Consider a deterministic linear-bounded automaton M. The description of M can be modified, if necessary, so that once it accepts an input string, it cycles in an accepting state. Also, M can be modified, if necessary, so that its head never moves off the end of its tape. There is a constant $c > 0$ such that for any input sequence x to M, where x includes endmarkers and $n = |x|$, if M accepts x, it does so within 2^{cn} moves. Thus M accepts x if and only if after 2^{cn} moves M is in an accepting state.

As shown below, M and x can be encoded into a strongly acyclic hierarchical monotone object description and an input value assignment such that the hierarchically specified object produces output τ for the given input assignment if and only if M accepts x. With M fixed, this encoding represents a polynomial-time algorithm whose input is x and whose output, consisting of a hierarchical object description together with an input value assignment, is of size $\Theta(|x|^2)$. Thus the algorithm is the required reduction from $L(M)$ to the simulation problem for strongly acyclic hierarchical monotone object descriptions.

For each m, where $0 \leq m \leq cn$, there is a composite module E_m. The ports of these modules are summarized in Fig. 6. For each tape cell i of M, where $1 \leq i \leq n$, and each tape symbol a of M, each module E_m has an input port $t_{i,a}$ and an output port $u_{i,a}$. In addition, for each tape cell i, $1 \leq i \leq n$, each module E_m has an input port r_i and an output port v_i. Finally, for each state g of M each module E_m has an input port s_g and an output port w_g. The purpose of the input ports is to encode a configuration of M. The value of input port $t_{i,a}$ indicates whether, in the configuration, cell i has tape symbol a written on it. A value of τ indicates that this is so, and a value of ϕ indicates that it is not so. Input port r_i indicates whether the tape head is scanning cell i. Input port s_g indicates whether the state is state g. The output ports encode a configuration in a similar manner. Note that since M is fixed, the number of ports is proportional to n, the length of x.

For each M, where $0 \leq m \leq cn$, module E_m will be designed so that if the inputs encodes a length n configuration of M, then the outputs of E_m will encode the configuration that results after 2^m moves by M.

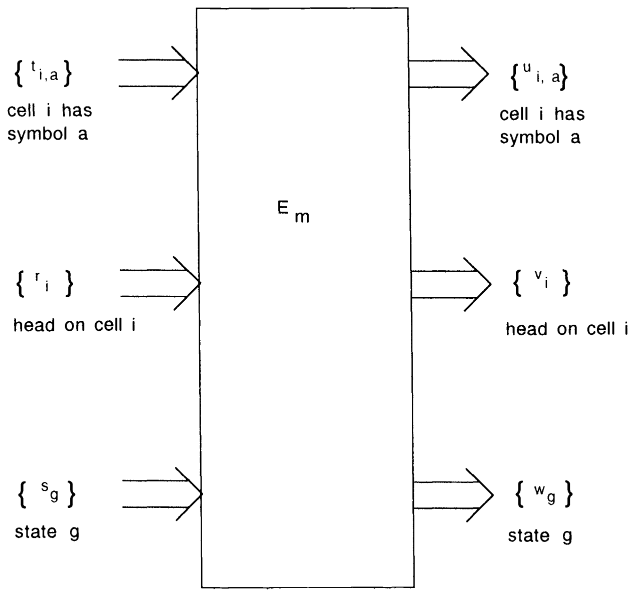


FIG. 6. *Ports of module E_m.*

Consider E_0, whose purpose is to simulate one move of automaton M. The body of E_0 is constructed by using instances of modules *OR* and *AND*. For simplicity, we describe E_0 as though *OR* and *AND* can have multiple inputs, with the understanding that the circuit is actually constructed by using two-input module instances.

Suppose that automaton M has transition function δ, where $\delta(g, a) = (h, b, p)$ means that when M is in state g and scanning tape symbol a, it makes a transition to state h, writes symbol b on the scanned tape cell, and moves its tape head p cells to the right, where the value of p is constrained to be $-1, 0$, or $+1$.

For each tape cell i the body of module E_0 contains a signal q_i, which is the *OR* of all body input ports r_j, where $j \neq i$. Thus q_i encodes whether the tape head of M is not on cell i. In addition, for each cell i, tape symbol a, and state g, the body of module E_0 contains a signal $y_{i,a,g}$, which is the *AND* of $t_{i,a}$, r_i and s_g.

For each state h, the output port w_h is the result of the *OR* of each $y_{i,a,g}$ such that $\delta(g, a) = (h, b, p)$ for some tape symbol b and some p.

For each tape cell i and each tape symbol b, output port $u_{i,b}$ is the result of the *OR* of

(i) *AND* of q_i and $t_{i,b}$, and

(ii) $y_{i,a,g}$ for each tape symbol a and state g such that $\delta(g, a) = (h, b, p)$ for some state h and some p.

For each tape cell i, output port v_i is the result of the *OR* of

(i) $y_{i,a,g}$ for each tape symbol a and state g such that $\delta(g, a) = (h, b, 0)$ for some state h and tape symbol b,

(ii) $y_{i-1,a,g}$ for each tape symbol a and state g such that $\delta(g, a) = (h, b, +1)$ for some state h and tape symbol b, provided cell i is not the leftmost tape cell, and

(iii) $y_{i+1,a,g}$ for each tape symbol a and state g such that $\delta(g, a) = (h, b, -1)$ for some state h and tape symbol b, provided cell i is not the rightmost tape cell.

E_0 has been designed so that if its inputs encode a configuration of M, then its outputs encode the configuration that results from one move of M. Note that the size of E_0, measured in terms of the number of instances of two input *AND* and *OR* modules, is quadratic in n.[1]

Next, consider the body of each E_m, where $1 \leq m \leq cn$. The body of each such E_m is constructed by using two instances of module E_{m-1}, connected as shown in Fig. 7. The body input ports of E_m drive the input ports of the first instances of E_{m-1}, whose output ports drive the input ports of the second instance of E_{m-1}, whose output ports drive the body output ports of E_m. Since the outputs of each instance of E_{m-1} encode the configuration of M produced after 2^{m-1} moves from the configuration encoded by its inputs, the outputs of E_m encode the configuration produced after 2^m moves from the configuration encoded by the inputs of E_m.

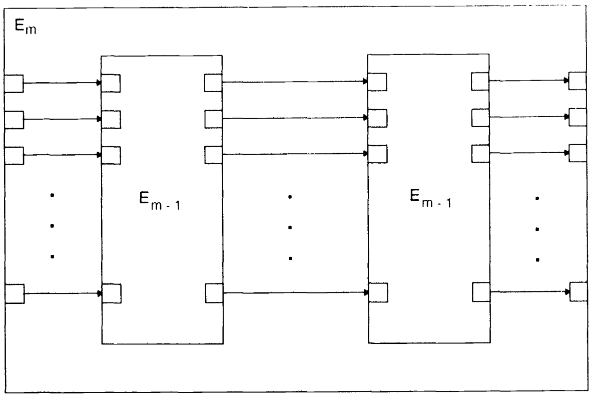


FIG. 7. *Body of E_m for $m > 0$.*

Note that the size of the description of the body of each such E_m is linear in n and that the number of such descriptions is cn.

Let *FIN* be a module with one output port and a number of input ports equal to the number of output ports of E_0. The body of *FIN* consists of instances of *OR* modules that compute the result of the *OR* of those input-port variables that represent an accepting state of M. Thus when the input ports of *FIN* encode a configuration of M, the output of *FIN* encodes whether the configuration is an accepting configuration.

Let E be a module with one output port and a number of input ports equal to the number of input ports of E_0. The body of E is shown in Fig. 8. The body input ports of E drive an instance of E_{cn}, whose output ports drive an instance of *FIN*.

Consider the assignment of values to the input ports of E such that this assignment encodes the initial configuration of M given x. (Each value in this assignment is either τ or ϕ.)

When module E is supplied with this input assignment, the value of the output port of E equals τ if and only if automaton M accepts x.

The module E is strongly acyclic, and the total size of the constructed modules is $\Theta(n^2)$. Thus the construction represents a polynomial-time $\Theta(n^2)$-size-bounded reduction from $L(M)$ to the simulation problem for E. $\square$

[1] Although it is possible to design E_0 so that its size is linear in n, this entails providing extra ports to hold complementary values and would not strength the result.

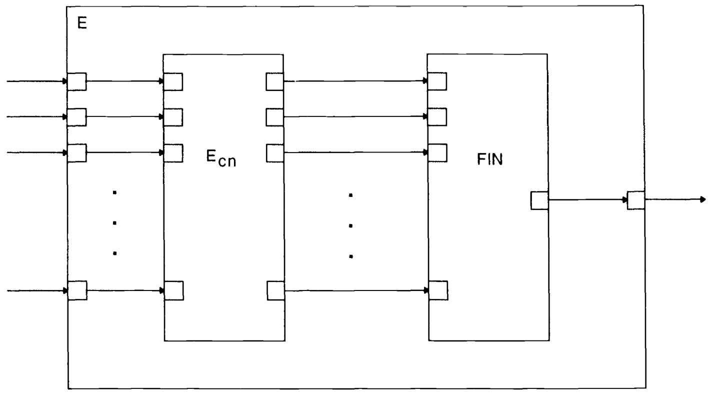


FIG. 8. *Body of root module* E.

An immediate consequence of Theorem 4.2 is that problem Γ is PSPACE-hard. Moreover, if some language in DSPACE(n) requires time $2^{\Omega(n)}$, then problem Γ requires time $2^{\Omega(\sqrt{n})}$.

5. $2^{O(\sqrt{n})}$ time simulation of acyclic circuits. In this section we show that weakly acyclic hierarchically specified logic circuits can be simulated deterministically using only $2^{(\sqrt{n})}$ time. The linear space simulation of §3 suggests that perhaps simulation of an acyclic hierarchical circuit of size n might require time $2^{\Theta(n)}$. Also, the size of the flattened circuit for a size n acyclic hierarchical circuit is $2^{\Theta(n)}$, so that the traditional approach of first flattening the hierarchically described circuit and then simulating it would take time $2^{O(n)}$. However, in Theorem 5.11 below we show that it is possible to do the simulation much faster when the domain of values involved in the simulation is finite, namely, in time $2^{O(\sqrt{n})}$. Moreover, we carefully analyze the constant in the exponent and show that it is of reasonable size. To do this we need the following notation and technical lemmas.

DEFINITION 5.1. The *submodule size* of a module M equals 1 if M is primitive and equals the number of occurrences of submodules appearing directly in the body of M otherwise. The *submodule size* of a hierarchical description D is the sum of the submodule sizes of the modules appearing in D.

DEFINITION 5.2. The *module expansion tree M-Tree(D)* of a hierarchically specified module D is defined as follows: *M-Tree(D)* is a labeled unordered tree such that

(i) each node of *M-Tree(D)* is labeled by the name of a module Z such that Z *SOMEWHERE-WITHIN D*,

(ii) the root of *M-Tree(D)* is labeled by the name of the root module D,

(iii) each node of *M-Tree(D)* labeled by the name of a primitive module has no children,

(iv) each node of *M-Tree(D)* labeled by the name of a composite module, say, module Z, has a child for each module instance in the body of Z. Each child is labeled with the module name of the instance.

As an example, Fig. 9 shows *M-Tree(A)* for the module A that appears in Fig. 5. (Note that module D is primitive.)

LEMMA 5.3. *Let $n \geq 1$ be an integer. The maximum number of leaves in the expansion tree of a hierarchically specified module whose hierarchical description is constrained to have*

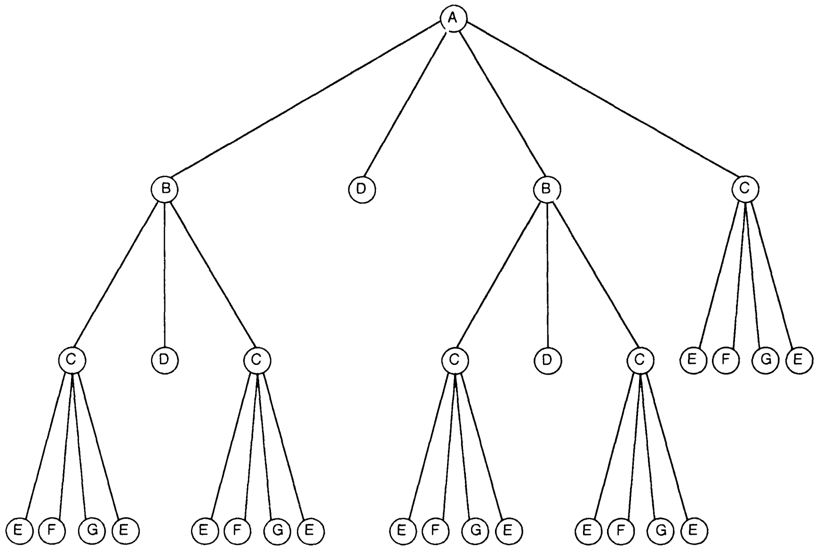


FIG. 9. *Module expansion tree M-Tree*(A).

submodule size at most n occurs when only one kind of submodule occurs at each level in the expansion tree of the hierarchically specified module.

Proof. The following algorithm, given a hierarchically specified module M, produces a hierarchically specified module N whose hierarchical description has submodule size no greater than that of M, such that

(i) the number of leaves in *M-Tree(M)* is less than or equal to the number of leaves in *M-Tree(N)*, and

(ii) only one kind of submodule occurs at each level in the expansion tree of hierarchically specified module N.

The algorithm processes the body of the composite modules that occur as labels in *M-Tree(M)* as follows. For each body let Q be a module such that there is an instance of Q within the body and *M-Tree(Q)* has a maximal number of leaves among the kinds of modules having instances in the body. The body is modified by replacing each instance of a module other than Q with an instance of Q.

After all the bodies have been modified, modules that no longer label nodes in *M-Tree(M)* are deleted.

The correctness of the lemma follows since the submodule size of each module is unchanged and the number of leaves in the expansion tree of each module is either unchanged or increased. □

LEMMA 5.4. *Let* $m, k, i_1, \ldots,$ *and* i_k *be integers* ≥ 1 *such that* $\sum_{j=1}^{k} i_j = m$. *Then the product* $\Pi_{j=1}^{k} i_j$ *is less than or equal to* 3^r if $m = 3 \cdot r$, $3^{r-1} \cdot 2^2$ *if* $m = 3 \cdot r + 1$, *and* $3^r \cdot 2$ *if* $m = 3 \cdot r + 2$. *Moreover, the indicated upper bound is obtainable.*

Proof. It is easily seen that the indicated upper bound is obtainable. To verify that the claimed upper bound is an upper bound, we need only verify the following: Let $m, k, i_1, \ldots,$ and i_k be integers ≥ 1. Let m be fixed.

(1) The product $i_1 \cdot \ldots \cdot i_k$, subject to the constraint $i_1 + \ldots + i_k = m$, is maximized when $i_1, \ldots, i_k \in \{2, 3\}$.

(2) Given that $i_1, \ldots, i_k \in \{2, 3\}$, the product $i_1 \cdot \ldots \cdot i_k$, subject to the constraint $i_1 + \ldots + i_k = m$, is maximized when at most two of the integers $i_1, \ldots, i_k$ equal 2.

To see the correctness of (1), we observe the following:

(1a) Suppose some $i_l \geq 5$. Then the product $i_1 \cdot \ldots \cdot i_k < \left(\Pi_{j=1,j\neq l}^{k} i_j\right) \cdot 2 \cdot (i_l - 2)$ (since $i_l < 2 \cdot (i_l - 2)$) and $\sum_{j=1,j\neq l}^{k} i_j + 2 + (i_l - 2) = m$.

(1b) Suppose some $i_l = 4$. This occurrence of 4 can be replaced by two occurrences of 2; the sum and product of the integers will be unchanged.

To see the correctness of (2), we observe that $6 = 2 + 2 + 2 = 3 + 3$ *but* $2^3 < 3^2$. □

DEFINITION 5.5. A set of primitive modules Π is *complete* over a domain D of values if D is the domain over which the primitives operate and if every function of the form $D^k \to D$ for finite k can be computed by a finite interconnection of modules in Π.

LEMMA 5.6. *For any set Π of primitive modules that is complete over a finite domain with d values, there is a constant c such that any acyclic circuit with z ports has an equivalent circuit of size at most czd^z.*

Proof. Of the z ports, let x be the number of input ports and let y be the number of output ports. The circuit computes a function from d^x to d^y. Let each of the d^x input value assignments be called a *minterm*.

Since Π is complete, a circuit of a fixed size can perform each of the binary operations in Z_d. Thus for each minterm a circuit that computes a signal whose value is 1 if the input assignment corresponds to that minterm, and whose value is 0 otherwise, can be constructed. The size of the circuit for each minterm is at most $c_1 x$, where constant c_1 depends on Π. Each of the y outputs can be computed by a circuit that combines the values of the d^x minterms, and the size of this circuit is at most $c_2 d^x$, where constant c_2 depends on Π. If c is $\max(c_1, c_2)$, the size of the overall circuit is at most czd^z. □

DEFINITION 5.7. Consider a hierarchical circuit description D of size n. Let M be a hierarchically described module occurring in D such that the number of ports of M is at least $\sqrt{n}$. The *module semiexpansion tree* of M, denoted by *S-Tree(M)*, is a labeled unordered tree such that

(i) each node of *S-Tree(M)* is labeled by the name of a module that occurs in the description D and that has at least $\sqrt{n}$ ports,

(ii) the root of *S-Tree(M)* is labeled by the name of module M,

(iii) each node of *S-Tree(M)* labeled by the name of a primitive module has no children,

(iv) each node of *S-Tree(M)* labeled by the name of a composite module, say, module Z, has a child for each instance in the body of Z of a module with at least $\sqrt{n}$ ports. Each child is labeled with the module name of the instance.

Note that *S-Tree(M)* corresponds to the expansion tree of M that would be produced if D were modified by deletion of all occurrences of modules and module instances having less than $\sqrt{n}$ ports.

LEMMA 5.8. *Consider a hierarchical description D of size n, and a module M occurring in D such that M has at least $\sqrt{n}$ ports. Then*

(1) *S-Tree(M) has depth at most $\sqrt{n}$, and*

(2) *S-Tree(M) contains at most $3^{\lceil\sqrt{n}/3\rceil}$ leaves.*

Proof. (1) Since each node along a path from the root to a leaf of *S-Tree(M)* corresponds to a distinct module whose description is of size at least $\sqrt{n}$, the number of such modules cannot exceed $\sqrt{n}$.

(2) Note that the proof of Lemma 5.3 applies to *S-Trees*. Thus for a given value of n the number of leaves is maximized when each module body contains instances of only

one module type. Thus assume that at each level of *S-Tree(M)* all the nodes at that level are labeled by the same module. Let $M_0, M_1, \ldots, M_k$ be the module labeling the root, children of the root, etc. For $1 \leq i \leq k$ let i_j be the number of instances of module M_j within the body of module M_{j-1}. Since the total size of the description is n and each module instance has size at least $\sqrt{n}$, there can be at most $\sqrt{n}$ module instances contributing to *S-Tree(M)*. Thus it must be the case that $\sum_{j=1}^{k} i_j \leq \sqrt{n}$. Therefore, from Lemma 5.4, $\sum_{j=1}^{k} i_j$ does not exceed $3^{\lceil \sqrt{n}/3 \rceil}$. But this product is the number of leaves *S-Tree(M)*. □

THEOREM 5.9. *For any set* Π *of primitive modules that is complete over a finite domain with d values there is a constant c such that any acyclic circuit specified by a hierarchical description of a size n has an equivalent circuit whose flat description is of size at most* $cn^{3/2}2^{((1/3)\log_2 3+\log_2 d)\sqrt{n}}$.

Proof. Suppose a module M has a most $\sqrt{n}$ ports. Then from Lemma 5.6 it has an equivalent circuit of size at most $c_1\sqrt{n}\,d^{\sqrt{n}}$, where c_1 depends on Π.

Suppose M has more than $\sqrt{n}$ ports. From Lemma 5.8 *S-Tree(M)* contains at most $3^{\lceil \sqrt{n}/3 \rceil}$ leaves. Each of these leaves has a module body containing at most n module instances, and each of these module instances has at most $\sqrt{n}$ ports. Therefore, if M were expanded by continually replacing module instances with more than $\sqrt{n}$ ports by their bodies and not expanding any module instance with at most $\sqrt{n}$ ports, the resulting circuit would contain at most $n3^{\lceil \sqrt{n}/3 \rceil}$ instances of modules, each with at most $\sqrt{n}$ ports. From Lemma 5.6 each of these module instances can be replaced by a flat circuit containing at most $c_1\sqrt{n}\,d^{\sqrt{n}}$ instances of primitive modules. If we let $c = 3c_1$, the overall circuit contains at most $cn^{3/2}2^{((1/3)\log_2 3+\log_2 d)\sqrt{n}}$ instances of primitive modules. □

Note that since $\log_2 3$ is less than 1.59, the bound from the preceding theorem is $cn^{3/2}2^{(0.53+\log_2 d)\sqrt{n}}$. When the domain of the circuit is binary, so that $d = 2$, the bound is $cn^{3/2}2^{1.53\sqrt{n}}$.

THEOREM 5.10. *For any set* Π *of primitive modules that is complete over a finite domain with d values, there is a constant c such that for any acyclic circuit specified by a hierarchical description of size n an equivalent flat description can be constructed in time* $cn^{5/2}2^{((1/3)\log_2 3+2\log_2 d)\sqrt{n}}$.

Proof. Let c_1 be the constant in the statement of Theorem 5.9. Let M be the module whose description is to be constructed. Then working from the bottom up in accordance with the *DIRECTLY-WITHIN* relation, an "official" flat circuit is constructed for each of the modules that are involved in the hierarchical description of M and have fewer than $\sqrt{n}$ ports. Finally, the flat circuit for M is constructed. For each module N considered with fewer than $\sqrt{n}$ ports, the official flat circuit is constructed as described in Lemma 5.6. To use this construction we need a table that gives the output values for each assignment of input values. This table is constructed by first producing a "working body" for N and then simulating the working body for each input assignment. The working body is constructed as described by Theorem 5.9, continually substituting the body for each instance of a module, but when an instance of a module with less than $\sqrt{n}$ ports is encountered, its already constructed official body is directly substituted for the instance. The result is that the working body for N is of size at most $c_1n^{3/2}2^{((1/3)\log_2 3+\log_2 d)\sqrt{n}}$. The table for N requires $d^{\sqrt{n}}$ rows, and each row can be filled in by simulating the working body for N, given the input assignment corresponding to that row. Thus the time to fill in the table is proportional to $n^{3/2}2^{((1/3)\log_2 3+2\log_2 d)\sqrt{n}}$.

After at most n official bodies are constructed, the flat circuit for M can be constructed as described in Theorem 5.9. □

THEOREM 5.11. *For any set Π of primitive modules that is complete over a finite domain with d values, there is a constant c such that any acyclic circuit specified by a hierarchical description of size n can be simulated in time at most* $cn^{5/2}c^{((1/3)\log_2 3+2\log_2 d)\sqrt{n}}$.

Proof. A flat circuit equivalent to the module to be simulated can be constructed as described in Theorem 5.10 and then can be directly simulated. □

In performing a simulation it is not really necessary to construct the circuits for modules with fewer than $\sqrt{n}$ ports. Rather, each such module can be represented by a table that provides the output values for each input assignment. Each such table can be initially empty. Each time the module is simulated with an input assignment that has not been submitted thus far, the output values in the appropriate row can be filled in. When the module is to be simulated with an input assignment that has already been simulated, the output values in the appropriate row can be used (without having to repeat this simulation). Thus a given overall simulation might take less time than the construction that is the basis of Theorem 5.11 would take.

Note that given a hierarchically described circuit, the size of the flattened circuit can be computed in linear time by using a bottom-up method. A choice can than be made between using the algorithm of Theorem 5.11 and simulating the flattened circuit, depending on a comparison of the size of the flattened circuit and the bound of Theorem 5.11. This hybrid algorithm never uses more time or more space than the traditional method of constructing the flattened circuit and then simulating it. Since the flattened circuit can be of size $2^{\Omega(n)}$, the hybrid algorithm often uses less time and much less space. If space is of the utmost importance, the algorithm of Theorem 3.1 can be used.

6. Analysis problems for circuits with cycles. In this section we consider hierarchical module descriptions in which dataflow cycles are permitted. Simulation for circuits with dataflow cycles involves computing the values of the output ports of a module, given its hierarchical description, a specification of the sequence of values for each input port, a specification of the initial state of the module, and a specification of which values should be reported.

As for acyclic modules, we assume that the domain of signal values is finite. We also assume that for each input port of a primitive module there is a rule for what value to use should the port be unconnected to a driver.

Because of the presence of cycles, timing issues arise. We assume that for any given set of primitive modules, all delays are multiples of some basic unit of time. We also assume that each primitive module can have a state (where combinational modules are a special case that have only a single state). For each primitive module we assume that there is a rule computable in polynomial time and linear space that when given a state and input-port values, determines a next state and output-port values, perhaps with a specified delay.

Consider a set of primitive modules that satisfy the conditions described above. The *input information* for a *simulation problem* consists of a hierarchical module description, the specification of a sequence of values for each input port of the root module, the specification of initial values for states of some of the primitive modules and some signals in the flattened body of the module, the specification of a condition of when to stop the simulation, and the specification of conditions for when the output values should be reported. We make the following assumptions about the language in which these specifications are expressed. We assume that time can be written in binary. We assume that the language for expressing the sequence of values for an input port can enumerate values at given times or during given time intervals, can specify repetition of a sequence of values, and can specify default values for times not explicitly described. We assume that the language for expressing initial values for states or signals in the flattened body can specify a value for certain explicitly listed signals and primitive module instances in the flattened body and can specify default values for those not explicitly listed. We assume that for explicitly listing a signal or module its *hierarchical name* is given, where the hierarchical name begins with the name of the root module and consists of an identifying sequence of names separated by periods. The size of a hierarchical name is the number of names in the sequence. For instance, hierarchical name $A.B.C.D$ refers to D within instance C within instance B of module A. The size of this hierarchical name is 4. We assume that the stop condition can be either a specified time or a specified value for one of the output ports. We assume that the report condition can be a list of times or time intervals or can be a Boolean combination of output-port values.

The *output information* for a simulation problem is the values of the output ports of the root module at those times satisfying the specified report conditions, up to the time the stop condition is satisfied.

PROPOSITION 6.1. *For a given set of primitive modules, the simulation problem for flat circuit descriptions can be solved in linear space.*

Proof. The simulation can keep track of the value of each signal in the flat body plus, if appropriate, states of the primitive modules. □

THEOREM 6.2. *A hierarchically specified circuit can be simulated in exponential space.*

Proof. The size of the flattened body of the module is at most exponential in the size of the hierarchical description. Thus the flattened body can be constructed in exponential space and then simulated. □

We now show that the simulation problem requires exponential space.

We say that a set of primitive modules has *flip-flop expressibility* if there exist two values ϕ and τ in the domain of values operated on by the primitive modules and if there exist four modules {*AND*, *OR*, *NOT*, *FLIP-FLOP*} either available as primitive modules or constructible as composite modules by an interconnection of primitive modules such that these modules behave like the standard logic modules with these names. The *FLIP-FLOP* module can behave like any of the standard flip-flop types. (The *FLIP-FLOP* module can be a composite module constructed out of the gate-type modules.)

For a set of primitive modules with flip-flop expressibility, a linear-bounded automaton with a given input string can be described as a flat module whose size is proportional to the length of the input string. Consequently, the simulation problem for flat modules constructed from primitive modules that have flip-flop expressibility is PSPACE-hard. Because of Proposition 6.1, the problems is PSPACE-complete. Hierarchical descriptions permit an exponential increase in conciseness but involve a corresponding exponential increase in the space required for the simulation problem, as described in the following result.

THEOREM 6.3. *For a set Π of primitive modules with flip-flop expressibility, there exists a constant $d > 0$ such that the simulation problem for hierarchical module descriptions requires space at least 2^{dn} on any Turing machine.*

Proof. Let M be an arbitrary 2^{cn}-space-bounded Turing machine for some constant $c > 0$. Assume that M serves as a language recognizer. Also assume that M has an explicit accept state and an explicit reject state. Without loss of generality it can be assumed that for every input string, M eventually enters either its explicit accept or explicit reject state without having touched more than its 2^{cn} leftmost tape cells, and it then moves to the right indefinitely in its accept or reject state.

Consider an input string x of length n, where x is assumed to include endmarkers. In processing x machine M uses at most 2^{cn} tape cells and then keeps moving to the right in either the accept or reject state.

The overall circuit to be specified hierarchically will contain an implementation of Turing machine M and enough tape cells to determine whether M accepts x. In this implementation each tape cell will be implemented explicitly by an instance of a submodule, called module D_0, that can both record the contents of a tape cell and simulate the operation of M's finite state control when the tape head of M is on that cell. A transition of M will be implemented by changes of signal values involved in instances of D_0 corresponding to the tape cells affected by the transition, with other instances of D_0 remaining unchanged.

Given x, a hierarchical circuit description can be constructed as follows. The circuit description depends only on n, the length of x. For each m, where $0 \leq m \leq cn$, there is a composite module D_m. In the overall circuit that is specified hierarchically, each instance of D_m represents a segment of 2^m contiguous cells of M's tape. The flip-flops in this instance of D_m record the contents of the tape cells in this segment, whether or not the tape head is residing on this segment and, if so, the state of M and on which cell in the segment the head is residing. The ports of D_m are summarized in Fig. 10. For each state g of M input port xl_g encodes whether the tape head, in state g, is moving onto the represented tape segment from the left. Similarly, input port xr_g encodes whether the tape head, in state g, is moving onto the tape segment from the right. Output port yl_g encodes whether the tape head, in state g, is moving off the tape segment to the left. Similarly, output port yr_g encodes whether the tape head, in state g is moving off the tape segment to the right.

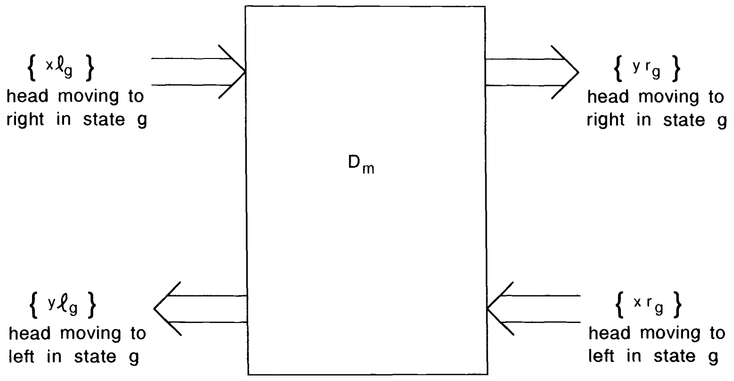

{ xℓg }
head moving to right in state g
{ yrg }
head moving to right in state g
Dm
{ yℓg }
head moving to left in state g
{ xrg }
head moving to left in state g

FIG. 10. *Ports of module D_m.*

Consider D_0, whose purpose is to simulate one cell of M. The body of D_0 can be constructed by using instances of modules *FLIP-FLOP*, *OR*, *AND*, and *NOT*. The flip-flops are used to remember the contents of the tape cell, whether or not the tape head is on the cell, and, if so, the state. A straightforward implementation can have a flip-flop for each tape symbol and a flip-flop for each state. At each step of the computation by M, the head of M resides at some given tape cell, say, cell α. On the basis of the contents of cell α and the current state, M makes a transition that involves a next state, new contents of cell α, and the determination of whether the head of M remains stationary, moves one cell to the left, or moves one cell to the right. Corresponding to this step of M, the instance of D_0 representing cell α uses its recorded value of the state and cell contents to simulate the transition. The new contents of cell α are recorded in this instance of D_0. If the transition involves no head movement, then the next state and the fact that the head is residing on this cell is recorded in this instance of D_0. If the transition involves movement of the tape head, then the appropriate output port of this instance of D_0 is given a value indicating the movement of the tape head onto an adjacent cell and the new state of M. This value on the output port causes the instance of D_0 representing the adjacent tape cell to record the fact that the tape head is now residing on it. (Note that for all instances of D_0 other than the one that represents cell α, the values on the output ports remain unchanged.) The details of the construction of the body of D_0 are routine and are left to the reader.

Now consider the body of each D_m, where $1 \leq m \leq cn$. The body of each D_m is constructed by using two instances of module D_{m-1} interconnected as shown in Fig. 11.

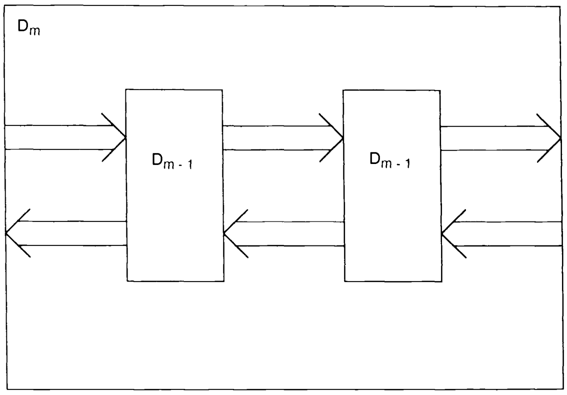


FIG. 11. *Body of D_m for $m > 0$.*

Note that the size of the description of each D_m is independent of n and that the number of such descriptions is cn.

Let *INIT* be a module whose body consists of a sequence of n instances of D_0 connected together. The purpose of *INIT* is to represent the first n tape cells of M, which are initially to contain string x. (All other cells of M are initially to contain the blank tape symbol.) Module *INIT* has the same set of ports as described in Fig. 10. Note that the size of the description of *INIT* is proportional to n.

Let *FIN* be a module with two output ports $\{A,R\}$ and a number of input ports equal to the number of right output ports of D_{cn}. The body of *FIN* computes whether its input ports indicate that the tape head is moving to the right in the explicit accept or reject state. Output port A is given a special value when *FIN* detects the tape head moving to the right in the accept state, and output port R is given a special value when *FIN* detects the tape head moving to the right in the reject state.

Let D be a module with two output ports and no input ports. The body of D is shown in Fig. 12. In Fig. 12 the left input ports of *INIT* and the right input ports of D_{cn} are shown as unconnected. This assumes that the unconnected input ports of D_0 are interpreted as representing the absence of tape head movement. If this is not the case for the primitive modules used in the body of D_0, the body of D could be given additional module instances to generate the appropriate input-port values.

Consider the following instance of the simulation problem. The hierarchical module description consists of the description of D, *INIT*, *FIN*, $D_0, \ldots, D_{cn}$, with D designated as the root module. The specification of the initial values of flip-flops in the flattened

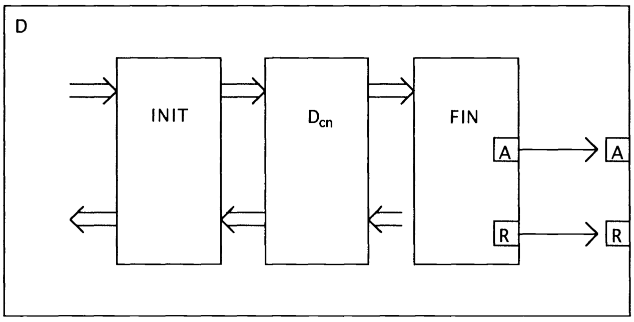


FIG. 12. *Body of root module D.*

circuit is that the n instances of D_0 within *INIT* are initialized with the n symbols of x and that all the instances of D_0 within D_{cn} are initialized with the blank symbol. Also, the leftmost instance of D_0 within *INIT* is specified to be initialized with the tape head present and the starting state of M. All other occurrences of D_0 are initialized with the tape head absent. The condition for both reporting the output and stopping the simulation is that either output of D has the special value that indicates acceptance or rejection by M.

For this simulation problem the simulation always halts, only one set of output-port values is reported, and the reported values indicate whether M accepts or rejects x.

Note that the size of the hierarchical object description is proportional to n. The size of the initialization specification is proportional to n since the size of the hierarchical name of each instance of D_0 within *INIT* is constant. (If the language for specifying initialization conditions were more restricted, the construction could be modified so that all flip-flops would be specified to be in the same neutral state and D would be given an input that could be used to load specified values into the first n cells and then initiate the operation of M.) Thus the construction represents a linear size (and polynomial time) reduction from the acceptance problem for M to the simulation problem for D. Since the size of the input information for the simulation problem is proportional to n, there is a constant $d > 0$ such that the simulation problem requires space at least 2^{dn}. □

Combining Theorems 6.2 and 6.3 gives the following result.

COROLLARY 6.4. *The simulation problem is* EXPSPACE*-complete for hierarchically specified objects when the set of primitive modules has flip-flop expressibility.*

Acknowledgment. We wish to acknowledge one of the referees for pointing out a flaw in our original proof of Theorem 3.1.

REFERENCES

[AHU] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[BS] J. BENKOSKI AND A. J. STROJWAS, *A new approach to hierarchical and statistical timing simulations*, IEEE Trans. Computer-Aided Design, CAD-6 (1987), pp. 1039–1052.

[BO] J. L. BENTLEY AND T. OTTMAN, *The complexity of manipulating hierarchically defined sets of rectangles*, in Proc. Mathematical Foundations of Computer Science, J. Gruska and M. Chytil, eds., Lecture Notes in Computer Science 118, Springer-Verlag, Berlin, New York, 1981, pp. 1–15.

[BHR] P. A. Bloniarz, H. B. Hunt iii, and D. J. Rosenkrantz, *Algebraic structures with hard equivalence and minimization problems*, J. Assoc. Comput. Mach., 31 (1984), pp. 879–904.

[Cr] J. D. Crawford, *EDIF: A mechanism for the exchange of design information*, IEEE Design and Test, 2 (1985), pp. 63–69.

[GJ] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, San Francisco, 1979.

[Ha] J. P. Hayes, *Digital simulation with multiple logic values*, IEEE Trans. Computer-Aided Design, CAD-5 (1986), pp. 274–283.

[KS] R. Kolla and B. Serf, *The virtual feedback problem in hierarchical representations of combinatorial circuits*, Acta Informatica , 28 (1991), pp. 463–476.

[Len1] T. Lengauer, *Exploiting hierarchy in* VLSI *design*, in Proc. Aegean Workshop on Computing, F. Makedon et al., eds., Lecture Notes in Computer Science 227, Springer-Verlag, Berlin, New York, 1986, pp. 180–193.

[Len2] ———, *Hierarchical planarity testing algorithms*, in Proc. 13th International Colloquium on Automata, Languages and Programming, L. Kott, ed., Lecture Notes in Computer Science 226, Springer-Verlag, Berlin, New York, 1986, pp. 215–225.

[Len3] ———, *Efficient algorithms for finding minimum spanning forests of hierarchically defined graphs*, J. Algorithms, 8 (1987), pp. 260–284.

[LWag] T. Lengauer and K. W. Wagner, *The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems*, in Proc. 4th Annual Symposium on Theoretical Aspects of Computer Science, F. J. Brandenburg et al., eds., Lecture Notes in Computer Science 247, Springer-Verlag, Berlin, New York, 1987, pp. 100–113.

[LWan] T. Lengauer and E. Wanke, *Efficient solution of connectivity problems in hierarchically defined graphs*, SIAM J. Comput., 17 (1988), pp. 1063–1089.

[LM] Y. Levendel and P. R. Menon, *Fault simulation*, in Fault-Tolerant Computing: Theory and Techniques, Vol. I, D. K. Pradhan, ed., Prentice-Hall, Englewood Cliffs, NJ, 1986, pp. 184–264.

[LMS] R. Lipsett, E. Marschner, and M. Shahdad, *VHDL—the language*, IEEE Design and Test, 3 (1986), pp. 28–41.

[MacLB] S. MacLane and G. Birkhoff, *Algebra*, MacMillan, New York, 1967.

[MKS] J. Magee, J. Kramer, and M. Sloman, *Constructing distributed systems in conic*, IEEE Trans. Software Engrg., 15 (1989), pp. 663–675.

[MC] C. Mead and L. Conway, *Introduction to* VLSI *Systems*, Addison-Wesley, Reading, MA, 1980.

[Ni] C. Niessen, *Hierarchical design methodologies and tools for* VLSI *chips*, Proc. IEEE, 41 (1983), pp. 65–75.

[Oea] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, *The magic* VLSI *layout system*, IEEE Design and Test, 2 (1985), pp. 19–30.

[SS] L. K. Scheffer and R. Soetarman, *Hierarchical analysis of* IC *artwork with user defined abstraction rules*, in Proc. ACM/IEEE 22nd Design Automation Conference, June 1985, pp. 293–298.

[SM] L. J. Stockmeyer and A. R. Meyer, *Word problems requiring exponential time*, in Proc. 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 1-9.

[WA] T. J. Wagner, *Hierarchical layout verification*, IEEE Design and Test, 2 (1985), pp. 31–37.

[Zim] U. Zimmerman, *Linear and Combinatorial Optimization in Ordered Algebraic Structures*, North-Holland, Amsterdam, 1981.

SIAM J. COMPUT.
Vol. 22, No. 3, pp. 650–659, June 1993

A STOCHASTIC CHECKPOINT OPTIMIZATION PROBLEM*

E. G. COFFMAN, JR.†, LEOPOLD FLATTO†, AND PAUL E. WRIGHT†

Abstract. This paper provides an examination of an abstract moving-server system that models several computer applications, including software debugging and accessing compressed data. In this model, the server moves on the unit interval [0, 1], serving requests where they are encountered. The locations of successive requests are not known in advance, but they are known to be independent samples from a given distribution F on [0, 1]. Before serving a request, the server must be moved to a reset point to the left of the request. There is a choice of two reset points, one fixed at 0 and one, called the *checkpoint*, that can be moved in the course of serving requests. The cost of serving a request is proportional to the distance moved to the request from the chosen reset point. This paper formulates a stochastic optimization problem whose solution, for a wide class of distributions F, yields a policy for deciding the successive locations of the checkpoint so as to minimize the expected total cost of serving a sequence of requests. Results for both finite and infinite-horizon variations of the problem are presented, along with the properties required of the distribution F.

Key words. checkpoints, stochastic optimization, moving-servers, online algorithms

AMS(MOS) subject classifications. 60H30, 60K10, 68M07, 68M20

1. Introduction. The following moving-server system is formulated as a model of certain problems in software debugging and in accessing compressed data. The server moves along a finite interval satisfying requests for service that appear at random locations. As a convenient normalization the interval is taken to be [0, 1]. At the origin and at some point x to be determined, reset points are located; the server must always visit a reset point before serving a request. The reset point at 0 remains fixed, but the other can be moved in the course of serving requests. The movable reset point will be called the *checkpoint* to distinguish it from the reset point at 0.

To serve a request at y, the server moves in two stages. In the first stage it moves to a reset point to the left of y. This can be the reset point at 0 or the checkpoint at x if $x \leq y$, but it must be the reset point at 0 of $x > y$. Let z denote the location of the reset point chosen in the first stage. Then in the second stage the server moves right from z to point y and performs the requested service. During this stage, the server either repositions the checkpoint at some location x', $z \leq x' \leq y$, along its path; or, after the service at y, it may continue rightward and place the checkpoint at $x' > y$.

The class $\mathcal{C}$ of policies for serving a sequence of requests contains just those policies following the above ground rules. Such a policy is defined by the rules deciding the sequence of points at which the server is reset in the first stage of motion and the sequence of locations where the checkpoint is placed in the second stage of motion. In serving a sequence of requests at $y_1, \ldots, y_n$, the cost c_i of serving the request at y_i is defined to be the distance moved during the second stage of motion times a given rate parameter $r > 0$; the reset motion in the first stage incurs no cost.[1] Assuming that the y_i are independently and identically distributed samples from a given distribution F on [0, 1], the problem is to find a policy $\mathcal{P} \in \mathcal{C}$ that, for a given $n \geq 1$ and an initial position x_0 of the checkpoint, minimizes the expected total cost $\sum_{i=1}^{n} E(c_i)$ of serving a sequence of n requests.

*Received by the editors December 9, 1991; accepted for publication (in revised form) April 9, 1992.

†AT& T Bell Laboratories, Murray Hill, New Jersey 07974.

[1]Charging a fixed cost c to modify the checkpoint (updating a fixed sized dictionary in the data decompression case, for example) does not lead to greater generality; the linear cost cn may be added at the end.

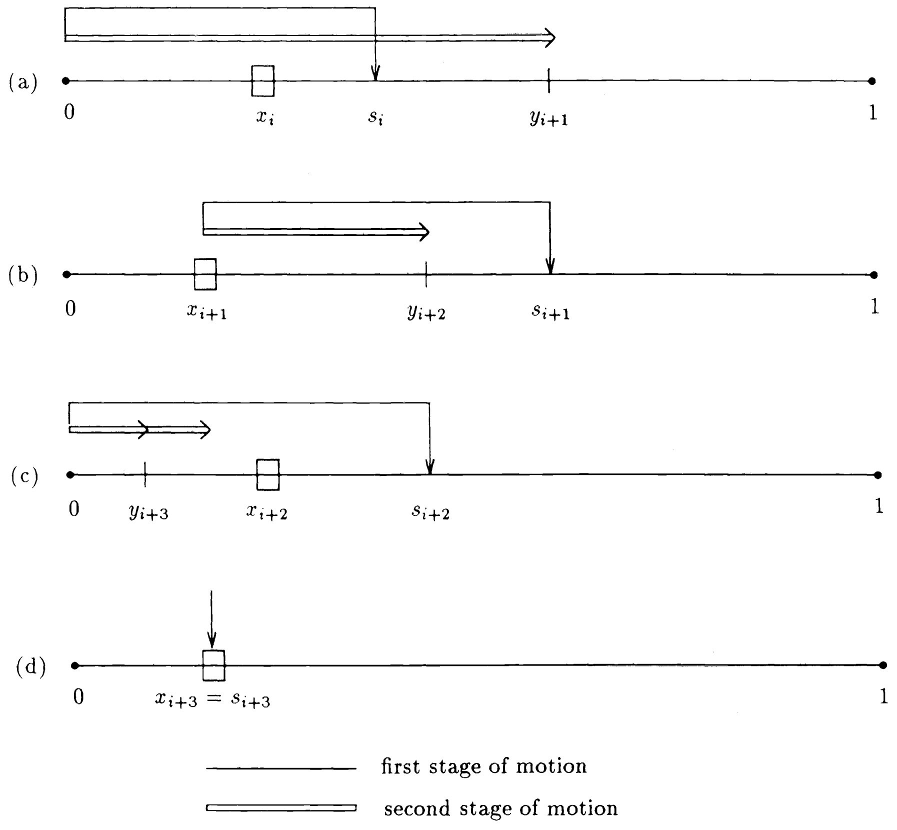


FIG. 1. *An example of decisions under policies in $\mathcal{C}$.*

Let x_i, s_i denote the respective locations of the checkpoint and the server after serving the ith request at $y_i, 1 \leq i \leq n; x_0, s_0$ denote the initial checkpoint and server positions. Figure 1 illustrates the various decisions of policies in $\mathcal{C}$. The first and second stages of the motion undergone by the server at s_i, in order to satisfy the next request at y_{i+1}, are indicated by single and double lines, respectively. In Fig. 1(b) the server at s_{i+1} moves during the first stage to x_{i+1}, which is the nearest reset point to the left of y_{i+2}. During the second stage, the server moves the checkpoint to x_{i+2}, where $x_{i+1} \leq x_{i+2} \leq y_{i+2}$, and then moves on to y_{i+2}, which becomes s_{i+2}. Such a decision is called *conservative* because it incurs a (locally) minimum cost in serving a request. In Fig. 1(a) the first stage of the server motion is nonconservative, since the server resets at 0, even though x_i is the nearest reset point to the left of y_{i+1}. In Fig. 1(c) the second stage of the server motion is nonconservative, since the checkpoint is relocated to the right of y_{i+3}. The two arrowheads in Fig. 1(c) indicate that the server first satisfies the request at y_{i+3} and then places the checkpoint at x_{i+3}, which becomes s_{i+3}.

The rate r has no effect on our optimization problem, so hereafter we assume $r = 1$ and identify cost with distance. Because the y_i are independently and identically distributed and the reset motion is cost free, the only state variable of interest is the position x_i of the checkpoint. Hereafter, this position is called the *state*. To simplify terminology we will often say "serve y_i" to mean "serve the request located at y_i."

A little reflection leads one to expect that, among the optimal policies in $\mathcal{C}$, there is one that is conservative, i.e., one that makes only conservative decisions. This is proved in §2. The proof requires some effort but the arguments are quite elementary. Section 3 begins by formulating a recurrence that defines the expected total cost $E_n(x)$ incurred by an optimal conservative policy in serving n requests starting in state x. For a broad class of distributions F, the properties of $E_n(x)$ are given in detail. Section 4 uses these properties to define an optimal policy having a simple structure. Under this policy, the sequence of states $x_1, \ldots, x_n$ forms a simple Markov chain with initial state x_0; the transient and stationary regimes of this chain are also described in §4. Section 5 studies the class of distributions F to which the optimal policy applies. The remainder of this section briefly discusses background and applications.

This paper was motivated by recent work of Bern et al. [1], who describe several applications of the moving-server checkpointing model. Here, we give only the flavor of these applications; detailed discussions can be found in [1]. Briefly, in the data base application the interval [0, 1] is taken as a normalized, continuous approximation to a large string of text, say, an encyclopedia, in compressed form. To access an item at a known location, say, y, in this compressed text, the compression statistics at the time the item was stored must be known to the request server [6]. Without checkpoints this requires the statistics to be recomputed from scratch, from 0 to the point y. Checkpoints storing the statistics generated at various points in the compressed text allow for faster access; the recomputation of statistics needs to be started only at the most recent checkpoint created before the item at y was stored.

In software testing and development, the interval approximates a long sequence of statements or lines of code. Checkpoints are placed at various points in the code in order to save the state of the software system during trial runs. After trial runs, the programmer may wish to track certain state variables beyond some point y in the code. To do so efficiently, the programmer steps through the code starting in the state saved by the last checkpoint encountered before reaching point y.

In general, the model applies to any irreversible process or computation whose states at various times need to be recaptured in order to modify the behavior or structure of the process. Physical simulations are another application of this type brought out by Bern et al. [1]. Note that the models here and in [1] apply to those situations where requests at points earlier than (to the left of) a checkpoint cause the checkpoint to be moved. This applies in those cases where a checkpoint to the right of a request becomes invalid. For example, the program is being modified in the program debugging application, or the compressed text is being modified in the data base application.

Bern et al. consider a more general model in which there are $k > 1$ checkpoints. Their primary goal is a combinatorial analysis of competitiveness, i.e., the best relative performance of on-line algorithms, such as those studied here, versus off-line algorithms that can see all requests in advance. The objective here is a more realistic stochastic analysis of optimal on-line algorithms. As a concession to the greater difficulty of a stochastic model, we limit ourselves to an important special case treated by Bern et al., viz., the case $k = 1$ of only one checkpoint.

A great deal of work has also been published on the use of checkpoints in the design of fault-tolerant systems. The references in [2], [5] provide an excellent gateway to this literature. Most of this research focuses on queueing models defined by given interarrival and running time distributions. System failures occur according to a given probability law. Recovery from failures includes rolling the system back to the most recent checkpoint (error-free state). There are also several papers on checkpointing within the setting of the results presented here, i.e., optimal stochastic scheduling of checkpoints for fault-tolerant computations [2]–[4], [7]. However, all of these models concern the placement of checkpoints at fixed locations; the moving checkpoint of the model studied here leads to fundamental changes in the analysis.

2. A reduction.

The following result then simplifies the policies that need to be considered.

THEOREM 2.1. *There is a conservative policy among the optimal policies in $\mathcal{C}$.*

Proof. We prove that, for any initial state x_0 and sample $y_1, \dots, y_n$, there is a sequence of conservative decisions whose total cost is minimum over all possible decision sequences of policies in $\mathcal{C}$. It follows that any policy $\mathcal{P}$ may be replaced by a conservative policy $\mathcal{P}'$ whose expected total cost is less than or equal to that of $\mathcal{P}$. In §3, we conclude from the Bellman equation that there exists an optimal policy among the conservative ones. By the previous remark this policy is then optimal among all policies.

There are two types of nonconservative decisions that must be considered, one in the first stage of motion and one in the second (see Fig. 1).

Type 1. In serving $y_i, 1 \leq i \leq n$, the server is reset to 0 even though the checkpoint is at x_{i-1} with $0 \leq x_{i-1} \leq y_i$.

Type 2. After serving y_i, the server motion is continued to the right so as to shift the checkpoint to a position at $x_i > y_i$.

Suppose policy $\mathcal{P}$ makes a type 2 decision in serving y_i, so that $x_i > y_i$. Let $\mathcal{P}'$ be a policy that makes the same decisions as $\mathcal{P}$ up through the first stage of the ith service but then proceeds as follows. In the second stage of the ith service $\mathcal{P}'$ makes the shorter move to y_i, where a new checkpoint is established. $\mathcal{P}'$ thus gains an advantage of $(x_i - y_i)$ over $\mathcal{P}$ upon completion of the ith service. If, on the $(i+1)$st request, $\mathcal{P}$ resets to 0, then so does $\mathcal{P}'$; $\mathcal{P}'$ mimics $\mathcal{P}$ thereafter and retains the advantage $(x_i - y_i)$. On the other hand, if $\mathcal{P}$ resets to x_i on the $(i+1)$st request, then $\mathcal{P}'$ resets to y_i. $\mathcal{P}'$ then begins the second stage of the $(i+1)$st service by moving to x_i, thus giving up the advantage $(x_i - y_i)$; $\mathcal{P}'$ mimics $\mathcal{P}$ thereafter. In either case the cost incurred by $\mathcal{P}'$ is at most that incurred by $\mathcal{P}$, and $\mathcal{P}'$ has made one fewer type 2 decision than $\mathcal{P}$. Repetitions of the above argument show that for given $x_0, y_1, \dots, y_n$ and policy $\mathcal{P}$, there exists a policy $\mathcal{P}'$ that makes no type 2 decision and incurs a cost a most that incurred under $\mathcal{P}$.

Next, let $\mathcal{P}$ be a policy that makes no decision of type 2 but makes a type 1 decision on serving y_i, i.e., the server resets at 0 even though $0 < x_{i-1} \leq y_i$. We may assume that in the second stage of serving y_i, $\mathcal{P}$ establishes a new checkpoint at $x_i < x_{i-1}$; otherwise, the cost under $\mathcal{P}$ can obviously be reduced by an amount x_{i-1} by resetting at x_{i-1} instead of at 0. Let $\mathcal{P}'$ be a policy that is identical to $\mathcal{P}$ up through the $(i-1)$st service but then proceeds as follows. $\mathcal{P}'$ resets to x_{i-1} during the ith service, keeping the checkpoint location at x_{i-1}. The costs of serving y_i under $\mathcal{P}$ and $\mathcal{P}'$ are given respectively by y_i and $(y_i - x_{i-1})$, so $\mathcal{P}'$ gains an advantage of x_{i-1} over $\mathcal{P}$ upon completion of ith service. If, on the $(i+1)$st request, $\mathcal{P}$ resets to 0, then so does $\mathcal{P}'$, mimicking $\mathcal{P}$ thereafter and retaining the advantage x_{i-1}. Otherwise, $\mathcal{P}$ resets to x_i, in which case $x_i \leq y_{i+1}$. As $\mathcal{P}$ makes no type 2 decision, we also have $x_i \leq x_{i+1} \leq y_{i+1}$. We now distinguish three possibilities.

(i) $x_i \leq y_{i+1} \leq x_{i-1}$. In the $(i+1)$st service, $\mathcal{P}'$ must reset to 0, thus reducing the advantage to $x_{i-1} - x_i$. $\mathcal{P}'$ then establishes a checkpoint at x_{i+1}, mimics $\mathcal{P}$ thereafter, and retains the advantage $x_{i-1} - x_i$.

(ii) $x_{i-1} \leq y_{i+1}$ *and* $x_{i+1} \geq x_{i-1}$. In the $(i+1)$st service, $\mathcal{P}'$ resets to x_{i-1}, establishes a checkpoint at x_{i+1}, and mimics $\mathcal{P}$ thereafter. At the end of the $(i+1)$st service, the advantage is increased to $x_{i-1} + (x_{i-1} - x_i) = 2x_{i-1} - x_i$, which is retained thereafter.

(iii) $x_i \le y_{i+1}$ *and* $x_i \le x_{i+1} < x_{i-1}$. In the $(i+1)$st service, $\mathcal{P}'$ resets to x_{i-1} and keeps the checkpoint there, the advantage under $\mathcal{P}'$ being increased to $2x_{i-1} - x_i$ at the end of the $(i+1)$st service. The above procedure is then repeated, replacing i by $i+1$.

The resulting policy $\mathcal{P}'$ makes no type 2 decision and one fewer type 1 decision than $\mathcal{P}$. Repetitions of the above argument show that for given $x_0, y_1, \ldots, y_n$ and policy $\mathcal{P}$, there exists a conservative policy $\mathcal{P}'$ that incurs a cost at most that incurred by $\mathcal{P}$. □

3. The Bellman equation. For convenience we assume initially that $F(0) = 0, F(1) = 1$ and that $F(x)$ is continuous and strictly increasing on [0, 1]. This section solves a Bellman equation defining the expected total cost $E_n(x)$ incurred by an optimal conservative policy serving n requests and starting in state x. By Theorem 2.1 the expression for $E_n(x)$ will apply to any optimal policy in $\mathcal{C}$.

LEMMA 3.1. *For* $n \ge 0$ *and* $0 \le x \le 1$

$$(3.1)\quad E_{n+1}(x) = \int_0^x \left[y + \inf_{0\le z\le y} E_n(z) \right] dF(y) + \int_x^1 \left[y - x + \inf_{x\le z\le y} E_n(x) \right] dF(y),$$

where $E_0(x) = 0, 0 \le x \le 1$.

Proof. If the first of the $n+1$ requests is at $y = y_1 < x$, then the conservative server resets to 0 before moving to y. If the checkpoint is repositioned at z, $0 \le z \le y$, then the optimal total expected cost of serving the $n+1$ requests can be expressed as $y + E_n(z)$. On the other hand, if $y \ge x$, then the conservative server resets to x, so the optimal total expected cost can be expressed as $y - x + E_n(z)$, where z, $x \le z \le y$, is again the point at which the checkpoint has been repositioned. Minimizing over z and averaging over y yields (3.1). □

Let $\mu = \int_0^1 u\,dF(u)$ denote the mean of F and define $G(u) = u[1 - F(u)]$. Then by integrating the first terms of the integrands in (3.1), we can write

$$(3.2)\quad E_{n+1}(x) = \mu - G(x) + \int_0^x \inf_{0\le z\le y} E_n(z)\,dF(y) + \int_x^1 \inf_{x\le z\le y} E_n(z)\,dF(y).$$

To obtain a simple optimal policy, the form of (3.2) suggests that it might be useful to ensure that $E_n(x)$ is unimodal with a unique minimum. To this end, consider the following two conditions on F:

C_1. $G(x)$ is unimodal on [0, 1] with a unique maximum at $\tau, 0 < \tau < 1$.

C_2. $H(x) = [G(\tau) - G(x)]/F(x)$ is increasing in $[\tau, 1]$.

C_1 does not imply C_2, as shown in §5.

THEOREM 3.1. C_1 *and* C_2 *hold jointly if and only if, for* $n \ge 1$, $E_n(x)$ *is unimodal on* [0, 1] *with a unique minimum at* τ. $E_n(x)$ *is given by*

$$(3.3)\quad E_n(x) = \mu - G(x) + (n-1)\alpha, \qquad 0 \le x \le \tau,$$

$$(3.4)\quad E_n(x) = E_n(\tau) + \frac{G(\tau) - G(x)}{F(x)}\{1 - [1 - F(x)]^n\}, \qquad \tau \le x \le 1,$$

where

$$(3.5)\quad \alpha = \int_0^\tau [\mu - G(y)]\,dF(y) + [\mu - G(\tau)][1 - F(\tau)].$$

Proof. Suppose that for $n \ge 1$, $E_n(x)$ is unimodal in [0, 1] with a unique minimum at $\tau, 0 < \tau < 1$. We refer to this as the unimodality condition. We first prove, by induction on $n \geq 1$, that this condition implies that the E_n's are given by formulas (3.3), (3.4) and then complete the proof by showing that the unimodality condition is equivalent to C_1, C_2.

Because $E_0(x) = 0$, we obtain from (3.2) that $E_1(x) = \mu - G(x)$, which coincides with (3.3), (3.4) for $n = 1$. Suppose we are given that $E_n(x)$ satisfies (3.3), (3.4) in addition to the unimodality condition. We show that $E_{n+1}(x)$ is given by (3.3), (3.4), with n replaced by $n+1$.

Let $0 \leq x \leq \tau$. Then $\inf_{0 \leq z \leq y} E_n(z) = E_n(y), 0 \leq y \leq x$, and for $x \leq y \leq 1$.

$$\inf_{x \leq z \leq y} E_n(z) = \begin{cases} E_n(y), & x \leq y \leq \tau, \\ E_n(\tau), & \tau \leq y \leq 1. \end{cases}$$

We conclude from (3.2) that

$$(3.6)\quad E_{n+1}(x) = \mu - G(x) + \int_0^\tau E_n(y)\, dF(y) + E_n(\tau)[1 - F(\tau)], \qquad 0 \leq x \leq \tau.$$

Substituting (3.3) into (3.6) both for $E_n(y)$ and $E_n(\tau)$ and taking into account (3.5), we obtain (3.3) with n replaced by $n+1$.

Let $\tau \leq x \leq 1$. We have for $0 \leq y \leq x$

$$\inf_{x \leq z \leq y} E_n(z) = \begin{cases} E_n(y), & 0 \leq y \leq \tau, \\ E_n(\tau), & \tau \leq y \leq x, \end{cases}$$

and for $x \leq y \leq 1$, $\inf_{x \leq z \leq y} E_n(z) = E_n(x)$. We conclude from (3.2) that

$$(3.7)\quad E_{n+1}(x) = \mu - G(x) + \int_0^\tau E_n(y)dF(y) + E_n(\tau)[F(x) - F(\tau)] + E_n(x)[1 - F(x)],$$

for $\tau \leq x \leq 1$. In (3.7) let $x = \tau$. Subtract the resulting equation from (3.7) to obtain

$$(3.8)\quad E_{n+1}(x) - E_{n+1}(\tau) = G(\tau) - G(x) + [1 - F(x)][E_n(x) - E_n(\tau)], \qquad \tau \leq x \leq 1.$$

Substituting for $E_n(x) - E_n(\tau)$ in (3.8) from (3.4), we obtain (3.4) with n replaced by $n+1$.

Next, we show that C_1, C_2 imply the unimodality condition. We proceed by induction. For $n = 1$, $E_1(x) = \mu - G(x)$, so C_1 implies that $E_1(x)$ satisfies the unimodality condition. Suppose that $E_1(x), \ldots, E_{n-1}(x)$ satisfy the unimodality condition. Our earlier argument shows that $E_n(x)$ satisfies (3.3), (3.4). From (3.3) and C_1, we obtain that $E_n(x)$ is decreasing on $[0, \tau]$ with a unique minimum at τ. The function $1 - [1 - F(x)]^n$ is strictly increasing on $[0, 1]$. Hence from (3.4) and C_2, $E_n(x)$ is increasing on $[\tau, 1]$ with a unique minimum at τ. Thus $E_n(x)$ satisfies the unimodality condition.

Finally, we show that the unimodality condition implies C_1, C_2. Since $E_1(x) = \mu - G(x)$, we obtain C_1. Suppose C_2 fails, i.e., $H(x_1) > H(x_2)$ for some $\tau \leq x_1 < x_2 \leq 1$. The unimodality condition implies that $E_n(x), n \geq 1$, satisfies (3.4). Because $\lim_{n \to \infty}\{1 - [1 - F(x)]^n\} = 0$ uniformly on $[\tau, 1]$, we obtain from (3.4) that, for n sufficiently large, $E_n(x_1) > E_n(x_2)$, a contradiction. Hence C_2 also holds. □

4. The optimal policy. Let $\mathcal{P}_o$ denote the optimal conservative policy, and let $x_0, x_1, \ldots$ denote the sequence of checkpoint locations under $\mathcal{P}_o$ with the initial state x_0 given. Assume that $F(x)$ satisfies conditions C_1 and C_2. By (3.2) $\mathcal{P}_o$ is defined by the case analysis determining z so as to minimize $\inf_{0\le z\le y} E_n(z)$ and $\inf_{x\le z\le y} E_n(z)$ in the proof of Theorem 3.1. Since $E_n(x)$ is unimodal with a unique minimum at τ, $\mathcal{P}_o$ always moves the checkpoint as close as possible to τ, the uniquely best state from which to start serving a request. Formally, the corollary below follows immediately from Theorem 3.1.

COROLLARY 4.1. *For each $i \ge 1$, the next-state decisions under $\mathcal{P}_o$ are: If $0 \le x_{i-1} \le \tau$, move the checkpoint to*

$$x_i = \begin{cases} y_i, & \textit{if } y_i < \tau, \\ \tau, & \textit{if } y_i \ge \tau, \end{cases} \tag{4.1}$$

and if $\tau < x_{i-1} \le 1$, move the checkpoint to

$$x_i = \begin{cases} y_i, & \textit{if } y_i < \tau, \\ \tau, & \textit{if } \tau \le y_i \le x_{i-1}, \\ x_{i-1}, & \textit{if } x_{i-1} \le y_i \le 1. \end{cases} \tag{4.2}$$

Under $\mathcal{P}_o$ the checkpoint locations form a Markov chain. For the translations from $x_{i-1} = x$ to $x_i = x'$ the transition probabilities follow from Corollary 4.1:

$$\begin{aligned} 0 \le x \le \tau: \quad & P(x, dx') = dF(x'), \qquad 0 \le x' < x \\ & P(x,\tau) = 1 - F(\tau) \end{aligned}$$

$$\begin{aligned} \tau < x \le 1: \quad & P(x, dx') = dF(x'), \qquad 0 \le x' < \tau \\ & P(x,\tau) = F(x) - F(\tau) \\ & P(x,x) = 1 - F(x). \end{aligned}$$

If the initial state satisfies $0 \le x_0 \le \tau$, then by (4.1) all subsequent states also satisfy $0 \le x_i \le \tau$, $i \ge 1$. Otherwise, if $\tau < x_0 \le 1$, subsequent states remain at x_0 until the first request y_j, $j \ge 1$, such that $0 \le y_j < x_0$. From that point onward, $0 \le x_i \le \tau$, $i \ge j$.

Thus, the transient states of $\{x_i\}$ are those in $(\tau, 1]$ and the recurrent states are those in $[0, \tau]$. The equilibrium measure of $\{x_i\}$ is on $[0, \tau]$ and defined by

$$\begin{aligned} dP(y) &= dF(y), \qquad 0 \le y < \tau, \\ P(\tau) &= 1 - F(\tau). \end{aligned} \tag{4.3}$$

It can be seen that C_1 and C_2 were introduced to control the transient behavior of $\mathcal{P}_o$ when started in an initial state $x_0 \in (\tau, 1]$. The result below shows that if initial states are suitably restricted, then an optimal policy can be defined for a broader class of distributions F. First, consider the following relaxation of C_1.

C_0. For some τ, $0 < \tau < 1$, $G(x)$ has a unique maximum at τ on [0, 1], and $G(x)$ is increasing on $[0, \tau]$.

COROLLARY 4.2. *Of C_0 holds, then for $n \ge 1$,* (3.3) *holds and $E_n(x)$ has a unique minimum at τ on $[\tau, 1]$.*

Proof. The proof parallels that of Theorem 3.1, proceeding by induction on $n \ge 1$. As $E_1(x) = \mu - G(x)$, the result holds for $n = 1$. Suppose it holds for $n \ge 1$. Formula (3.3) remains valid for $0 \le x \le \tau$. For $\tau \le x \le 1$, we obtain from (3.2)

(4.4)
$$E_{n+1}(x) = \mu - G(x) + \int_0^\tau E_n(y)\,dF(y) + E_n(\tau)[F(x) - F(\tau)] + \int_x^1 \inf_{x\le z\le y} E_n(z)dF(y),$$

which readily implies $E_{n+1}(x) > E_{n+1}(\tau)$, $\tau < x \le 1$. Then (3.6) and (4.4) imply the corollary for $n+1$. □

If $x_0 \in [0, \tau]$, with τ defined by C_0, then by Corollary 4.2 $\mathcal{P}_o$ is completely defined by (4.1). Hence, if C_0 holds and the initial state x_0 is a sample from the equilibrium measure (4.3), then $\mathcal{P}_o$ makes the decisions in (4.1), and $E_n(x_0) = n\alpha$, $n \ge 1$, with α given by (3.5)

5. Examples. First, we verify that there is no redundancy in the conditions C_1 and C_2 of Theorem 3.1.

THEOREM 5.1. *There exist distributions for which C_1 holds and C_2 fails. Also, there exist distributions for which C_0 holds but C_1 and C_2 fail.*

Proof. We only prove the first assertion; the second assertion can be proved by a similar approach.

The following definitions simplify the search for examples. Let $p(x)$ be a continuous nonnegative function on [0, 1) with $\int_0^1 p(x)dx = \infty$. Define $\phi(x) = \exp(-\int_0^x p(t)dt)$, $0 \le x \le 1$, so that $F(x) = 1 - \phi(x)$ is a probability distribution on [0, 1]. On [0, 1] we have $G(x) = x\phi(x)$ and on [0, 1) $G'(x) = \phi(x)(1 - xp(x))$, so C_1 holds if

(5.1)
$$\begin{aligned} xp(x) &< 1, \qquad 0 < x < \tau, \\ xp(x) &> 1, \qquad \tau < x < 1. \end{aligned}$$

Now differentiating the function $H(x)$ defined by C_2 gives

$$F^2(x)H'(x) = -\phi(x) - x\phi'(x) + \phi^2(x) + \tau\phi(\tau)\phi'(x),$$

so C_2 holds only if $h(x) = -1 + xp(x) + \phi(x) - \tau\phi(\tau)p(x) \ge 0, \tau < x < 1$.

Thus, let us choose a $p(x)$ as defined above so that, for given $0 < \tau < \theta < 1$, $p(x)$ satisfies (5.1), $\theta p(\theta) < 1 + \tau\phi(\tau)/4$, and $\int_0^\theta p(t)dt > -ln(\tau\phi(\tau)/4)$. Then C_1 holds but C_2 fails, since

$$h(\theta) = [\theta p(\theta) - 1] + \phi(\theta) - \tau\phi(\tau)p(\theta) < \frac{\tau\phi(\tau)}{4} + \frac{\tau\phi(\tau)}{4} - \tau\phi(\tau) < 0.$$

Figure 2 sketches a general example, where the condition $\int_0^\theta p(x)dx > -\ln(\tau\phi(\tau)/4)$ is guaranteed by defining $p(x)$ so that it has a sufficiently large hump in (τ, θ). □

Although C_1 and C_2 were needed to establish the unimodality of $E_n(x)$, they are otherwise somewhat recondite properties of distribution functions. On the other hand, they embrace a wide class of interesting distributions, as the examples below illustrate. Recall our assumptions that $F(0) = 0$, $F(1) = 1$, and $F(x)$ is continuous and strictly increasing in [0, 1].

(i) Consider the convex distributions.

THEOREM 5.2. *Suppose F has a nonnegative second derivative, $F''(x) \ge 0, 0 \le x \le 1$. Then C_1 and C_2 hold.*

Proof. From $G(x) = x[1 - F(x)]$ we have $G(0) = G(1) = 0$ and

(5.2)
$$G'(x) = 1 - F(x) - xF'(x), \qquad G''(x) = -2F'(x) - xF''(x).$$

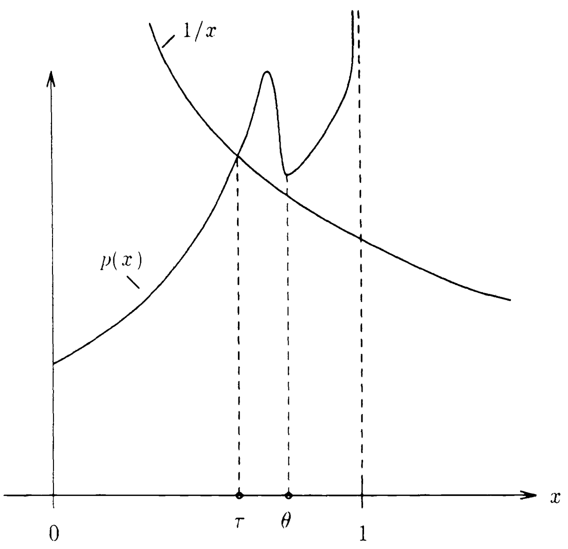


FIG. 2. *An example where C_1 holds but C_2 fails for* $F(x) = 1 - \exp\left(-\int_0^x p(t)dt\right), 0 \le x \le 1$.

The assumptions on $F(x)$ readily imply that $F'(x) > 0$ on (0, 1]. We conclude from (5.2) that $G''(x) < 0$ for $0 < x \le 1$, so $G(x)$ is strictly concave in [0, 1]. It follows that $G(x)$ is unimodal in [0, 1] with a unique maximum at the solution of

$$G'(\tau) = 1 - F(\tau) - \tau F'(\tau) = 0. \tag{5.3}$$

To show that $H(x)$ is increasing in $\tau \le x \le 1$, differentiate and get

$$F^2(x)H'(x) = K(x), \tag{5.4}$$

where

$$K(x) = [x - \tau + \tau F(\tau)]F'(\tau) - F(x)[1 - F(x)]. \tag{5.5}$$

Differentiating $K(x)$ gives

$$K'(x) = [x - \tau + \tau F(\tau)]F''(x) + 2F(x)F'(x). \tag{5.6}$$

From (5.3) and (5.5) we get $K(\tau) = 0$, and from (5.6) and $F''(x) \ge 0$ we get $K'(x) > 0$ for $\tau \le x \le 1$. Then $K(x)$ and $H'(x)$ are strictly positive in $\tau < x \le 1$. We conclude that $H(x)$ is increasing in $\tau \le x \le 1$. □

(ii) The convex property in Theorem 5.2 is not necessary for C_1 and C_2. Indeed, C_1 and C_2 also hold for the following useful concave distributions.

THEOREM 5.3. *Let $F(x) = x^\alpha, 0 \le x \le 1$, where $0 < \alpha < 1$. Then C_1 and C_2 hold.*

Proof. We have $F''(x) = \alpha(\alpha - 1)x^{\alpha-2} < 0$, $0 < x \le 1$, so $G''(x) = -(\alpha + \alpha^2)x^{\alpha-1} < 0$, $0 < x \le 1$. Then $G(x)$ is unimodal with a unique maximum in [0, 1], as in Theorem 5.2.

With $K(x)$ as defined in (5.5), we get

$$K'(x) = \alpha x^{\alpha-2}L(x),$$

where

$$L(x) = (\alpha - 1)(x - \tau + \tau^{\alpha+1}) + 2x^{\alpha+1}.$$

From (5.3), we get $(\alpha+1)\tau^{\alpha}=1$, so that

$$L'(x)=\alpha-1+2(\alpha+1)x^{\alpha}\geq\alpha-1+2(\alpha+1)\tau^{\alpha}=\alpha+1,\ x>\tau.$$

Because $L(\tau)=\tau>0$, we obtain that $L(x)$ and $K'(x)$ are strictly positive for $x>\tau$. As in the proof of Theorem 5.2, we conclude that $H(x)$ is increasing for $\tau\leq x\leq 1$. □

(iii) Symmetry of F does not in general imply C_1 and C_2. However, C_1 and C_2 *are* implied by the symmetric triangular distribution on [0, 1]. This distribution has the density

$$F'(x)=\begin{cases} 4x, & 0\leq x\leq \frac{1}{2},\\ 4-4x, & \frac{1}{2}\leq x\leq 1.\end{cases}$$

Routine calculations reveal that

$$G'(x)=\begin{cases} 1-6x^2, & 0\leq x\leq \frac{1}{2},\\ 2(x-1)(3x-1), & \frac{1}{2}\leq x\leq 1,\end{cases}$$

so that $G(x)$ is unimodal with a unique maximum at $\tau=1/\sqrt{6}$. Hence C_1 holds. From (5.5), (5.6) we obtain $K(\tau)=0, K(1)>\tau, K'(x)>0$ on $(\tau,\frac{1}{2}]$ and $K'(x)<0$ on $(\frac{1}{2},1]$. Then $K>0$ and, hence, $H'=K/F^2>0$ on $(\tau,1)$. It follows that C_2 also holds.

(iv) Further examples are furnished by the two-parameter family of concave, piecewise linear distribution functions

$$F(x)=\begin{cases} \alpha x, & 0\leq x\leq\beta,\\ \frac{1-\alpha\beta}{1-\beta}+\frac{\beta(\alpha-1)}{1-\beta}, & \beta\leq x\leq 1,\end{cases}$$

where $\alpha>1, 0<\beta<1/\alpha$. It is easy to see that condition C_1 is violated if $1/(2\alpha)<\beta<\frac{1}{2}$ and is satisfied otherwise. If $\beta\leq 1/(2\alpha)$, then $\tau=\frac{1}{2}$, and if $\beta\geq\frac{1}{2}$, then $\tau=1/(2\alpha)$. An analysis of $K(x)$ in (5.5) shows that if either $\beta\leq 1/(2\alpha)$ or $\beta\geq\frac{1}{2}$, then $K(x)>0$, $\tau\leq x\leq 1$. Hence, $H(x)$ is increasing on $[\tau,1]$ and C_2 holds.

REFERENCES

[1] M. Bern, D. H. Greene, A. Raghunathan, and M. Sudan, *On-line algorithms for locating checkpoints*, in Proc. 22nd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1990, pp. 359–368.

[2] L. B. Boguslavsky, E. G. Coffman, Jr., E. N. Gilbert, and A. Y. Kreinin, *Scheduling checks and saves*, ORSA J. Comput., to appear.

[3] E. G. Coffman, L. Flatto, and A. Y. Kreinin, *Scheduling saves in fault-tolerant computations*, AT&T Bell Laboratories, Murray Hill, NJ, to appear.

[4] E. G. Coffman, Jr. and E. N. Gilbert, *Optimal strategies for scheduling saves and preventive maintenance*, IEEE Trans. Reliab., 39 (1990), pp. 9–18.

[5] V. G. Kulkarni, V. F. Nicola, and K. S. Trivedi, *Effects of checkpointing and queueing on program performance*, Stochastic Models, 6 (1990), pp. 615–648.

[6] J. Storer, *Data Compression*, Computer Science Press, Rockville, MD, 1988.

[7] S. Toueg and O. Babaoglu, *On the optimum checkpoint selection problem*, SIAM J. Comput., 13 (1984), pp. 630–649.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 661–683, August 1993

REGULAR RESOLUTION VERSUS UNRESTRICTED RESOLUTION*

ANDREAS GOERDT†

Abstract. A resolution proof of an unsatisfiable propositional formula is called regular if and only if no variable is eliminated (with the resolution rule) twice on any branch of the proof tree representing the resolution proof. An infinite family of unsatisfiable propositional formulas is constructed and the following shown: These formulas have polynomial size unrestricted resolution proofs, whereas all regular resolution proofs of these formulas are of superpolynomial length.

Key words. resolution theorem proving, regular resolution, propositional logic

AMS subject classifications. 68T15, 03B35

Introduction. The resolution proof rule is a basic principle of many implementations of inference mechanisms (e.g., in logic programming). But mostly not resolution itself is implemented, but a restriction (or refinement) of resolution. There are a variety of restrictions of resolution that are used in theorem proving algorithms. See [13] for many articles in this direction. A more concise account of resolution restrictions is [12, p. 103ff]. The idea of these restrictions is to reduce the search space necessary for a deterministic implementation of the nondeterministic resolution rule. Little theoretical work on the advantages and disadvantages of these restrictions has been done. More typical in this area are experimental results as they can be found, for example, in [14]. The purpose of this paper is to contribute to a theoretical analysis of these restrictions.

We analyze the regular resolution restriction in the context of propositional logic and show that the price to be paid for the reduction of the search space induced by this restriction is that proofs get substantially (i.e., superpolynomially) longer. The question of whether resolution restrictions make proofs longer is mentioned as virtually unexplored in [12, p. 106]. This question for regular resolution in particular is mentioned several times in the literature [2, p. 292], [15, p. 470], [6, p. 45], [11, p. 236]. A first (rather weak) result of the type we are interested in here is proved in [6, Prop. 3.2.1]. In [7], [8] we obtained such results for two other resolution restrictions: N-resolution and Davis–Putnam resolution. In N-resolution one clause to which the resolution rule is applied only consists of negative literals. Davis–Putnam resolution (as we defined it following [6]) is a further restriction of regular resolution. Hence, here we improve the result from [8]. The Davis–Putnam procedure (in the sense of [9, p. 53]) to test the satisfiability of a propositional formula employs a strategy similar to regular resolution. In [19] a single propositional tautology is presented whose minimal resolution proof is not regular. In [18] it is shown that the regularity restriction makes proofs in cut-free Gentzen Systems superpolynomially longer. The formulas used there cannot be used to show the same for resolution.

Though there are not many papers concerned with a complexity analysis of resolution restrictions in particular, there is a wide range of literature dealing with the complexity of propositional proof system in general. In [5] several propositional proof systems (resolution, systems with modus ponens (so-called Frege systems), and so on) are compared with respect to lengths of shortest proof. In that paper the basic notion of comparing proof systems is introduced: System A p-simulates system B if and only if any proof in B can be transformed into an at most polynomially longer proof in A. In this terminology we show that regular resolution does not p-simulate unrestricted resolution. A challenging question in this area is:

*Received by the editors March 7, 1990; accepted for publication (in revised form) April 24, 1991.

†Universität-GH-Paderborn, Fachbereich 17, Mathematik-Informatik, Warburges Straße 100, DW 4790 Paderborn, Germany.

Given a propositional proof system, find an infinite family of propositional formulas provable in this system, whose proofs are not polynomially bounded. Such proof systems are called not polynomially bounded. As reasonable proof systems are just nondeterministic algorithms for the *co*NP-complete language of propositional tautologies, the NP $\neq$ *co*NP assumption implies that any proof system is not polynomially bounded. The point is to show this property for a given propositional proof system without assuming NP $\neq$ *co*NP. Resolution was shown not polynomially bounded only in 1985 by Haken [10]. Before Haken's paper there were several attempts at this question. In [6], [15] it is shown that regular resolution is not polynomially bounded. In [16] Urquhart showed that the formulas used by Tseitin/Galil in [15], [6] have only superpolyomial unrestricted resolution proofs, too. (Hence, our result does not follow from Galil's and Tseitin's result.) To obtain his result Haken introduced a new technique, which was subsequently applied in [16], [17], [18], [4], [3]. Ajtai [1] introduces new techniques to show that bounded depth Frege systems are not polynomially bounded. Bounded depth Frege systems are better systems with respect to proof length rather than resolution. It is not known whether Frege proof systems are polynomially bounded.

In this paper we extend the technique of Haken in such a way that we can prove a superpolynomial lower bound on the length of regular resolution proofs of formulas whose unrestricted resolution proofs are polynomially bounded. To do this, we have to isolate a property peculiar to regular resolution, which allows us to make use of a suitably modified version of Haken's technique. The formulas we use in this paper are extensions of the formulas in [7].

Section 1 contains basic definitions and lemmas and an explanation of the idea underlying our formulas. In §2 we define the family of formulas and present their polynomial size unrestricted resolution proof. In §3 we prepare the proof from §4. In §4 we prove the superpolynomial lower bound on the length of the regular resolution proofs of our formulas.

1. Preliminaries. By $\log n$ we always mean $\log_2(n)$, $\exp(n) = 2^n$, hence $\exp(\log n) = n$. Card M is the cardinality of the set M.

Let f be a partial mapping from M to N; we write $f : M \dashrightarrow N$. Dom f is the domain of f, that is, Dom $f = \{x \in M | f(x) \text{ is defined}\}$.

For $L \subseteq M$ we let $f(L) = f(L \cap \text{Dom } f)$ and $f|_L : L \dashrightarrow N$, the restriction of f to L. For $x \in M$, $y \in N$, $f[y/x] : M \dashrightarrow N$ is given by

$$f[y/x](z) = \begin{cases} y & \text{if } z = x, \\ f(z) & \text{otherwise.} \end{cases}$$

A partial mapping $g : M \dashrightarrow N$ is an extension of f if Dom $g \supseteq$ Dom f and $g(x) = f(x)$ for all $x \in$ Dom f. In writing $f : M \to N$ we mean that $f : M \dashrightarrow N$ and Dom $f = M$. In this case f is called total.

A subset $R \subseteq M \times N$ is a relation. For $x \in M$, $y \in N$ we write $R(x, y)$ instead of $(x, y) \in R$. We define $R^{-1} \subseteq N \times M$ by $R^{-1}(y, x)$ if and only if $R(x, y)$ and $R(x) = \{y | R(x, y)\}$.

Let Var be a set of propositional variables. The set of literals over Var consists of positive and negative literals, x and $\overline{x}$ with $x \in$ Var. A clause over Var is a finite set of literals over Var, a formula over Var a finite set of clauses over Var. Sometimes instead of $\{L_1, \dots, L_n\}$ we write $L_1 \dots L_n$ for a clause. The length of a formula is the number of clauses in it. A partial truth value assignment π of a set of variables Var is a partial mapping π : Var $\dashrightarrow \{0, 1\}$ (0 for false, 1 for true). A truth value assignment of Var is a total mapping π : Var $\to \{0, 1\}$.

A truth value assignment π of Var can be extended to the set of literals over Var by setting

$$\pi(\overline{x}) = \begin{cases} 1 & \text{if } \pi(x) = 0, \\ 0 & \text{if } \pi(x) = 1. \end{cases}$$

Let C be a clause over Var, π an assignment of Var, then $\pi \models C$, C is valid under π or π satisfies C if and only if there is a literal $L \in C$ with $\pi(L) = 1$. Let F be a formula over Var. We write $\pi \models F$; π satisfies F or F is valid under π if and only if $\pi \models C$ for all clauses C in F. Our formulas are interpreted as being in conjunctive normal form. A clause C is a tautology if and only if both $x \in C$ and $\overline{x} \in C$ for a variable x.

The resolution rule reads

$$\frac{C, x \qquad D, \overline{x}}{C, D},$$

where C, x stands for $C \uplus x$, $D, \overline{x}$ for $D \uplus \overline{x}$, and C, D for $C \cup D$. ($\uplus$ is the disjoint union, i.e., $x \notin C, \overline{x} \notin D$.) C, D is called a resolvent of C, x and $D, \overline{x}$. We say that x is eliminated in going from C, x and $D, \overline{x}$ to C, D.

A resolution proof of the clause C from the formula F is a sequence of clauses $D_1 D_2 \ldots D_n$ with $D_n = C$, and for all i either $D_i \in F$ or there are $j, k < i$ such that D_i is obtained by applying the resolution rule to D_j and D_k, i.e., D_i is a resolvent of D_j and D_k. A resolution proof of F is a resolution proof of [ ], the empty clause from F. The length of a resolution proof is the number of clauses in it. We assume that the clauses in a resolution proof are pairwise distinct. Resolution is sound and complete, i.e., F is unsatisfiable if and only if there exists a resolution proof of F. We could also measure the length of formulas and proofs by counting the number of characters needed to write them down. With this measure the results of this paper hold, too: We consider only formulas where the number of clauses is an upper bound on the number of variables. Assume we are given such a formula with m clauses. Each clause of this formula and of a resolution proof with this formula needs at most $C \cdot 2m^2$ characters. Hence the length of the formula (resolution proof) in characters is polynomially related to the number of clauses in the formula (resolution proof).

We visualize resolution proofs as trees: An elimination of x by the resolution rule is denoted by Fig. 1. Sometimes we omit the labels on the edges. The number of nodes of the tree can be exponentially larger than the length of the resolution proof in linear form. The proof tree need not be unique. We always choose one of the possible proof trees. Clauses above a clause C in the tree precede C in the linear representation of the proof. An initial piece of a path in a proof is a sequence of clauses $C_1 \ldots C_m$ such that C_{i-1} is a son of C_i and $C_1 \in F$. A path in a proof of a clause C from a formula F is a path in the proof, whose first clause is a leaf and whose last clause is C, the root of the proof tree.

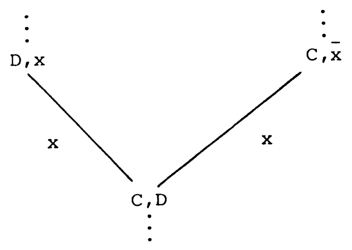


FIG. 1.

A regular resolution proof of a formula F is a resolution proof of F that can be represented by a tree with the following property: For each path of the proof, the sequence of variables occurring as labels on the edges connecting the clauses of this path in the proof tree does not contain the same variable twice. That is, no variable is eliminated more than once on any path of the proof tree. See [15, p. 472] for a remark concerning the idea underlying regular resolution. The depth of a regular proof tree is bounded by the number of variables of the formula to be proved.

The following theorem is well known.

THEOREM 1.1. *Regular resolution is complete.*

Proof. The proof proceeds by induction on the number of variables occurring in an unsatisfiable formula F. The induction base (F has no variables) is trivial. If F contains $n > 0$ variables we choose one variable x of F and construct a new formula G that consists of all clauses (except of tautologies) of F not containing x or $\overline{x}$ plus all resolvents of clauses (except of tautologies) of F that are obtained by eliminating x. Then, G is satisfiable implies F is satisfiable and x does not occur any more in G. By induction hypothesis the claim follows. $\square$

The following lemma (cf. [10]) relates truth value assignments and resolution proofs. As the number of assignments is exponential in the number of variables it gives us a handle to obtain superpolynomial lower bounds on the length of resolution proofs.

LEMMA 1.2. *Given a resolution proof $\mathfrak{R}$ of the clause C from the formula F, let π be a truth value assignment of the variables from F with $\pi \not\models C$. There is a unique path $D_1 \dots D_m$ in the proof tree visualizing $\mathfrak{R}$ such that $D_m = C$, $D_1 \in F$ and $\pi \not\models D_j$ for all D_j on the path.*

Proof. Let x be a variable from F and π an assignment of the variables from F. For two clauses C, D with $\pi \not\models C \cup D$ we have either $\pi \not\models C \uplus \{x\}$ or $\pi \not\models D \uplus \{\overline{x}\}$ depending on whether $\pi(x) = 0$ or $\pi(x) = 1$. This gives us the required path. $\square$

It is difficult to construct infinite families formulas whose resolution proofs have certain properties. In order to get our formulas we start from the formulas considered in [7].

Convention 1.3. In the following let $N \in \mathbb{N}$ be such that $M = \log N \in \mathbb{N}$.

DEFINITION 1.4. The set of $M \cdot N$ variables Var^N is given by

$$\mathrm{Var}^N = \{ij \mid i \in \{1, \dots, M\}, j \in \{1, \dots, N\}\}.$$

Var^N is and $M \times N$-matrix of variables, as shown in Fig. 2.

$$\begin{array}{ccccc}
11 & 12 & \cdots & 1(N-1) & 1N \\
21 & 22 & \cdots & 2(N-1) & 2N \\
\vdots & \vdots & & \vdots & \\
M1 & M2 & \cdots & M(N-1) & MN
\end{array}$$

FIG. 2.

We have M rows, and write Row $i = \{i1, \dots, iN\}$. We have N columns and write Column j $= \{1j, \dots, Mj\}$.

The Row i and Column j are pairwise disjoint, for $x \in$ Var, Row x, Col x (or Column x) is the unique row, column containing x.

For $X, Y \subseteq$ Var we say: X is covered by Y if and only if $X \subseteq \cup_{y \in Y}$ Col y. We usually use this expression when $X \subseteq$ Row i, $Y \subseteq$ Row j. We say X meets Y if $X \cap Y \neq \emptyset$. Usually we use this expression when $X \subseteq$ Col i and $Y \subseteq$ Row j or vice versa.

Each row is partitioned into sections. For i, j with $1 \leq i \leq M$, $1 \leq j \leq N/2^i$ we define

$$\mathrm{Sec}\; ij = \left\{i\left(2^i \cdot (j-1) + 1\right), \dots, i\left(2^i \cdot j\right)\right\},$$

and for i, j with $1 \leq i \leq M$, $1 \leq j \leq 2 \cdot \frac{N}{2^i}$,

$$\mathrm{Halfsec}\; ij = \left\{i\left(2^{i-1} \cdot (j-1) + 1\right), \dots, i\left(2^{i-1} \cdot j\right)\right\}.$$

The Sec ij are called sections, the Halfsec ij halfsections. Note that Sec ij is the jth section of Row i, *not* the section containing the variable ij. For $x \in$ Var, Sec x is the unique section containing x; it is similar for Halfsec. (To avoid ambiguities, we do not use the notation Sec ij with a concrete variable ij.)

$$\text{Sec}\, ij = \text{Halfsec}\, i(2 \cdot j - 1) \uplus \text{Halfsec}\, i(2 \cdot j).$$

Halfsec $i(2 \cdot j - 1)$ is the left half section of Sec ij, Halfsec $i(2 \cdot j)$ the right half section. For i, j with $2 \leq i \leq M$, $1 \leq j \leq 4 \cdot \frac{N}{2^i}$ we write

$$\text{Quartersec}\, ij = \left\{i\left(2^{i-2} \cdot (j-1) + 1\right), \ldots, i\left(2^{i-2} \cdot j\right)\right\}.$$

For each half section H of Row i there is a unique section of Row $i - 1$ covering H and vice versa.

An easy induction on M shows that Var^N has $N - 1$ sections. From the definition it follows that Row i has $N/2^i$ sections, $N/2^{i-1}$ half sections, and $N/2^{i-2}$ quarter sections. Partial truth value assignments of Var^N are denoted by an $M \times N$-matrix with values 0, 1, undefined. A column C of Var^N is a 0-column of a partial truth value assignment π if and only if $\pi(C) = 0$ or $\pi(C)$ is totally undefined.

Example 1.5. Var^4 can be visualized as Fig. 3.

11	12	13	14
21	22	23	24

FIG. 3.

The additional line separates the sections. Row 1 has two sections, Row 2 only one.
Var^{16} can be visualized as Fig. 4.

11	12	13	14	15	16	17	18	19	110	111	112	113	114	115	116
21	22	23	24	25	26	27	28	29	210	211	212	213	214	215	216
31	32	33	34	35	36	37	38	39	310	311	312	313	314	315	316
41	42	43	44	45	46	47	48	49	410	411	412	413	414	415	416

FIG. 4.

DEFINITION 1.6. The formula MPHP^N (for the modified pigeonhole principle, the formulas are obtained by modifying the formulas used by Haken [10], which encode the pigeonhole principle) consists of positive and negative clauses.

The positive clauses are the columns of Var^N.

The negative clauses are all clauses $\{\overline{x}, \overline{y}\}$ with $x \in$ Halfsec $i(2j - 1)$, the left half of a section, $y \in$ Halfsec $i(2j)$ the corresponding right half, for an $i \in \{1, \ldots, M\}$, $j \in \{1, \ldots, \frac{N}{2^i}\}$.

MPHP^N has $O\left(N^3\right)$ many clauses.

COROLLARY 1.7. MPHP^N *has a resolution proof whose length is of* $O\left(N^4\right)$.

Proof. The short (i.e., polynomial) resolution proof of MPHP^N is obtained according to the pattern in Fig. 5, shown for MPHP^4. (Some clauses are missing; for details see [8].)

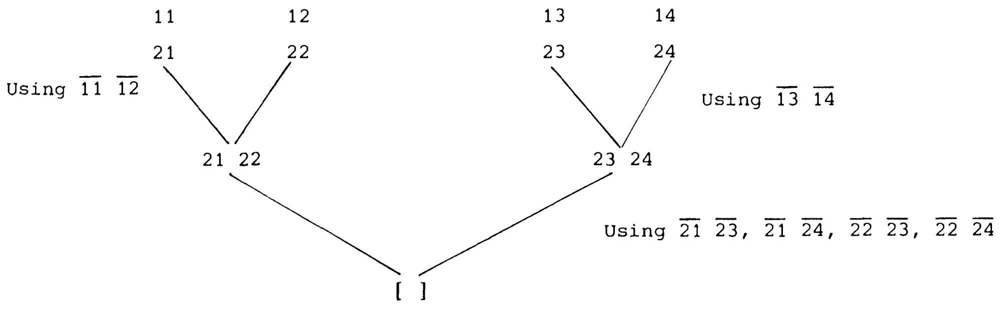

11 12 13 14
21 22 23 24
Using $\overline{11}\ \overline{12}$
Using $\overline{13}\ \overline{14}$
21 22
23 24
Using $\overline{21}\ \overline{23}$, $\overline{21}\ \overline{24}$, $\overline{22}\ \overline{23}$, $\overline{22}\ \overline{24}$
[ ]

FIG. 5.

This proof proceeds in stages $0, \dots, M$. The clauses of stage 0 are the positive clauses of MPHP^N. To generate a clause of stage $i \geq 1$, we take two clauses of stage $i-1$, eliminate the variables of Row i from these clauses, putting the rest together. To eliminate the variables of Row i from two clauses and putting the rest together we need at most $(N/2)^2 + (N/2) \leq N^2$ clauses. Let L_i be the number of clauses necessary to generate a clause of stage i. We have $L_0 = 1$ and $L_i \leq N^2 + 2 \cdot L_{i-1}$. Then we get $L_i \leq (\sum_{s=0}^{i} 2^s \cdot N^2) + 2^i$. Hence $L_M \leq N \cdot N \cdot N^2 + N$, and the length of this proof is $O\left(N^4\right)$. □

For the rest of this section we develop the ideas leading to the definition of our formulas in §2.

First we observe that a short proof of MPHP^N must essentially eliminate the variables of the positive clauses from top to bottom, i.e., Row 1 first, Row 2 second, and so on. A proof working the other way around (Row M first) is of superpolynomial length anyway. (This can be proved as in [7].) Hence we only care about this top to bottom proof and modify MPHP such that this top to bottom proof becomes long (i.e., superpolynomial) if the regularity restriction is obeyed, but stays short if irregularities are allowed. Note that the short proof of MPHP as above is regular.

In our proof of MPHP^4 above, to get from 21 22, 23 24 to [ ], we either have to make 21 22 to $\overline{23}$ and to $\overline{24}$ or 23 24 to $\overline{21}$ and to $\overline{22}$. Assume our proof uses the first alternative. We want to modify MPHP such that a proof obeying the regularity restriction has to decide very early if the 21 22 to be generated will finally become $\overline{23}$ or $\overline{24}$. We force this decision to be made early by extending the negative clause $\overline{11}\ \overline{12}$ with $\overline{23}$ and $\overline{24}$ to get two negative clauses $\overline{11}\ \overline{12}\ \overline{23}$ and $\overline{11}\ \overline{12}\ \overline{24}$; it is similar for $\overline{13}\ \overline{14}$. We get $\overline{13}\ \overline{14}\ \overline{21}$ and $\overline{13}\ \overline{14}\ \overline{22}$. The formula that we get by modifying MPHP by taking these extended negative clauses for Row 1 instead of the original negative clauses or Row 1 is satisfiable.

The assignment in Fig. 6

1	1	1	1
0	0	0	0

FIG. 6.

satisfies all clauses of this formula. To get an unsatisfiable formula we add some more negative clauses, clauses that allow us to get rid of the literals extending the negative clauses—and in addition introduce an irregularity into the short proofs of this formula. The new clauses are

$$\begin{Bmatrix} \overline{11} & \overline{12} & & \\ 21 & 22 & 23 & 24 \end{Bmatrix} \quad \text{and} \quad \begin{Bmatrix} & & \overline{13} & \overline{14} \\ 21 & 22 & 23 & 24 \end{Bmatrix}.$$

With these new clauses we get rid of the extending literals and can derive

$$\begin{matrix} \overline{11} & \overline{12} \\ 21 & 22 \end{matrix} \quad \text{and} \quad \begin{matrix} \overline{13} & \overline{14} \\ 23 & 24. \end{matrix}$$

Then we can proceed as in MPHP^4. The nice fact about the formula is that the described proof has an irregularity: In generating

$$\begin{matrix} \overline{11} & \overline{12} \\ 21 & 22 \end{matrix}$$

we have eliminated 23 and 24 and in generating

$$\begin{matrix} \overline{13} & \overline{14} \\ 23 & 24 \end{matrix}$$

we have eliminated 21 and 22. We have seen above that after generating 21 22 we eliminate 23 and 24 or the other way around.

The bad fact about this formula is that the proof can be transformed into a regular proof, which, after generalizing the above construction to arbitrary N, still is short: First we observe that either

$$\begin{matrix} \overline{11} & \overline{12} & & \\ 21 & 22 & 23 & 24 \end{matrix} \quad \text{or} \quad \begin{matrix} & & \overline{13} & \overline{14} \\ 21 & 22 & 23 & 24 \end{matrix}$$

is not necessary to obtain an unsatisfiable formula because the only assignment that satisfies all clauses except the two above is that from Fig. 6, which does not satisfy both clauses above. Assume we omit

$$\begin{matrix} & & \overline{13} & \overline{14} \\ 21 & 22 & 23 & 24. \end{matrix}$$

Even with this omission we get a regular proof, which for arbitrary N turns out to be short.

First we get rid of the extension of $\overline{11}\ \overline{12}\ \overline{23}$ and $\overline{11}\ \overline{12}\ \overline{24}$ and get

$$\begin{matrix} \overline{11} & \overline{12} \\ 21 & 22. \end{matrix}$$

From this we get 21 22 as usual. With this clause 21 22 we generate $\overline{13}\ \overline{14}$, eliminating the extending literals $\overline{21}$ and $\overline{22}$. Now, instead of first generating 23 24 and then [ ] using 21 22, which in any case would be irregular, we first eliminate 23 from

$$\begin{matrix} 13 \\ 23 \end{matrix}$$

(by transforming 23 to $\overline{21}$ and $\overline{22}$) and then 24 from

$$\begin{matrix} 14 \\ 24. \end{matrix}$$

Then we derive the empty clause from 13 and 14 and $\overline{13}\ \overline{14}$.

We indicate how to define formulas as above for arbitrary N. Let $N = 16$. The positive clauses of the formula are the columns of Var^{16}; the negative clauses for the lower half (Row 3 and Row 4) of Var^{16} are those from MPHP^{16}. The negative clauses for the upper half are extended negative clauses from MPHP^{16}. The extensions can be visualized by the picture of Var^{16} in Fig. 7. The arrows indicate the literals extending the negative clauses of MPHP^{16} (the additional lines separate the sections as usual).

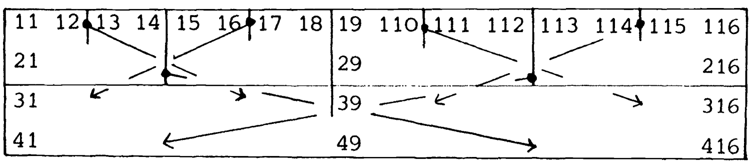


FIG. 7.

First of all, we want the extensions to jump over an unbounded number of rows. Otherwise no significant growth of proof length can be expected. That is, in general we want to extend the negative clauses of MPHP^N of the upper half with negative literals from the lower half. In the example, instead of $\overline{11}\ \overline{12}$, we have $\overline{11}\ \overline{12}\ \overline{35}$, $\overline{11}\ \overline{12}\ \overline{36}$, $\overline{11}\ \overline{12}\ \overline{37}$, $\overline{11}\ \overline{12}\ \overline{38}$, and to get rid of the extensions introducing irregularity (and to ensure unsatisfiability),

$$\begin{matrix} \overline{11} & \overline{12} \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38. \end{matrix}$$

Instead of $\overline{13}\ \overline{14}$ we have $\overline{13}\ \overline{14}\ \overline{x}$ with $x \in \{35, 36, 37, 38\}$. To eliminate these extensions we have

$$\begin{matrix} & & \overline{13} & \overline{14} \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38. \end{matrix}$$

Then we have $\overline{15}\ \overline{16}\ \overline{x}$ and $\overline{17}\ \overline{18}\ \overline{x}$ with $x \in \{31, 32, 33, 34\}$. As demonstrated above for $N = 4$, we do not need extra clauses to eliminate these extensions.

For Row 2 we get $\overline{21}\ \overline{23}\ \overline{x}$ with $x \in \{49, \ldots, 416\}$, and so on. To eliminate the extensions we have a clause $\{\overline{21}, \overline{23}\} \cup$ Row 4. It is similar for the other negative clauses of MPHP for Row 2. Clauses to get rid of the extending literals are only needed for the negative clauses concerning Col 1 through Col 8. In general these clauses are only needed for negative clauses of MPHP, which are in an area where the arrow starts to the right.

A short irregular proof of this formula is obtained as follows.

(1) Get rid of the extending literals of the negative clauses that belong to an area, where the arrow starts going to the right (only for these negative clauses is this possible right at the beginning).

(2) Generate

$$\begin{array}{cccccccc} 31 & 32 & 33 & 34 & & & & \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \end{array}$$

using the negative clauses without negative extending literals (the remaining positive literals now extending these negative clauses are such that they do no harm).

(3) Using the above clause we can eliminate the extensions added to $\overline{15}\,\overline{16}$ and $\overline{17}\,\overline{18}$ and generates

$$\begin{array}{cccccccc} & & & & 35 & 36 & 37 & 38 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48. \end{array}$$

From this we get Halfsec 41. Here irregularities occur.

(4) With Halfsec 41 we can eliminate the literals extending the negative clauses for Row 2 of the right half. This allows us as in (1), (2), and (3) to generate Halfsec 42. Together with Halfsec 41 the empty clause can be derived, again with irregularities.

We construct a regular proof of our formula that strictly sticks to the principle that the variables from the positive clauses are eliminated top to bottom. This proof becomes essentially longer than the above proof.

(1) and (2) are the same as above.

(3) We do not eliminate the literals extending $\overline{15}\,\overline{16}$ and $\overline{17}\,\overline{18}$ because this would lead to irregularities. Instead we derive, by leaving the extending literal $\overline{31}$ as it is,

$$\begin{array}{ccccc} \overline{31} & 35 & 36 & 37 & 38 \\ & 45 & 46 & 47 & 48, \end{array}$$

and then

$$\begin{array}{ccccc} \overline{31} & & & & \\ & 45 & 46 & 47 & 48. \end{array}$$

Doing this for the extending literals $\overline{32}$, $\overline{33}$, and $\overline{34}$, too, we can derive Halfsec 41 by using

$$\begin{array}{cccccccc} 31 & 32 & 33 & 34 & & & & \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48, \end{array}$$

generated in (2).

(4) We generate (similarly to (1)–(3))

$$\overline{41} \qquad 49 \quad 410 \quad \ldots \quad 416$$

analogously for $\overline{42}$ through $\overline{48}$, which together with Halfsec 41 gives us the empty clause. The characteristic "multiplicative effect," yielding superpolynomial growth in general shows up here in the necessity to derive

$$\begin{array}{cccccc} & \overline{39} & 313 & 314 & 315 & 316 \\ \overline{41} & & 413 & 414 & 415 & 416 \end{array}$$

similarly for all combinations of negative literals from $\overline{41}$ through $\overline{48}$ and $\overline{39}$ through $\overline{312}$.

In general a proof according to the above pattern needs something like

$$\frac{N}{2} \cdot \frac{N}{4} \cdot \dots \cdot \frac{N}{2^{\log N}}$$

clauses, which is superpolynomial in N.

Unfortunately, by violating the strict top to bottom strategy we get a short regular proof of our formula.

(1) and (2) are the same as before.

(3) Using the clause generated in (2), we eliminate the negative literals extending $\overline{15}\ \overline{16}$ and $\overline{17}\ \overline{18}$. Moreover, we eliminate 35, 36, 37, 38 from the positive clauses Col 5, Col 6, Col 7, Col 8 to get

$$\left\{\begin{matrix} 15 \\ 25 \end{matrix}\right\} \cup \text{Halfsec } 41, \qquad \left\{\begin{matrix} 16 \\ 26 \end{matrix}\right\} \cup \text{Halfsec } 41, \dots, \left\{\begin{matrix} 18 \\ 28 \end{matrix}\right\} \cup \text{Halfsec } 41.$$

From these, by eliminating

$$\begin{matrix} 15 \\ 25 \end{matrix} \quad \text{through} \quad \begin{matrix} 18 \\ 28 \end{matrix}$$

top to bottom we get Halfsec 41 without irregularity.

(4) Using the clause generated above we eliminate the extending literals of the negative clauses for Row 2 of the right half and the variables from Row 4 of Col 9, ..., Col 16. An inductive argument allows us to derive the empty clause by a short regular proof.

So, by not respecting a strict top to bottom strategy in eliminating the positive literals of the columns, we can avoid the multiplicative effect, making strict top to bottom regular proofs long. In the next section we extend not only the negative clauses for the first rows with negative literals below, but also the negative clauses at the bottom with literals above. This ensures that in a regular proof, according to the last pattern above, the multiplicative effect occurs, too.

2. The formulas. In this section we introduce our formulas and present their short unrestricted resolution proofs.

Convention 2.1. Let, for the rest of this paper, $N \in \mathbb{N}$ be such, that $M = \log N$ is divisible by 3. Let $K = \frac{1}{3} \cdot M$, hence $K = \log \sqrt[3]{N}$.

Our formulas are defined over the variables Var^N from Definition 1.4. We need some more structure on Var^N to define which literal extends which clause.

DEFINITION 2.2. All our notations defined in the following refer to an underlying Var^N, which in applying these notations is clear from the context.

(a) Third 1 is the set of variables from the first K rows, Third 2 the second K rows, and Third 3 the last K rows. We write

$$\text{Lowerthirds} = \text{Third } 2 \cup \text{Third } 3.$$

For $X \subseteq \text{Var}^N$ we define Third $1(X) =$ Third $1 \cap X$. It is similar for the other notations.

(b) The correspondence relation between Third 1 and Third 3, Cor13 $\subseteq$ Third 1 $\times$ Third 3 is given by (for visualization see the example below)

Cor13(x, y)
if and only if
$x \in$ Row i, $y \in$ Row $2K + i$
and
x is covered by Sec y, x is not covered by Halfsec y.

For w, x from one section of Row i we have $\operatorname{Cor} w = \operatorname{Cor} x$ as any section of Row i is covered by a half section of Row $i + 1$. We define $\text{Cor31} \subseteq \text{Third } 3 \times \text{Third } 1$ given by $\text{Cor31} = \text{Cor13}^{-1}$.

The correspondence mapping between Third 3 and Third 2,

$$\text{Cormap32} : \text{Third } 3 \to \text{Third } 2,$$

is given by

$$(2K + i)j \to (K + i)j.$$

It is bijective.

The correspondence relation between Third 3 and Third 2,

$$\text{Cor32} \subseteq \text{Third } 3 \times \text{Third } 2,$$

is given by $\text{Cor32}(x, y)$ if and only if $x \in \text{Row } (2K + i)$, $y \in \text{Row } (K + i)$ and $\operatorname{Sec} y$ covers x, $\operatorname{Halfsec} y$ does not cover x.

Let $x \in$ Third 3 and let y be a variable with

$$\operatorname{Sec}(\operatorname{Cormap} x) = \operatorname{Halfsec}(\operatorname{Cormap} x) \uplus \operatorname{Halfsec} y;$$

then $\text{Cor32}(x) = \operatorname{Halfsec} y$ because if $x \in \text{Row } (2K + i)$, the only section of Row $K + i$ covering x is $\operatorname{Sec}(\operatorname{Cormap} x)$.

Example 2.3. For $N = 64$, $M = 6$, $K = 2$ we get the following structure for Var^N. The correspondences are indicated by arrows (the arrows start and end in the middle of the areas corresponding to each other).

Figure 8 illustrates correspondence relation Cor13.

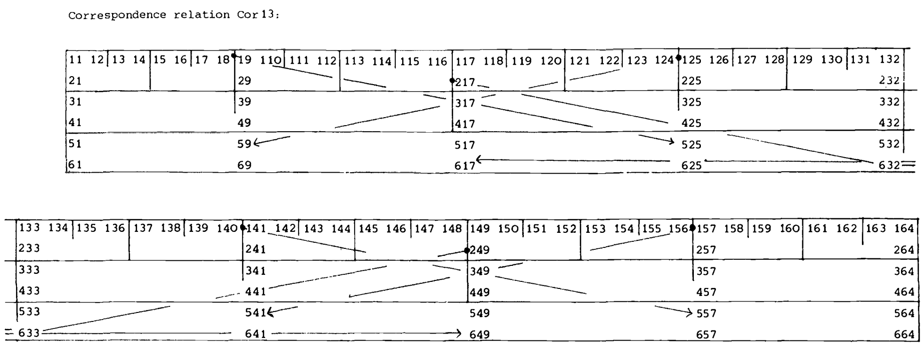


FIG. 8.

Figure 9 illustrates correspondence relation Cor32.

DEFINITION 2.4. For $N \in \mathbb{N}$ having the properties as specified in Convention 2.1, the formula REG^N consists of positive and negative clauses.

The positive clauses of REG^N are the columns of Var^N. The negative clauses of REG^N can be classified into the negative clauses for Third 1, Third 2, Third 3.

The negative clauses for Third 1 are the union over all i with $1 \leq i \leq K$ of all negative clauses for Row i. They are formed as the negative clauses with extending literal introduced in §1. They are the clauses as specified in (1) and (2).

(1) The set of clauses $\{\bar{x}, \bar{y}, \bar{z}\}$ with $z \in \text{Cor13}(x)$ and $x \in \operatorname{Halfsec} i(2j - 1)$, $y \in \operatorname{Halfsec} i(2j)$ for a j with $1 \leq j \leq N/2^i$, that is, x belongs to the left half, y to the right half

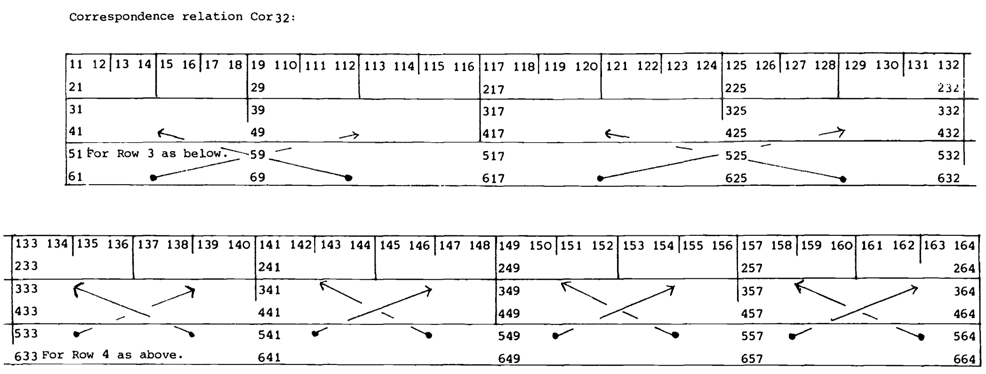


FIG. 9.

of the same section. (As $x, y \in \text{Sec}\, ij$ we have $\text{Cor13}(x) = \text{Cor13}(y)$, hence the definition is well formed.) Here the negative clause $\{\overline{x}, \overline{y}\}$ of MPHP is extended by $\overline{z}$. We call $\overline{z}$ extending literal.

(2) The set of clauses $\{\overline{x}, \overline{y}\} \cup \text{Sec}\, z$, and $z \in \text{Cor13}(x)$, $x \in \text{Halfsec}\, i(2j-1)$, $y \in \text{Halfsec}\, i(2j)$, and x, y are covered by the left half of the section Sec z. This clause allows us to get rid of the extensions of the negative clauses in the short unrestricted resolution proof of REG^N.

The negative clauses for Third 2 are just the clauses $\{\overline{x}, \overline{y}\}$ with $x \in \text{Halfsec}\, i(2j-1)$, $y \in \text{Halfsec}\, i(2j)$ for an i with $K+1 \leq i \leq 2K$, that is, just as in MPHP.

The negative clauses for Third 3 are the union over all i of the negative clauses for Row i with $2K+1 \leq i \leq 3K$. They are specified in (1) and (2).

(1) The set of all clauses $\{\overline{x}, \overline{y}, \overline{z}, \overline{w}\}$ with $x \in \text{Halfsec}\, i(2j-1)$, $y \in \text{Halfsec}\, i(2j)$, $z \in \text{Cor32}(x)$, $w \in \text{Cor32}(y)$. The clause $\{\overline{x}, \overline{y}\}$ would be just a negative clause of MPHP. It is extended by $\overline{z}$ and $\overline{w}$, and $\overline{z}$ and $\overline{w}$ are called extending literals. Note that $\overline{z}$ and $\overline{w}$ belong to different sections.

(2) The set of clauses $\{\overline{x}, \overline{y}, \overline{z}\} \cup \text{Cor32}(y)$ with x, y as in (1), $z \in \text{Cor32}(x)$. The set of clauses $\{\overline{x}, \overline{y}, \overline{w}\} \cup \text{Cor32}(x)$, x, y as above, $w \in \text{Cor32}(y)$. The set of clauses $\{\overline{x}, \overline{y}\} \cup \text{Cor32}(x) \cup \text{Cor32}(y)$, x, y as above.

These clauses allow us to get rid of the extensions of the negative clauses of (1) in the short unrestricted resolution proof. The number of clauses of REG^N is certainly bounded above by $C \cdot \text{Card}\left\{\{x, y, z, w\} | x, y, z, w \in \text{Var}^N\right\}$, which is $O\left(N^8\right)$.

Example 2.5. For $N = 64$ the formula REG^N consists of clauses as follows: the positive clauses are the columns of Var^{64}; the negative clauses for Third 1 are specified in (1) or (2).

(1) For Row 1 we get all clauses of the form $\left\{\overline{1(2j-1)}, \overline{1(2j)}, \overline{5l}\right\}$, which satisfy the following: If $1 \leq j \leq 8$, then $17 \leq l \leq 32$ (cf. the arrows indicating Cor13 in Example 2.3), if $9 \leq j \leq 16$, then $1 \leq l \leq 16$, analogously for $17 \leq j \leq 32$. For Row 2 we get all clauses of the form $\left\{\overline{2(4 \cdot j - s)}, \overline{2(4 \cdot j - r)}, \overline{6l}\right\}$ with $0 \leq r \leq 1$, $2 \leq s \leq 3$, which satisfy the following: If $1 \leq j \leq 8$ then $l \geq 33$; if $9 \leq j \leq 16$, then $l \leq 32$.

(2) For Row 1 we get all clauses of the form $\left\{\overline{1(2j-1)}, \overline{1(2j)}\right\} \cup \{51, \ldots, 532\}$ with $1 \leq j \leq 8$. And $\left\{\overline{1(2j-1)}, \overline{1(2j)}\right\} \cup \{533, \ldots, 564\}$ if $17 \leq j \leq 24$. Note that we do not need these type of clauses for j with $9 \leq j \leq 16$ or $25 \leq j \leq 32$.

The negative clauses for Third 2 should be clear; they are just as in MPHP.

The negative clauses of Third 3 are specified in (1) or (2).

(1) For Row 5 we get all clauses of the form $\left\{\overline{5l_1}, \overline{5l_2}, \overline{3j_1}, \overline{3j_2}\right\}$, which satisfy $1 \le l_1 \le 16$ and $17 \le l_2 \le 32$ or $33 \le l_1 \le 48$ and $49 \le l_2 \le 64$, and, if $1 \le l_1 \le 4$, then $5 \le j_1 \le 8$; if $5 \le l_1 \le 8$, then $1 \le j_1 \le 4$; if $9 \le l_1 \le 12$, then $13 \le j_1 \le 16$, and so on.

Analogously for l_2 and j_2: If $17 \le l_2 \le 20$, then $21 \le j_2 \le 24$; if $21 \le l_2 \le 24$, then $17 \le j_2 \le 20$, and so on.

For Row 6 we get all clauses of the form $\left\{\overline{6l_1}, \overline{6l_2}, \overline{4j_1}, \overline{4j_2}\right\}$, which satisfy $1 \le l_1 \le 32$ and $33 \le l_2 \le 64$, and, if $1 \le l_1 \le 8$, then $9 \le j_1 \le 16$; if $9 \le l_1 \le 16$, then $1 \le l_1 \le 8$, and so on.

Analogously for l_2 and j_2: If $33 \le l_2 \le 40$, then $41 \le j_2 \le 48$ and so on.

(2) For Row 5 we get the following: The set of clauses $\{\overline{5l_1}, \overline{5l_2}, \overline{3j_1}\} \cup \{3j_2, \ldots, 3(j_2+3)\}$ with l_1, l_2, j_1 as before (for Row 5), and if $17 \le l_2 \le 20$, then $j_2 = 21$; if $21 \le l_2 \le 24$, then $j_2 = 17$, and so on.

The set of clauses $\left\{\overline{5l_1}, \overline{5l_2}, \overline{3j_2}\right\} \cup \{3j_1, \ldots, 3(j_1+3)\}$ with l_1, l_2, j_2 as before, and, if $1 \le l_1 \le 4$, then $j_1 = 5$; if $5 \le l_1 \le 8$, then $j_1 = 1$, and so on.

The set of clauses $\left\{\overline{5l_1}, \overline{5l_2}\right\} \cup \{3j_1, \ldots, 3\,(j_1+3)\} \cup \{3j_2, \ldots, 3\,(j_2+3)\}$ with l_1, l_2 as before, and, if $1 \le l_1 \le 4$, then $j_1 = 5$; if $17 \le l_2 \le 20$, then $j_2 = 21$, and so on.

For Row 6 we get the set of clauses $\left\{\overline{6l_1}, \overline{6l_2}, \overline{4j_1}\right\} \cup \{4j_2, \ldots, 4\,(j_2+7)\}$ with l_1, l_2, j_1 as above (for Row 6), and, if $33 \le l_2 \le 40$, then $j_2 = 41$; if $41 \le l_2 \le 48$, then $j_2 = 33$, and so on. We omit the remaining clauses.

THEOREM 2.6. *The formulas* REG^N *have resolution proofs whose length is of* $O\left(N^8\right)$. *In particular, they are unsatisfiable.*

Proof. Let N be fixed. The following is a resolution proof of REG^N.

(1) Get rid of the negative literals extending the negative clauses from Third 3: Let $\mathrm{Cor}\, x = \mathrm{Cor32}(x)$. From $\{\overline{x}, \overline{y}\} \cup \mathrm{Cor}\, x \cup \mathrm{Cor}\, y$ and $\{\overline{x}, \overline{y}, \overline{z}\} \cup \mathrm{Cor}\, y$ for $z \in \mathrm{Cor}\, x$ we generate $\{\overline{x}, \overline{y}\} \cup \mathrm{Cor}\, y$ using N additional clauses. By symmetry, we get $\{\overline{x}, \overline{y}\} \cup \mathrm{Cor}\, x$ using N clauses. With $\{\overline{x}, \overline{y}, \overline{z}, \overline{w}\}$ for $z \in \mathrm{Cor}\, x$, $w \in \mathrm{Cor}\, y$ we get $\{\overline{x}, \overline{y}, \overline{w}\}$ for $w \in \mathrm{Cor}\, y$ in N^2 clauses; using $\{\overline{x}, \overline{y}\} \cup \mathrm{Cor}\, y$ we get $\{\overline{x}, \overline{y}\}$ with another N clauses. As there are at most N^3 clauses $\{\overline{x}, \overline{y}\}$, this process for all negative clauses of Third 3 needs at most N^6 clauses.

(2) We eliminate the extending literals of negative clauses of Third 1 as far as it is possible at this point of the proof, that is, for all clauses $\{\overline{x}, \overline{y}, \overline{z}\}$ with $x, y \in$ Third 1 being covered by the left half of Sec z. In this case we have the clause $\{\overline{x}, \overline{y}\} \cup \mathrm{Sec}\, z$ available. We get $\{\overline{x}, \overline{y}\} \cup (\mathrm{Sec}\, z \backslash \mathrm{Halfsec}\, z)$. Doing this for all clauses for which it is possible requires N^4 additional clauses.

(3) Let, for k with $1 \le k \le K$, the clause C_k be defined by

$$C_k = \bigcup_{i \in \{k, \ldots, K\}} \mathrm{Halfsec}(2K+i)1.$$

Our proof now proceeds as the irregular proof described on page 668. Using only negative clauses whose extending negative literals are already eliminated, we can generate the clause C_1 by constructing a binary tree proof as for MPHP. With C_1 we eliminate the extending literals from the negative clauses $\{\overline{x}, \overline{y}, \overline{z}\}$, where $z \in \mathrm{Halfsec}(2K+1)1$, i.e., $x, y \in$ Row 1 are covered by $\mathrm{Halfsec}(2K+1)2$. From this we get, using the positive clauses that are covered by $\mathrm{Halfsec}(2K+1)2$, the clause

$$C_2 \cup \mathrm{Halfsec}(2k+1)2,$$

which, together with C_1, allows us to derive C_2.

With C_2 we can eliminate the extending literals from the negative clauses $\{\overline{x}, \overline{y}, \overline{z}\}$ with $x, y \in$ Row 2, being covered by Halfsec$(2K + 2)2$, hence $z \in$ Halfsec$(2K + 2)1$ to get $\{\overline{x}, \overline{y}\} \cup C_3$.
From this we get

$$C_3 \cup \text{Halfsec}(2K + 1)3 \cup \text{Quartersec}(2K + 2)3 \quad \text{and}$$
$$C_3 \cup \text{Halfsec}(2K + 1)4 \cup \text{Quartersec}(2K + 2)4.$$

From this we get

$$C_3 \cup \text{Halfsec}(2K + 2)2,$$

which, together with C_2, allows us to derive C_3. Finally we get C_K and Halfsec $M2$, which give the empty clause.

Thus, this proof follows essentially the pattern of our proof of MPHP, except that we have to generate new positive clauses (by this we mean the union of the C_k, with certain fractions of sections) in a prescribed order so that we can get rid in time (that is, before starting the binary tree proof) of the remaining negative literals extending negative clauses. The elimination of these extending literals requires at most N steps, and the length of this part of the proof stays in $O\left(N^4\right)$; see Corollary 1.7. □

The irregularity of this proof shows up in situations where we have generated clauses like C_1 and $D = C_2 \cup$ Halfsec$(2K + 1)2$. In generating C_1, Halfsec$(2K + 1)2$ has already been eliminated by resolution, when eliminating the extending literals. In generating D, Halfsec$(2K + 1)1$ (from C_1) has been eliminated by resolution to eliminate the extending literals.

Note that a short proof, following the proof pattern as sketched on page 670, would not be regular in this case: We would derive first C_1, then we would use C_1 to eliminate the extending literals of the negative clauses as above, and we would use C_1 to eliminate the variables $(2K + 1)j \in$ Halfsec$(2K + 1)2$ from the positive clauses Col j. For this we need the negative clauses $\left\{\overline{(2K + 1)i}, \overline{(2K + 1)j}\right\}$. In generating these clauses, the variables from Sec$(K + 1)i\backslash$Halfsec$(K + 1)i$ and Sec$(K + 1)j\backslash$Halfsec$(K + 1)j$ have been eliminated by resolution. In resolving away the Col j by a short proof, i.e., from top to bottom, these variables have to be eliminated again.

3. Critical assignments and their proof paths.

In this section notions essential for our lower bound proof are introduced and some of their properties are proved.

DEFINITION 3.1. A partial assignment of π of Var^N is s-critical for $s \le M$ if and only if π is obtained by Steps 1–Step s of the following choosing algorithm.

Variable declaration: π is a variable to take as values partial truth value assignments of Var^N, $\pi : \text{Var}^N - \to \{0, 1\}$.

Initialization: π is the totally undefined assignment.

Step 1. For each section Sec of Row 1 choose exactly one half section $\{x\}$ of Sec and set $\pi(x) = 1$. For all $y \in$ Row 1, which have not been chosen, set $\pi(y) = 0$.

⋮

Step i for $i \le K$. For each section Sec of Row i choose exactly *one half section* Hsec $\subseteq$ Sec, and set $\pi(x) = 1$ for all $x \in$ Hsec. For all $y \in$ Row i not chosen, set $\pi(y) = 0$.

⋮

Step $K + 1$. For each section Sec of Row $K + 1$ choose exactly *one variable* $x \in$ Sec such that Col x is a 0-column of π and set $\pi(x) = 1$. Set $\pi(y) = 0$ for all $y \in$ Row $K + 1$ that have not been chosen.

⋮

Step i for $K + 1 \le i \le M$. Analogous to step $K + 1$ with i substituted for $K + 1$.

An assignment of Var^N is critical if and only if it is M-critical. Let ρ be s-critical, π is a critical extension of ρ if and only if π is an extension of ρ and a critical assignment.

Example 3.2. A critical assignment is shown in Fig. 10 (omitting the 0's). Col 2 is the only 0-column of this assignment.

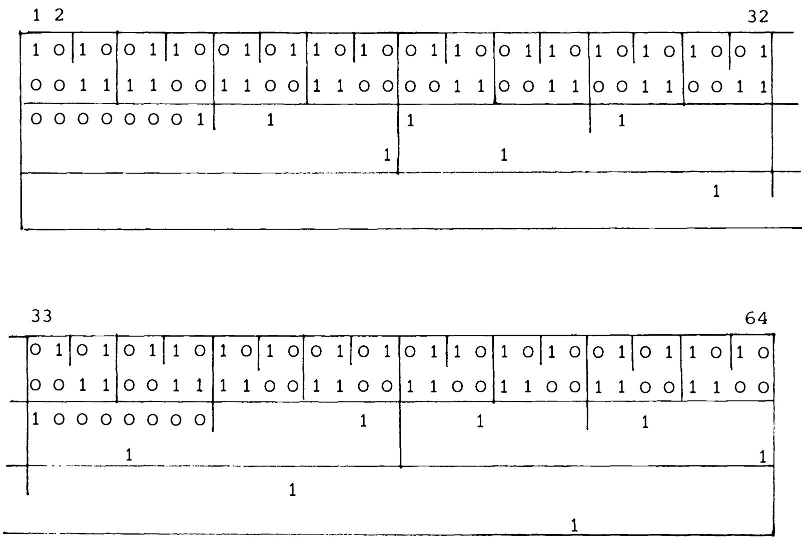


FIG. 10.

LEMMA 3.3.

(a) *Let π be s-critical. For each section* Sec *of* Row s *there is exactly one* 0*-column* Col *of π, such that* Col *meets* Sec. *If π is critical, then π has exactly one* 0*-column.*

(b) *Let π be s-critical,* Col *a* 0*-column of π,* Sec *a section of* Dom π *that meets* Col. *If $x \in$* Sec *and $\pi(x) = 1$, then* Halfsec x *does not meet* Col. *Moreover, there is an $x \in$* Sec *with $\pi(x) = 1$. If* Sec $\subseteq$ Lowerthirds, *this x is unique.*

(c) *We have*

$$\begin{aligned} &\mathrm{Card}\{\pi \mid \pi \textit{ is s-critical}\} \\ = \;& \exp\,(\mathrm{Card}\{\mathrm{Sec} \mid \mathrm{Sec} \textit{ is section of } \mathrm{Dom}\,\pi\}) \\ = \;& \exp\left(\sum_{i=1}^{s} \frac{N}{2^i}\right). \end{aligned}$$

(d) *Let* $\mathrm{Col}_1, \ldots, \mathrm{Col}_m$ *be columns of* Var^N *such that each section of* Row s *meets at most one* Col_r. *Then holds*

$$\mathrm{Card}\,\{\pi \mid \pi \textit{ s-critical and all } \mathrm{Col}_r \textit{ are } 0\textit{-columns of } \pi\} = \exp\left(\left(\sum_{i=1}^{s} \frac{N}{2^i}\right) - s \cdot m\right).$$

(e) *Let $x_1, \ldots, x_n \in$* Third 1 *such that the sections* Sec x_r *are pairwise distinct. Then holds*

$$\mathrm{Card}\,\{\pi \mid \pi \textit{ s-critical with } \pi\,(x_r) = 0 \textit{ for all } r\} = \exp\left(\left(\sum_{i=1}^{s} \frac{N}{2^i}\right) - n\right).$$

Proof. (a) Induction on s. The induction base ($s = 1$) holds because each section of Row 1 has two variables, one of which is assigned 1 by a 1-critical assignment. Let $s > 1$, and let π be s-critical. We have

$$\rho = \pi \Big|_{\text{Row } 1 \cup \ldots \cup \text{Row}(s-1)} \quad \text{is } (s-1)\text{- critical.}$$

By induction hypothesis for each section Sec of Row $(s-1)$ there is exactly one 0-column of ρ meeting Sec. The claim follows from the definition of our choosing algorithm because each section of Row s is made up from two halfsections, and each halfsection of Row s corresponds to a section of Row $s-1$.

(b) Let π, s, Col, Sec be as in the hypothesis. Let Sec $\in$ Row i for $i \leq s$, and

$$\rho = \pi \Big|_{\text{Row } 1 \cup \ldots \cup \text{Row}(i-1)} \cdot$$

If $i = 1$, the claim holds. Otherwise, the claim follows by applying (a) to ρ, noting that for each section Row $i-1$ there is a half section of Row i meeting the same columns and vice versa.

(c) This follows by induction on s using (a). For each section of Dom π the choosing algorithm chooses one of two possibilities.

(d) Let $\text{Col}_1, \ldots, \text{Col}_m$ be as in the hypothesis. Let $s \leq M$. By induction on t with $t \leq s$ we show

$$\text{Card}\,\{\pi \,|\, \pi\ t\text{-critical with } \pi(\text{Col}_r) = 0 \text{ for all } r\}$$
$$= \exp\left(\left(\sum_{i=1}^{t} \frac{N}{2^i}\right) - t \cdot m\right).$$

The induction base ($t = 1$) holds because to get a 1-critical π satisfying the requirements of the theorem we can only choose the half section $\{x\}$ with $\pi(x) = 1$ for the $N/2 - m$ sections, not meeting any of the Col_r. Note that each Col_r meets a different section of Row 1. For the induction step let t be such that $s \geq t > 1$. A t-critical π with π $(\text{Col}_r) = 0$ for all r is constructed from a $(t-1)$-critical ρ with this property by choosing one of two possibilities (cf. the proof of 3.3(a)) for the sections of Row t not meeting any of the Col_r and having no choice for the sections meeting a Col_r. Note that each Col_r is covered by a different section of Row t. These are m sections. Hence we have by induction hypothesis

$$\text{Card}\,\{\pi \,|\, \pi\ t\text{-critical with } \pi\ (\text{Col}_r) = 0 \text{ for all } r\}$$

$$= \exp\left(\left(\sum_{i=1}^{t-1} \frac{N}{2^i}\right) - (t-1) \cdot m\right) \cdot \exp\left(\frac{N}{2^t} - m\right)$$
$$= \exp\left(\left(\sum_{i=1}^{t} \frac{N}{2^i}\right) - t \cdot m\right).$$

(e) The claim follows as our choosing algorithm cannot choose to which half section to assign 1's for exactly whose m sections containing an x_r. (Note that the x_r are from Third 1.) □

LEMMA 3.4. *Let π be a critical assignment of* Var^N; *let* Col *be the* 0*-column of π.*

(a) *The only clause C of* REG^N *with $\pi \not\models C$ is the clause $C =$* Col.

(b) *Let* $u \in$ Lowerthirds(Col); *let* $v \in$ Sec u *be uniquely determined by* $\pi(v) = 1$, *and let* $\rho = \pi[1/u, 0/v]$. (ρ *is obtained from* π *by toggling the values of* u *and* v.) *Then* ρ *is a critical assignment. The* 0-*column of* ρ *is* Col v.

(c) *Let* $u \in$ Third 3(Col), *and let* $v \in$ Sec u *be uniquely determined by* $\pi(v) = 1$. *Let* $x \in$ Cor32(u) *with* $\pi(x) = 1$, $y \in$ Cor32(v) *with* $\pi(y) = 1$. *The only clause* C *of* REG *with* $\pi[1/u] \not\models C$ *is the clause* $C = \{\overline{u}, \overline{v}, \overline{x}, \overline{y}\}$.

(d) *Let* $u \in$ Third 2(Col), *and let* $v \in$ Sec u *with* $\pi(v) = 1$. *The only clause* C *of* REG *with* $\pi[1/u] \not\models C$ *is the clause* $C = \{\overline{u}, \overline{v}\}$.

(e) *Let* $u \in$ Third 1(Col), *and let* $E =$ Sec $u \setminus$ Halfsec u (*then* $\pi(E) = 1$ *as* π *is critical*). *If* C *is a clause of* REG *with* $\pi[1/u] \not\models C$, *then* C *has the form* $C = \{\overline{u}, \overline{e}, \overline{y}\}$ *for an* $e \in E$, $y \in$ Cor13(u).

Proof. Let π be a critical assignment of Var^N with 0-column Col.

(a) If C is a negative clause of REG, then C has the form $\{\overline{x}, \overline{y}\} \cup E$ for a certain E, with x, y belonging to different halfs of the same section. From our choosing algorithm we see that no section contains x, y as above with $\pi(x) = \pi(y) = 1$. Hence for all negative clauses C of REG we have $\pi \models C$. As π has exactly one 0-column, the claim follows.

(b) The claim follows because $\pi[1/u, 0/v]$ can be obtained by our choosing algorithm by making all choices that have been made to generate π with the single exception: u instead of v is chosen to be assigned a 1.

(c) Let u, v, x, y be as in the hypothesis. x, y exist: Let Cor32(u), Cor32(v) $\subseteq$ Row s. Then $\rho = \pi|_{\text{Row}1\cup\ldots\cup\text{Row}s}$ is s-critical, and Col u, Col v are 0-columns of ρ. Hence Cor32(u), Cor32(v) are halfsections not meeting a 0-column, hence there exist unique variables $x \in$ Cor32(u), $y \in$ Cor32(v) with $\rho(x) = \rho(y) = 1$ by Lemma 3.3(b). Let $\delta = \pi[1/u]$. For no positive clause of REG we have $\delta \not\models C$ because δ has no 0-columns. As $x \in$ Cor32(u) and $y \in$ Cor32(v) with $\delta(x) = \delta(y) = 1$ and u, v are the only variables in different halfs of the same section with $\delta(u) = \delta(v) = 1$, we get that $\{\overline{u}, \overline{v}, \overline{x}, \overline{y}\}$ is the only clause of REG which is not satisfied by δ.

(d) The claim follows as $\pi[1/u]$ has no 0-column and u, v are the only variables lying in different halfs of the same section with $\pi(u) = \pi(v) = 1$.

(e) Cor13(u) is a halfsection not meeting Col of a section meeting Col. By 3.3(b) there is a $y \in$ Cor13(u) with $\pi(y) = 1$, this y is unique. Let $\delta = \pi[1/u]$. δ has no 0-columns. As u and $e \in E$ are the only variables with $\delta(u) = \delta(e) = 1$ lying in different halfs of the same section, the clauses C with $\delta \not\models C$ are of the form $C = \{\overline{u}, \overline{e}, \overline{y}\}$. $\square$

DEFINITION 3.5. Let $\mathfrak{R}$ be a resolution proof of REG^N, π a critical assignment of Var^N, and let $\mathfrak{P} = C_1 \ldots C_n$ be the path belonging to π in $\mathfrak{R}$, that is $C_1 \in$ REG^N, $C_n = [\ \ ]$ and $\pi \not\models C_t$ for all t. By Lemma 1.2 this path exists and is unique. (C_1 must be the 0-column of π by Lemma 3.4(a).)

Let $x \in$ Lowerthirds (C_1), and let C_t be the last clause on $\mathfrak{P}$ that contains x. We define x runs ahead on $\mathfrak{P}$ if and only if there is a successor C_r of C_t (i.e., $r > t$) such that there exists a $z \in$ Sec x with $\overline{z} \in C_r$.

Note that the above z is unique, as $\pi(z) = 1$ and $z \in$ Sec $x \subseteq$ Lowerthirds. C_1 runs ahead on $\mathfrak{P}$ if and only if each $x \in$ Lowerthirds (C_1) runs ahead on $\mathfrak{P}$. π runs ahead in $\mathfrak{R}$ if and only if C_1 runs ahead on $\mathfrak{P}$.

The last definition is well formed because for each π we have exactly one path belonging to π in $\mathfrak{R}$.

The following theorem ensures the existence of the running ahead phenomenon.

THEOREM 3.6. *Let* $\mathfrak{R}$ *be a resolution proof of* REG^N. *For each* K-*critical assignment* S (*in the tradition of Haken we denote* K-*critical assignments by* S) *there exists a critical extension* π *of* S, *which runs ahead in* $\mathfrak{R}$.

Proof. Let $\mathfrak{R}$ be a resolution proof of REG^N, and let S be a K-critical assignment of Var^N.

We assume that the claim of the theorem does not hold for S, that is, no critical extension of S runs ahead in $\mathfrak{R}$.

We show the following: For all clauses C_s of $\mathfrak{R}$ that belong to the path $\mathfrak{P} = C_1 \dots C_n$ of a critical extension π of S in $\mathfrak{R}$ holds: There exist clauses $D_1, \dots, D_u$, $u \geq 0$ such that $\mathfrak{Q} = D_1 \cdots D_u C_s \cdots C_n$ is the path of a critical extension ρ of S, and there exist $r \geq s, x \in$ Lowerthirds such that $x \in C_r$ and x does not run ahead on the path $\mathfrak{Q}$. This is a contradiction because [ ] belongs to any path in $\mathfrak{R}$.

The proof proceeds by induction on the number of proof steps to derive the clause C_s.

Induction base. If C_s is of REG^N satisfying the hypothesis of our claim, C_s is the 0-column of π, and the claim follows from our assumption.

Induction step. Assume C_s is a derived clause satisfying the hypothesis; let $\mathfrak{P} = C_1 \dots C_n$ and π be as in the hypothesis. The clause C_s is derived from C_{s-1} and another clause; we call it D.

C_{s-1} satisfies the induction hypothesis and we have clauses $D_1, \dots, D_m$, such that $\mathfrak{Q} = D_1 \dots D_m C_{s-1} \cdots C_n$ is the path in $\mathfrak{R}$ of another critical extension ρ of S. Moreover, we have an $x \in$ Lowerthirds which occurs in a C_r for $r \geq s-1$ and x does not run ahead on $\mathfrak{Q}$. If $x \in C_r$ for an $r \geq s$, the claim holds. The interesting case is: In deriving C_s the x is eliminated, and C_{s-1} is the last clause on $\mathfrak{Q}$ containing x. We must show that C_r contains another variable satisfying our claim (for an $r \geq s$).

Let $y \in \mathrm{Sec}\, x$ be such that $\rho(y) = 1$. y is unique. Col ya is a 0-column of S because $y \notin \mathrm{Dom}\, S$ and ρ is a critical extension of S with $\rho(y) = 1$. We define an assignment of δ by $\delta = \rho[1/x, 0/y]$. Then δ is another critical extension of S (cf. Lemma 3.4(b)). We have $\delta \not\models D$ and $\delta \not\models C_r$ for all $r \geq s$ because $x, \overline{x}, y, \overline{y} \notin C_r$ for $r \geq s$ and $\rho \not\models C_r$ for all $r \geq s$ and $x, y, \overline{y} \notin D$. ($x, \overline{y} \notin C_r$ by assumption; $\overline{x}, y \notin C_r$ because $\rho \not\models C_r$ for all $r \geq s$.) By Lemma 1.2 we have an initial piece of a path $G_1 \cdots G_l D$ in $\mathfrak{R}$ with $\delta \not\models G_i$, $\delta \not\models D$ in $\mathfrak{R}$. Then $\mathfrak{H} = G_1 \cdots G_l D C_s \cdots C_n$ is the path of δ in $\mathfrak{R}$. Hence D satisfies the hypothesis of our claim, and by induction hypothesis there is a $v \in D$ or $v \in C_r$ for an $r \geq s$, satisfying our claim. If $v \in D$, then $v \in C_s$, hence C_s satisfies or claim. □

4. The lower bound.

The following main theorem of this paper is proved in 4.2–4.5.

THEOREM 4.1. *Each regular resolution proof of* REG^N *contains at least* $N^{1/18^2 \cdot \log N}$ *many different clauses.*

Convention 4.2. Let N be in the following a fixed natural number such that H, K, L, M with

$$M = \log N, \quad K = \tfrac{1}{3}M, \quad L = \tfrac{1}{3}K, \quad H = \tfrac{1}{6}K = \tfrac{1}{2}L$$

are natural numbers. Let $\mathfrak{R}$ be a fixed regular resolution proof of REG^N.

DEFINITION 4.3. (a) Let π be a critical assignment of Var^N; let $\mathfrak{P} = C_1 \cdots C_n$ be the path of π in $\mathfrak{R}$. If π runs ahead in $\mathfrak{R}$, C_π is the first clause of $\mathfrak{P}$ with

$$\mathrm{Card}\,(\mathrm{Lowerthirds}\,(C_1) \cap C_\pi) = K.$$

C_π is the first clause of $\mathfrak{P}$, in which half of the variables of Lowerthirds(C_1) are eliminated.

(b) For each K-critical S let π_S be a fixed critical extension of S, such that π_S runs ahead in $\mathfrak{R}$. For each S such a π_S exists by Theorem 3.6. Let $C_S = C_{\pi_S}$.

The clauses C_π have a certain complexity, that is, a certain number of literals.

THEOREM 4.4. *Let* π *be a critical assignment such that* C_π *is defined. Then statement* (*1*) *or statement* (2) (*or both*) *hold.*

(1) *There are H variables* $x_1, \ldots, x_{H^2} \in$ Lowerthirds, *such that for all r $\overline{x}_r \in C_\pi$. Moreover, each section of* Row K *meets at most one of the columns* Col x_r.

(2) *There are $H \cdot H$ variables* $x_1, \ldots, x_{H^2} \in$ Third 1, *such that for all r $x_r \in C_\pi$. Moreover, the x_r are situated such that the sections* Sec x_r *are pairwise distinct.*

Proof. Let π be a critical assignment that runs ahead in $\mathfrak{R}$. Let $\mathfrak{P} = C_1 \ldots C_n$ be the path of π in $\mathfrak{R}$. We define

$D =$ Lowerthirds (C_1),
$Z = D \backslash C_\pi$,
$Z2 = Z \cap$ Third 2,
$Z3 = Z \cap$ Third 3,
$K2 =$ Card $Z2$,
$K3 =$ Card $Z3$.

Z consists of those K variables from Lowerthirds(C_1) that have been eliminated on $\mathfrak{P}$ between C_1 and C_π. We have $K = K2 + K3$.

We distinguish two cases. In the first case we assume $K3 \geq \frac{2}{3}K$, in the second $K3 \leq \frac{2}{3}K$.

Case 1. $K3 \geq \frac{2}{3}K$.

Let $u_1, \ldots, u_L \in Z3$ be $L = \frac{1}{3}K$ different variables, such that for all r Cormap32 $(u_r) \in C_\pi$. (Note, u_r, Cormap32 $(u_r) \in C_1$, $u_r \notin C_\pi$.) As $K2 \leq L$ and Cormap32 $|_{C_1}$ is bijective, there are at most L variables $v \in Z3$ with Cormap32$(v) \in C_\pi$. Hence the u_r exist. Let $x_1, \ldots, x_L \in$ Third 2 be L variables with

$$x_r \in \text{Cor32}\,(u_r) \quad \text{and} \quad \pi\,(x_r) = 1.$$

The x_r exist by Lemma 3.3(b). x_r is the unique variable $x \in$ Sec (Cormap32 (u_r)) with $\pi(x) = 1$.

Let $w_1, \ldots, w_L \in$ Third 3 be given by

$$w_r \in \text{Sec}\, u_r \quad \text{and} \quad \pi\,(w_r) = 1.$$

Let $y_1, \ldots, y_L \in$ Third 2 be given by

$$y_r \in \text{Cor32}\,(w_r) \quad \text{and} \quad \pi\,(y_r) = 1.$$

The y_r exist by Lemma 3.3(b).

In the sequel we show the following: The x_r (or H of them) satisfy statement (1) of the claim of the theorem.

First, the claim concerning the position of the x_r holds: As $\pi\,(x_r) = 1$ and $x_r \in$ Third 2 we get Col x_r is a 0-column of $\pi\,|_{\text{Third1}}$. As $\pi\,|_{\text{Third1}}$ is K-critical, Lemma 3.3(a) implies the claim.

Let r be fixed. We show that $\overline{x}_r \in C_\pi$. We define the assignment ρ by $\rho = \pi\,[1/u_r]$. Note, $u_r \in C_1$, the 0-column of π. As $u_r \notin C_\pi$ we have $\rho \not\models C_\pi$. By Lemma 3.4(c) the only clause C of REG with $\rho \not\models C$ is the clause

$$C = \left\{\overline{u}_r, \overline{w}_r, \overline{x}_r, \overline{y}_r\right\}.$$

By Lemma 1.2 there is an initial piece $L_1 \ldots L_m$ of a path in $\mathfrak{R}$ with $L_1 = C$ and $L_m = C_\pi$. Hence in $\mathfrak{R}$ we have the situation in Fig. 11.

As C_1 runs ahead on the path $\mathfrak{P} = C_1 \cdots C_n$, the variable Cormap32 $(u_r) \in$ Third 2 (C_1) runs ahead on $\mathfrak{P}$. As Cormap32 $(u_r) \in C_\pi$, $x_r \in$ Sec (Cormap32 (u_r)), and $\pi\,(x_r) = 1$, $\overline{x}_r$ occurs in a successor of C_π on $\mathfrak{P}$. As $\overline{x} \in L_1$, the regularity of $\mathfrak{R}$ implies $\overline{x}_r \in C_\pi$.

Case 2. $K3 \leq \frac{2}{3}K$.

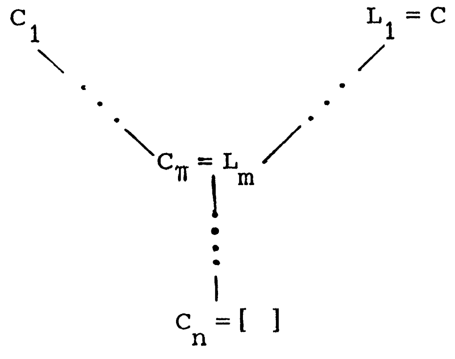


FIG. 11.

Let $u_1, \ldots, u_L \in Z2$ be L different variables. The u_r exist as $K2 \geq L = \frac{1}{3}K$. Let $v_1, \ldots, v_L \in$ Third 2 be uniquely determined by

$$v_r \in \text{Sec}\, u_r \quad \text{and} \quad \pi\,(v_r) = 1.$$

Let $w_1, \ldots, w_L \in$ Third 3 $(C_1) \cap C_\pi$ be L different variables. The w_r exist as $K3 \leq \frac{2}{3}K$.

Let $y_1, \ldots, y_L \in$ Third 3 be uniquely given by

$$y_r \in \text{Sec}\, w_r \quad \text{and} \quad \pi\,(y_r) = 1.$$

Let, for r, s with $1 \leq r \leq L$ and $1 \leq s \leq L$, $z_{r_s} \in$ Third 1 be given by

$$\{z_{r_s}\} = \text{Col}\; v_s \cap \text{Cor31}\,(y_r)\,.$$

If they exist, the z_{r_s} are unique and pairwise distinct because the rows Row y_r and the columns Col v_s are pairwise different. Their existence is shown as follows: Let $y_r \in \text{Row}(2K + i)$. By Definition 2.2 we have $x \in$ Cor31 (y_r) if and only if $x \in$ Row i, x is covered by Sec y_r, and x is not covered by Halfsec y_r. As Halfsec y_r does not meet C_1 (Lemma 3.3(b)) and $w_r \in C_1$, we get

$$\text{Sec}\; y_r = \text{Halfsec}\; y_r \uplus \text{Halfsec}\; w_r.$$

Hence, $x \in$ Cor31 (y_r) if and only if $x \in$ Row i and x is covered by Halfsec w_r. Hence Cor31(y_r) covers Halfsec w_r.

As each section Sec $\subseteq$ Third 2, which meets C_1, is covered by Halfsec w_r (as $w_r \in$ Third 3 (C_1)), we get that Cor31(y_r) covers Sec. As each v_s belongs to such a Sec $\subseteq$ Third 2 that meets C_1, Col v_s meets Cor31(y_r) and z_{r_s} exists.

We show the following: For all s, r with $1 \leq s \leq L$ and $1 \leq r \leq L$,

$$\overline{v}_s \notin C_\pi \quad \text{and} \quad \overline{y}_r \notin C_\pi \quad \text{imply} \quad z_{r_s} \in C_\pi.$$

This implies the theorem by the following case distinction. If

$$\text{Card}\,\{v_s\,|\overline{v}_s \in C_\pi\} \geq H \quad \text{or} \quad \text{Card}\,\{y_r\,|\overline{y}_r \in C_\pi\} \geq H,$$

claim (1) of the theorem follows as in Case 1 because v_s, $y_r \in$ Lowerthirds.

Otherwise, let

$$\text{Card}\,\{v_s\,|\overline{v}_s \notin C_\pi\} > H \quad \text{and} \quad \text{Card}\,\{y_r\,|\overline{y}_r \notin C_\pi\} > H,$$

and let

$$X = \{z_{r_s} \,|\overline{v}_s \notin C_\pi\} \quad \text{and} \quad \{\overline{y}_r \notin C_\pi\}.$$

We have

$$\{z_{r_s} \,|z_{r_s} \in C_\pi\} \supseteq X$$

by the above statement and Card $X \geq H^2$ by the assumption.

Moreover, for $x, y \in X$ with $x \neq y$ we have the following: If x, y belong to the same row of Var^N, there exist $v, v' \in \{v_1, \ldots, v_L\}$, $v \neq v'$ with $x \in \text{Col } v$, $y \in \text{Col } v'$ and Col v, Col v' are 0-columns of $\pi\,|_{\text{Third }1}$. Lemma 3.3(a) implies that Sec $x \neq$ Sec y. Thus claim (2) of the theorem holds.

We still have to show the above statement: Let r, s with $\overline{v}_s \in C_\pi$ and $z_{r_s} \notin C_\pi$ be fixed. We show $\overline{y}_r \in C_\pi$.

Let the assignment ρ be given by $\rho = \pi\,[0/v_s, 1/u_s]$. By Lemma 3.4 (b), ρ is a critical assignment whose 0-column is Col v_s. As $v_s, \overline{v}_s, u_s, \overline{u}_s \notin C_\pi$, we get $\rho \not\models C_\pi$ ($\overline{v}_s, u_s \notin C_\pi$ by assumption, $\overline{u}_s, v_s \in C_\pi$ as $\pi \not\models C_\pi$). Let the assignment δ be given by $\delta = \rho[1/z_r]$. As $\pi(v_s) = 1$ we get Col v_s is a 0-column of $\pi\,|_{\text{Third }1}$ (cf. Lemma 3.3(a)); as $z_{r_s} \in$ Col v_s we have $\pi\left(z_{r_s}\right) = 0$. We get $\delta \not\models C_\pi$ as $z_{r_s} \notin C_\pi$ by assumption and $\rho \not\models C_\pi$. Let $E = \text{Sec } z_{r_s} \backslash \text{Halfsec } z_{r_s}$. Then $\rho(E) = \pi(E) = \delta(E) = \{1\}$, as $\pi\left(z_{r_s}\right) = 0$, $z_{r_s} \in$ Third 1, and π is critical. By Lemma 3.4(e) the clauses C of REG with $C \not\models \delta$ have the form

$$C = \{\overline{z}_{r_s}, \overline{y}_r, \overline{e}\}$$

for an $e \in E$. Note that y_r is the variable y of Cor13$\left(z_{r_s}\right)$ with $\pi(y) = 1$. By Lemma 1.2 there is an initial piece of a path in $\mathfrak{R}$, $L_1 \ldots L_m$ with $L_m = C_\pi$, and L_1 a clause having the form of C above, such that we have a situation as in Fig. 11 in $\mathfrak{R}$.

As C_1 runs ahead on the path $\mathfrak{P} = C_1 \cdots C_n$, w_r runs ahead on this path. As $w_r \in C_\pi$, $\overline{y}_r$ occurs after C_π on $\mathfrak{P}$. As $\overline{y}_r \in L_1$, we get that $\overline{y}_r \in C_\pi$. □

The following corollary finishes the proof of Theorem 4.1.

COROLLARY 4.5. *There are at least* $N^{(1/18^2)\cdot \log N}$ *different clauses* C *in* $\mathfrak{R}$, *such that there exists a* K*-critical assignment* S *with* $C = C_S$.

Proof. Let

$$h(N) = \text{Card}\,\{S | S\ K\text{-critical assignment}\} = \exp\left(\sum_{i=1}^{K} \frac{N}{2^i}\right)$$

by Lemma 3.3(c). Each K-critical S gives us a clause $C_S = C_{\pi_S}$ in the proof $\mathfrak{R}$. That is, we have a total mapping $S \mapsto C_S$. This mapping need not be injective (otherwise we had $h(N)$ many clauses C_S and the proof was finished).

Let

$$g(N) = \text{Max}\left\{\text{Card}\left\{S' \,|C_{S'} = C_S\right\} \middle|\ S\ K\text{-critical assignment}\right\};$$

$g(N)$ is the maximal number of S' that are mapped to one C_S by the above mapping.
We have

$$h(N) \leq g(N) \cdot \text{Card}\,\{C_S \,| S\ K\text{-critical}\},$$

hence

$$\frac{h(N)}{g(N)} \leq \text{Card}\,\{C_S \,| S\ K\text{-critical}\} \leq \text{length of } \mathfrak{R}.$$

Let for $t = 1, 2$,

$$g_t(N) = \mathrm{Max}\{\mathrm{Card}\{S' \mid C_S = C_{S'}\} \mid S\ K\text{-critical and } C_S \text{ satisfies statement } (t) \text{ of Theorem 4.4}\}.$$

As each C_S satisfies (1) or (2) of the last theorem we get

$$g(N) = \mathrm{Max}\,\{g_1(N), g_2(N)\}$$

and

$$\frac{h(N)}{g(N)} = \mathrm{Min}\left\{\frac{h(N)}{g_t(N)}\right\}.$$

We determine a lower bound for $h(N)/g_1(N)$. Let S be K-critical, and let C_S be such that it satisfies statement (1) of the last theorem. Let $x_1, \ldots, x_H \in$ Lowerthirds H variables with $\overline{x}_r \in C_S$, and each section of Row K meets at most one Col x_r.

If S' is a K-critical assignment with $C_s = C_{s'}$, there must exist a critical extension ρ of S' with $\rho \not\models C_S$. We count the maximal number of S' for which this condition can hold: As $\overline{x}_r \in C_S$, S' must be such that it allows for a critical extension ρ with $\rho\,(x_r) = 1$ for all r. As for all x_r holds $x_r \in$ Lowerthirds, the columns Col x_r must be 0-columns of S'. By Lemma 3.3(d) we have at most

$$\exp\left(\left(\sum_{i=1}^{K}\frac{N}{2^i}\right) - H\cdot K\right)$$

such S'. That is,

$$g_1(N) \le \exp\left(\left(\sum_{i=1}^{K}\frac{N}{2^i}\right) - H\cdot K\right).$$

Hence,

$$\frac{h(N)}{g_1(N)} \ge \frac{\exp\left(\sum_{i=1}^{K}\frac{N}{2^i}\right)}{\exp\left(\left(\sum_{i=1}^{K}\frac{N}{2^i}\right) - H\cdot K\right)}$$
$$= \exp\,(H\cdot K) \ge \exp H\cdot H = N^{(1/18^2)\cdot\log N}.$$

We determine a lower bound for $h(N)/g_2(N)$: Let S be K-critical, and let C_S be such that it satisfies statement (2) of the last theorem. Let $x_1, \ldots, x_{H^2} \in$ Third 1 be H^2 variables with $x_r \in C_S$ and belonging to pairwise different sections. If S is a K-critical assignment with $C_S = C_{S'}$, there must be a critical extension ρ of S' with $\rho \not\models C_S$. Hence $\rho\,(x_r) = 0$ for all r with $1 \le r \le H^2$; hence $S'\,(x_r) = 0$.

The x_r satisfy the hypothesis of Lemma 3.3(e), and we have

$$\exp\left(\left(\sum_{i=1}^{k}\frac{N}{2^i}\right) - H^2\right)$$

such S'. Hence

$$g_2(N) \leq \exp\left(\left(\sum_{i=1}^{K} \frac{N}{2^i}\right) - H^2\right)$$

and

$$\frac{h(N)}{g_2(N)} \geq \frac{\exp\left(\sum_{i=1}^{K} \frac{N}{2^i}\right)}{\exp\left(\left(\sum_{i=1}^{K} \frac{N}{2^i}\right) - H^2\right)}$$
$$= \exp\left(H^2\right) = \left(2^H\right)^H = N^{\frac{1}{18^2}\cdot \log N}. \qquad \square$$

Acknowledgment. I should like to thank Professor Kleine Büning and my colleagues for encouragement and advice.

REFERENCES

[1] M. Ajtai, *The complexity of the propositional pigeonhole principle*, in Proceedings of the 29th Symposium on Foundations of Computer Science, IEEE Computer Society Press, Washington, D.C., 1988, pp. 346–355.

[2] W. Bibel, *A comparative study of several proof procedures*, Artif. Intell., 18(1982), pp. 269–293.

[3] S. R. Buss and G. Turán, *Resolution proofs of generalized pigeonhole principles*, Theoret. Comput. Sci., 62(1988), pp. 311–317.

[4] V. Chvátal and E. Szemeredi, *Many hard examples for resolution*, J. Assoc. Comput. Mach., 35(1988), pp. 759–768.

[5] S. A. Cook and R. A. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44(1979), pp. 36–50.

[6] Z. Galil, *On the complexity of regular resolution and the Davis–Putnam procedure*, Theoret. Comput. Sci., 4(1977), pp. 23–46.

[7] A. Goerdt, *Unrestricted resolution versus N-resolution*, Theoret. Comput. Sci., 93 (1992), pp. 159–167.

[8] ———, *Davis–Putnam resolution versus unrestricted resolution*, Ann. Math. and Artif. Intell., 6 (1992), pp. 169–184.

[9] D. W. Loveland, *Automated theorem proving: A logical basis*, North Holland, Amsterdam, 1978.

[10] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39(1985), pp. 297–308.

[11] R. A. Reckhow, *On the lengths of proofs in the propositional calculus*, Ph. D. thesis, Dept. of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1975.

[12] U. Schöning, *Logik für Informatiker*, BI-Taschenbauch, Reihe Informatik 56, Bibliographisches Institut, Mannheim, Germany, 1987.

[13] J. Siekmann and G. Wrightson, eds., *Automation of reasoning– classical papers on computational logic*, Vols. 1 and 2, Springer, New York, 1983.

[14] M. E. Stickel, *A* PROLOG *technology theorem prover: Implementation by an extended* PROLOG *compiler*, J. Automat. Reason., 4(1988), pp. 353–380.

[15] G. S. Tseitin, *On the complexity of derivation in the propositional calculus*, in Structures in Constructive Mathematical Logic, Part II, O.A. Slisenko, ed., Steklov Mathematical Institute, Leningrad, Russia, 1968, pp. 115–125.

[16] A. Urquhart, *Hard examples for resolution*, J. Assoc. Comput. Mach., 34(1987), pp. 209–219.

[17] ———, *The complexity of Gentzen Systems for propositional logic*, Theoret. Comput. Sci., 66(1989), pp. 87–97.

[18] ———, *The relative complexity of resolution and cut-free Gentzen systems*, Ann. Math. and Artif. Intell., 6 (1992).

[19] W. Huang and Y. Xiandon, *A DNF without shortest regular consensus path*, SIAM J. Comput., 16(1987), pp. 836–840.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 684–694, August 1993

LOCAL RANDOMNESS IN POLYNOMIAL RANDOM NUMBER AND RANDOM FUNCTION GENERATORS*

H. NIEDERREITER[†] AND C.P. SCHNORR[‡]

Abstract. A distribution on n-bit strings is called (ε, e)-locally random, if for every choice of $e \leq n$ positions the induced distribution on e-bit strings is in the L_1-norm at most ε away from the uniform distribution on e-bit strings. Local randomness in polynomial random number generators (RNG) that are candidate one-way functions is established. Let N be a squarefree integer and let $f_1, \dots, f_\ell$ be polynomials with coefficients in $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. The RNG that stretches a random $x \in \mathbb{Z}_N$ into the sequence of least significant bits of $f_1(x), \dots, f_\ell(x)$ is studied. It is shown that this RNG provides local randomness if for every prime divisor p of N the polynomials $f_1, \dots, f_\ell$ are linearly independent modulo the subspace of polynomials of degree ≤ 1 in $\mathbb{Z}_p[x]$. Also established is local randomness in polynomial random function generators. This yields candidates for cryptographic hash functions. The concept of local randomness in families of functions extends the concept of universal families of hash functions by Carter and Wegman [*J. Comput. System Sci.*, 18 (1979) pp. 143–154]. The proofs of the results rely on upper bounds for exponential sums.

Key words. random number generator, random function generator, polynomial random number generator, local randomness, families of hash functions, one-way functions

AMS subject classifications. 68Q99, 11K06, 11K45, 11Lxx

1. Introduction and summary. A major open problem in cryptography is to establish one-way functions. While we cannot prove one-wayness it makes sense to analyse candidate one-way functions and to prove properties of these functions that are useful in cryptographic applications. We call a distribution on n-bit strings (ε, e)-locally random if for every choice of $e \leq n$ positions the induced distribution on e-bit strings is in the L_1-norm at most ε away from the uniform distribution on e-bit strings. We prove (ε, e) local randomness for large classes of candidate one-way functions and candidate cryptographic hash functions.

We show that ℓ-tuples of polynomials $(f_1, \dots, f_\ell) \in \mathbb{Z}_N[x]^\ell$ with fixed coefficients in $\mathbb{Z}_N$ and for arbitrary odd squarefree N provide local randomness if for every prime divisor p of N the polynomials $f_1, \dots, f_\ell$ are linearly independent modulo the subspace of polynomials of degree ≤ 1 in $\mathbb{Z}_p[x]$. To give an example let N be prime $N > 2^n$, and let $f_1, \dots, f_\ell \in \mathbb{Z}_N[x]$ be any polynomials that are linearly independent modulo the subspace of polynomials of degree ≤ 1 in $\mathbb{Z}_N[x]$. We prove in Corollary 2 that for random $x \in \mathbb{Z}_N$ the bit string

$$(f_1(x)[1, \dots, f_\ell(x)[1)$$

consisting of the parity bits $f_i(x)[1$ of the residues $f_i(x) \bmod N$ in $[0, N-1]$ is (ε, e) locally random provided that ε, n, ℓ, and e satisfy the inequality

$$2^{-n/2}(2n \log 2)^{e+1} 2\ell \leq \varepsilon \,, \tag{1}$$

where $\log$ denotes the natural logarithm, e.g., we can choose $n \geq 64$, $\ell = \lfloor 2^{n/7} \rfloor$, $\varepsilon = 2^{-n/7}$, $e = \lfloor n/(7 \log n) \rfloor$. Our main result comprises the case that N is an arbitrary odd squarefree integer, that the output contains several bits from each of the residues $f_i(x) \bmod N$, $i = 1, \dots, \ell$, and that x is chosen to be random in a subinterval $[0, M-1]$ of $[0, N-1]$.

Note that the above function

$$[0, N-1] \ni x \;\mapsto\; (f_1(x)[1, \dots, f_\ell(x)[1) \tag{2}$$

*Received by the editors March 4, 1991; accepted for publication April 28, 1992.

[†]Österreichische Akademie der Wissenschaften, Institut für Informationsverarbeitung, Sonnenfelsgasse 19, A – 1010 Wien, Austria (nied@qiinfo.oeaw.ac.at).

[‡]Fachbereich Mathematik / Informatik, Universität Frankfurt, Postfach 111932, Robert Mayer Str. 6–10, 6000 Frankfurt/M., Germany (schnorr@informatik.uni-frankfurt.de).

is a candidate one-way function. No inversion algorithm is known that is polynomial time in $\min(\ell, \log_2 N)$. So far the one-wayness of function (2) has only been proved for random RSA moduli N and RSA polynomials $f_i = x^{e^i}$ (see the following) provided that the RSA scheme is secure. It is, however, possible that this one-way function is more secure than the RSA scheme. We are not aware of any inversion algorithm that, for RSA-moduli N, runs in time $\min(2^\ell, N)^{o(1)}$. On the other hand the RSA scheme can be broken by factoring N using only $\exp(\sqrt{\log N \log\log N})$ many steps. Is there any inversion algorithm that uses knowledge of the factorization of RSA numbers N? Is there any inversion algorithm that uses the structure of particular odd moduli N and of particular nonconstant polynomials f_i? Of course function (2) can easily be inverted for $N = 2$ since $f_i(x)[1$ depends only on $x[1 = x \bmod 2$. Also, the problem of inverting is trivial for constant functions as $f_i(x) = x^{N-1}(\text{mod} N)$ with N prime. Are there more exceptions? The inversion problem for the polynomial map $x \mapsto (f_1(x), \dots, f_\ell(x))$ has been studied by Lagarias and Reeds [9]. Almost nothing is known about the problem to invert (2). However, if we cannot even find inverting algorithms for particular cases given the factorization of the modulus, then this may be a sign that the function (2) is a truly one-way function.

It is important that the source of randomness in

$$(f_1(x)[1, \dots, f_\ell(x)[1)$$

is the random argument x, while the coefficients of $f_1, \dots, f_\ell$ are all fixed. Such functions are cryptographically interesting. A well-known example is the random number generator (RNG) related to the RSA scheme by Alexi et al. [1] and Micali and Schnorr [17], e.g., let N be the product of two large random primes and let the integer $e \geq 3$ be relatively prime to $\varphi(N)$. Then the mapping

$$[1, N] \ni x \mapsto (x^e[1, x^{e^2}[1, \dots, x^{e^\ell}[1),$$

where x^{e^i} is taken modulo N, is a perfect (in the sense of Yao [19]) RNG provided that the RSA scheme is secure.

The functions $x \mapsto (f_1(x)[1, \dots, f_\ell(x)[1)$ extend the class of polynomial random number generators (RNG) proposed by Micali and Schnorr [17] and which stretch a random seed $x \in [1, N2^{-k}]$ into a polynomial residue $P(x)(\text{mod}\, N)$. Micali and Schnorr prove that the m least significant bits of $P(x)(\text{mod}\, N)$ are statistically random within $O(N^{-1/2}2^{k+m}(\log N)^2 \deg_N(P))$ if N is prime and $\deg_N(P) \geq 2$, where $\deg_N(P)$ is the degree of P when P is considered modulo N.

So far local randomness has mainly been studied in functions that are easy to invert; see Alon, Babai, and Itai [2], Luby [11], Schnorr [17], Maurer and Massey [12], Naor and Naor [14], Nisan [16] and Alon et al. [3]. Most of these constructions of local randomness are methodically simple and are not directed towards cryptographic applications. They aim at minimizing the number of random bits that are used in randomised algorithms. Only the quadratic character construction by Alon et al. [3] is similar to our generator—it relies on Weil's theorem. Our proof of local randomness relies on upper bounds for exponential sums and an inequality on quantitative Fourier inversion. We use upper bounds for the discrepancy of polynomial residues from Niederreiter [15] and we extend these bounds from prime moduli to arbitrary squarefree moduli.

We also establish random function generators, associated with fixed polynomials, that provide local statistical randomness. These generators are candidates for cryptographic hash functions. We associate with a polynomial $P \in \mathbb{Z}_N[x]$ of degree d a polynomial function family $P_z(y) = P(z + y)$, where z is the function name and y is the input. For fixed

$k, m \leq \log_2 N$ we associate with a random $z \in \mathbb{Z}_N$ a random function

$$P_z^m : [0, 2^k - 1] \longrightarrow \{0, 1\}^m, \qquad y \longmapsto P(z+y)\,[m$$

where $P(z+y)[m$ denotes the bit string consisting of the m least significant bits of the residue $P(z+y) \bmod N$ in $[0, N-1]$.

We call a function family $\{P_z\}$ *(ε, e)-locally random* if for random z and for any e distinct points $y_1, \dots, y_e$ the distribution of the bit string $P_z(y_1) \cdots P_z(y_e)$ is in the L_1-norm at most ε away from the uniform distribution on em-bit strings.

We prove in Theorem 3.1 that the above family of functions $\{P_z^m\}$ is (ε, e)-locally random, if N is prime, $d = \deg P$ satisfies $e + 1 \leq d < N$, and if

$$N^{-1/2}(\log N)^{e+1} 2^{em+2} d \leq \varepsilon.$$

A family of functions is an e-universal family of hash functions as introduced by Carter and Wegman [7] if and only if it is $(0, e)$-locally random. Our hash functions require fewer random bits than those of Carter and Wegman since we randomize only the input of the polynomial, whereas Carter and Wegman randomize all its coefficients. The main point, however, is that our hash functions are—if $\deg P$ is sufficiently large—candidates for a cryptographically secure hash function, whereas the Carter–Wegman hash functions are easy to invert. Thus for the first time we establish local randomness in families of cryptographic hash functions.

2. Random number generators that provide statistical local randomness.

We present in Theorem 2.1 our main result and we derive from it RNGs that are locally random. In order to prove Theorem 2.1 we establish in Theorem 2.3 an upper bound on the discrepancy for multidimensional polynomial number sequences. This upper bound relies on an upper bound for exponential sums given in Lemma 2.4 and on an inequality of Niederreiter [15] on quantitative Fourier inversion.

Notation. Let $p_1, \dots, p_r$ be r distinct primes, $N = p_1 \cdots p_r$ (i.e., N is squarefree), and $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. Let $\mathbf{F} = (f_1, \dots, f_\ell)$ be an ℓ-tuple of polynomials $f_j \in \mathbb{Z}[x]$, $j = 1, \dots, \ell$. We denote by $d_i(f_j)$ the degree of f_j when f_j is considered mod p_i and we put $d_i(\mathbf{F}) = \max_{1 \leq j \leq \ell} d_i(f_j)$. We define $c_i(\mathbf{F}) = \min(d_i(\mathbf{F}) - 1, \sqrt{p_i})$ for $i = 1, \dots, r$ and $c(\mathbf{F}) = \prod_{i=1}^r (c_i(\mathbf{F}) + 1)$. We call $\mathbf{F}$ *N-admissible* if for every prime divisor p_i of N the polynomials $f_1, \dots, f_\ell$ are linearly independent modulo the subspace of polynomials of degree ≤ 1 in $\mathbb{Z}_{p_i}[x]$. In this case we also call the set of polynomials $f_1, \dots, f_\ell$ N-admissible. Thus $f_1, \dots, f_\ell$ are N-admissible if for $i = 1, \dots, r$ and for all $a_1, \dots, a_\ell \in \mathbb{Z}$ either the polynomial $\sum_{j=1}^{\ell} a_j f_j (\bmod\, p_i)$ is nonlinear or $a_1 = \cdots = a_\ell = 0 (\bmod\, p_i)$.

We let $\log N$ denote the natural logarithm of N. We identify $\mathbb{Z}_N$ with the integer interval $[0, N-1]$. We abbreviate the set $\{0, 1\}^n$ as I_n and we identify the integer interval $[0, 2^n - 1]$ with I_n. If $y \in [0, N-1] = \mathbb{Z}_N$ and $n \leq \log_2 N$, we let $y[n \in I_n$ denote the bit string consisting of the n least significant bits of y. Let $\mathbb{N}$ denote the set of positive integers.

A collection of m least significant output bits. We associate with $\mathbf{F} = (f_1, \dots, f_\ell) \in (\mathbb{Z}[x])^\ell$, $N \in \mathbb{N}$, and $\mathbf{m} = (m_1, \dots, m_\ell) \in \mathbb{N}^\ell$ the mapping

$$\mathbf{F}^{\mathbf{m}} : [1, N] \to I_m, \quad x \mapsto \prod_{j=1}^{\ell} (f_j(x)[m_j) \quad \text{with} \quad f_j(x) \in \mathbb{Z}_N,$$

where $m = \sum_{j=1}^{\ell} m_j$ and $\prod$ is the concatenation of strings. The mapping $\mathbf{F}^{\mathbf{m}}$ outputs a collection of m least significant bits of $\mathbf{F}(x)$, where $\mathbf{F}(x)$ is taken modulo N.

Our main theorem provides explicit estimates for the max-norm difference between the distribution induced by $\mathbf{F}^{\mathbf{m}}(x)$ for random $x \in [1, M] \subset [1, N]$, N-admissible $\mathbf{F}$ and the uniform distribution on $\{0, 1\}^m$.

THEOREM 2.1. *Let N be odd and squarefree, let $\mathbf{F}, \mathbf{m}, m, \mathbf{F}^{\mathbf{m}}$ be as above and let $\mathbf{F}$ be N-admissible. Then for $N > 148$, $1 \le M \le N$, and random $x \in [1, M]$ we have*

$$\max_{z \in \{0,1\}^m} \left|\operatorname{prob}[\mathbf{F}^{\mathbf{m}}(x) = z] - 2^{-m}\right| \le \frac{4}{M}\sqrt{N}(\log N)^{l+1}c(\mathbf{F}) .$$

The condition that $\mathbf{F}$ is N-admissible cannot be completely removed from Theorem 2.1. Theorem 2.1 does not hold for linear polynomials $f_1, \dots, f_\ell$ with $\ell \ge 2$. This is because the least significant bits in two linear polynomials are highly correlated. On the other hand, our proof shows that Theorem 2.1 holds for a single polynomial of degree 1 in the case that $N = M$.

For example, let $N > 2^{512}$ be prime and let $d = 2^{32}$. Then the polynomials $x^2, \dots, x^d$ are N-admissible. Consider for random $x \in [0, N-1]$ the bit string $(x^2[1, \dots, x^d[1) \in I_{d-1}$. For any choice of 24 bit positions $2 \le i_1 < i_2 \cdots < i_{24} \le 2^{32}$ and every $z \in \{0,1\}^{24}$ we have that

$$\left|\operatorname{prob}(x^{i_1}[1 \cdots x^{i_{24}}[1 = z) - 2^{-24}\right| < 2^{-44} .$$

This follows from Theorem 2.1 with $\ell = 24$, $f_j = x^{i_j}$ for $j = 1, \dots, 24$, $N = M$, and $c(\mathbf{F}) \le 2^{32}$.

DEFINITION. A random variable y ranging over a finite set S is called *statistically random* within ε (in S) if $\sum_{s\in S} |\operatorname{prob}(y = s) - 1/\#S| \le \varepsilon$, i.e., the L_1-norm statistical difference of y from the uniform distribution on S is at most ε.

DEFINITION. A probability distribution D on I_n is called (ε, e)*-locally random* if for any sequence of positions $1 \le j_1 < j_2 < \cdots < j_e \le n$ the substring $(y_{j_1}, \dots, y_{j_e}) \in I_e$ of a D-random string $y = (y_1, \dots, y_n)$ is statistically random within ε.

Using Theorem 2.1 we can stretch a short random seed into a long bit string that is "locally random."

COROLLARY 2.2. *Let $N = p_1 \cdots p_r$ be a product of r distinct odd primes, $1 \le M \le N$, and $N > 148$. Let $f_1, \dots, f_\ell \in \mathbb{Z}[x]$ be polynomials of degree at most d that are N-admissible. Then for random $x \in [0, M-1]$ the bit string $(f_1(x)[1, \dots, f_\ell(x)[1) \in I_\ell$ with $f_j(x) \in \mathbb{Z}_N$ is (ε, e)-locally random with $\varepsilon = 2(\sqrt{N}/M)(2\log N)^{e+1}d^r$ for $e = 1, \dots, \ell$.*

Proof. Let $1 \le j_1 < j_2 < \cdots < j_e \le \ell$ be any sequence of e output bit positions. We apply Theorem 2.1 with $\mathbf{F} = (f_{j_1}, \dots, f_{j_e})$, $\mathbf{m} = (1, \dots, 1) \in \mathbb{N}^e$, and $m = \ell = e$. The L_1-norm difference between the distribution induced by $\mathbf{F}^{\mathbf{m}}(x) \in \{0,1\}^e$ and the uniform distribution is at most 2^e-times the max-norm difference. We have $c(\mathbf{F}) \le d^r$, and thus by Theorem 2.1 $\mathbf{F}(x)$ is statistically random within $2(\sqrt{N}/M)\,(2\log N)^{e+1}d^r$. □

The *discrepancy* $D_M^{(\ell)} = D_M^{(\ell)}(\mathbf{y}_1, \dots, \mathbf{y}_M)$ of M points $\mathbf{y}_1, \dots, \mathbf{y}_M \in [0,1)^\ell$ is defined to be

$$D_M^{(\ell)}(\mathbf{y}_1, \dots, \mathbf{y}_M) = \sup_{\mathcal{I}} |F_M(\mathcal{I}) - V(\mathcal{I})|$$

where $\mathcal{I}$ ranges over all half-open subintervals $\mathcal{I}$ of $[0,1)^\ell$, i.e.,

$$\mathcal{I} = \{(z_1, \dots, z_\ell) \in [0,1)^\ell \,\big|\, a_i \le z_i < b_i \text{ for } i = 1, \dots, \ell\}$$

with $0 \le a_i < b_i \le 1$ for $i = 1, \dots, \ell$. $V(\mathcal{I})$ is the volume of $\mathcal{I}$ and $F_M(\mathcal{I}) = M^{-1}\#\{k \mid \mathbf{y}_k \in \mathcal{I}\}$.

The proof of Theorem 2.1 relies on the following upper bound for the discrepancy of multidimensional polynomial sequences. For a real number a we let $\{a\}$ denote the residue of $a \bmod \mathbb{Z}$ in the interval $[0, 1)$.

THEOREM 2.3. *Let N be squarefree and let $D_M^{(\ell)}$ be the discrepancy of the M points $(\{f_1(k)/N\}, \dots, \{f_\ell(k)/N\}) \in [0,1)^\ell$ for $k = 1, \dots, M$. If $\mathbf{F} = (f_1, \dots, f_\ell)$ is N-admissible, then $D_M^{(\ell)} \le (4B/M)\sqrt{N}(\log N)^{\ell+1}c(\mathbf{F})$ for $1 \le M \le N$ and $N > 148$. Here $B = \sqrt{2}$ if N is even and $B = 1$ if N is odd.*

The proof is based on a bound for exponential sums. For $f \in \mathbb{Z}[x]$ and $n \in \mathbb{N}$ define

$$S(f,n) = \sum_{x=1}^{n} e\left(\frac{f(x)}{n}\right) \quad \text{with } e(u) = e^{2\pi\sqrt{-1}u} \quad \text{for} \quad u \in \mathbb{R}.$$

LEMMA 2.4. *If $N = p_1 \dots p_r$ is squarefree, B is as in Theorem 2.3, and $f \in \mathbb{Z}[x]$ is arbitrary, then $|S(f,N)| \le B\sqrt{N}\prod_{i=1}^{r} c_i'(f)$, where $c_i'(f) = \sqrt{p_i}$ if $d_i(f) < 1$ and $c_i'(f) = \min(d_i(f) - 1, \sqrt{p_i})$ if $d_i(f) \ge 1$.*

Proof. Since

$$|S(f,n)| = \left| e\left(\frac{f(0)}{n}\right) \sum_{x=1}^{n} e\left(\frac{f(x) - f(0)}{n}\right) \right| = |S(f - f(0), n)|,$$

we can assume that $f(0) = 0$. Let $n_1, n_2 \in \mathbb{N}$ with $gcd(n_1, n_2) = 1$. If y_1 ranges over $\mathbb{Z} \cap [1, n_1]$ and y_2 ranges over $\mathbb{Z} \cap [1, n_2]$, then $x = n_2 y_1 + n_1 y_2$, considered mod$n_1 n_2$, ranges over $\mathbb{Z} \cap [1, n_1 n_2]$. Therefore

$$S(f, n_1 n_2) = \sum_{y_1=1}^{n_1} \sum_{y_2=1}^{n_2} e\left(\frac{f(n_2 y_1 + n_1 y_2)}{n_1 n_2}\right).$$

If $f(x) = \sum_{j=1}^{d} a_j x^j$, then

$$\begin{aligned} f(n_2 y_1 + n_1 y_2) &= \sum_{j=1}^{d} a_j (n_2 y_1 + n_1 y_2)^j = \sum_{j=1}^{d} a_j (n_2^j y_1^j + n_1^j y_2^j) \\ &= f(n_2 y_1) + f(n_1 y_2) \bmod n_1 n_2. \end{aligned}$$

Thus

$$\begin{aligned} S(f, n_1 n_2) &= \sum_{y_1=1}^{n_1} \sum_{y_2=1}^{n_2} e\left(\frac{f(n_2 y_1) + f(n_1 y_2)}{n_1 n_2}\right) \\ &= \left(\sum_{y_1=1}^{n_1} e\left(\frac{f(n_2 y_1)}{n_1 n_2}\right)\right) \left(\sum_{y_2=1}^{n_2} e\left(\frac{f(n_1 y_2)}{n_1 n_2}\right)\right) \qquad (3) \\ &= S\left(\frac{f(n_2 x)}{n_2}, n_1\right) S\left(\frac{f(n_1 x)}{n_1}, n_2\right). \qquad \square \end{aligned}$$

Now consider $S(f,N)$ with $N = p_1 \dots p_r$. We prove the bound in the lemma by induction on r. For $r = 1$, $d_1(f) \ge 1$, and $p_1 \nmid d_1(f)$ we have the inequality

$$|S(f, p_1)| \le (d_1(f) - 1)\sqrt{p_1}.$$

This inequality is a consequence of the Riemann hypothesis for finite fields established by Weil [18]; see Carlitz and Uchiyama [6], and also Theorem 5.38 in Lidl and Niederreiter [10]. We always have the trivial bound

$$|S(f, p_1)| \le p_1.$$

Thus in all cases

$$|S(f, p_1)| \le Bc'_1(f)\sqrt{p_1}. \tag{4}$$

We need the additional factor $B = \sqrt{2}$ to cover the case $p_1 = d_1(f) = 2$, where $c'_1(f) = 1$ and $|S(f, 2)| \le 2$. Now suppose the result has been shown for some $r \ge 1$ and consider $N = p_1 \dots p_r p_{r+1}$, where the prime factors have been arranged in such a way that p_{r+1} is odd. We apply (3) with $n_1 = p_1 \dots p_r$ and $n_2 = p_{r+1}$, then

$$\begin{aligned} |S(f, N)| &= \left|S\left(\frac{f(n_2x)}{n_2}, n_1\right)\right| \left|S\left(\frac{f(n_1x)}{n_1}, n_2\right)\right| \\ &\le B\sqrt{N}c'_{r+1}\left(\frac{f(n_1x)}{n_1}\right)\prod_{i=1}^{r} c'_i\left(\frac{f(n_2x)}{n_2}\right), \end{aligned}$$

where we used the induction hypothesis and (4) in the second step. We have

$$\frac{f(n_2x)}{n_2} = \sum_{j=1}^{d} a_j n_2^{j-1} x^j.$$

Thus $d_i(f(n_2x)/n_2)$ is the largest j such that $a_j n_2^{j-1} \not\equiv 0 \bmod p_i$. If $1 \le i \le r$, then $a_j n_2^{j-1} \not\equiv 0 \bmod p_i$ if and only if $a_j \not\equiv 0 \bmod p_i$, and so $d_i(f(n_2x)/n_2) = d_i(f)$. Therefore $c'_i(f(n_2x)/n_2) = c'_i(f)$ for $1 \le i \le r$. Similarly, we show that $c'_{r+1}(f(n_1x)/n_1) = c'_{r+1}(f)$, and the proof is complete. □

LEMMA 2.5. *Let $D_M^{(\ell)}$ be the discrepancy of the M points $\mathbf{y}_k \in [0, 1)^\ell$ for $k = 1, \dots, M$ and let $D_N^{(\ell+1)}$ be the discrepancy of the N points $(\mathbf{y}_k, (k-1)/N)$ for $k = 1, \dots, N$. Then $D_M^{(\ell)} \le (N/M)D_N^{(\ell+1)}$ for $1 \le M \le N$.*

Proof. Let J be a half-open subinterval of $[0, 1)^\ell$. Then the conditions $\mathbf{y}_k \in J$ and $1 \le k \le M$ hold simultaneously if and only if

$$\left(\mathbf{y}_k, \frac{k-1}{N}\right) \in J \times \left[0, \frac{M}{N}\right).$$

Thus, if λ_s denotes the s-dimensional Lebesgue measure, then

$$\begin{aligned} &\left|\frac{1}{M}\#\{1 \le k \le M : \mathbf{y}_k \in J\} - \lambda_\ell(J)\right| \\ &= \left|\frac{1}{M}\#\left\{1 \le k \le N : \left(\mathbf{y}_k, \frac{k-1}{N}\right) \in J \times \left[0, \frac{M}{N}\right)\right\} - \lambda_\ell(J)\right| \\ &= \frac{N}{M}\left|\frac{1}{N}\#\left\{1 \le k \le N : \left(\mathbf{y}_k, \frac{k-1}{N}\right) \in J \times \left[0, \frac{M}{N}\right)\right\}\right. \\ &\qquad \left. -\lambda_{\ell+1}\left(J \times \left[0, \frac{M}{N}\right)\right)\right| \\ &\le \frac{N}{M}D_N^{(\ell+1)}. \qquad \square \end{aligned}$$

Proof of Theorem 2.3. Let N have r distinct prime factors. Put $C_\ell(N) = (-N/2, N/2]^\ell \cap \mathbb{Z}^\ell$, $C_\ell^*(N) = C_\ell(N)\backslash\{\mathbf{0}\}$ (here we use, as in Niederreiter [15], the interval $(-N/2, N/2]$ rather than $[0, N)$). For $\mathbf{h} = (h_1, \dots, h_\ell) \in C_\ell(N)$ we put

$$r(\mathbf{h}, N) = \prod_{j=1}^{\ell} r(h_j, N) \text{ with } r(h_j, N) = \begin{cases} 1 & \text{if } h_j = 0 \\ N\sin\frac{\pi|h_j|}{N} & \text{if } h_j \ne 0. \end{cases}$$

By Lemma 2.2 of Niederreiter [15], we get

$$(5)\qquad D_N^{(\ell+1)} \le \frac{\ell+1}{N} + \frac{1}{N} \sum_{\mathbf{h}\in C_{\ell+1}^*(N)} \frac{1}{r(\mathbf{h}, N)} \left|S(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x, N)\right|,$$

where $\mathbf{h} = (h_1, \ldots, h_{\ell+1})$ and $f_1, \ldots, f_\ell \in \mathbb{Z}[x]$. By Lemma 2.4 we have

$$(6)\qquad |S(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x, N)| \le BN^{1/2} \prod_{i=1}^{r} c_i'(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x).$$

If $\mathbf{h} \in C_{\ell+1}^*(N)$ with $(h_1, \ldots, h_\ell) = \mathbf{0}$, then $c_i'(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x) = c_i'(h_{\ell+1}x) = 0$ for some i, namely, when $h_{\ell+1} \not\equiv 0 \bmod p_i$, and so

$$\prod_{i=1}^{r} c_i'(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x) = 0.$$

Thus we have to consider only those $\mathbf{h} \in C_{\ell+1}^*(N)$ with $(h_1, \ldots, h_\ell) \ne \mathbf{0}$. We split up the set of $(h_1, \ldots, h_\ell) \in C_\ell^*(N)$ according to the set of i's for which $d_i(h_1 f_1 + \ldots + h_\ell f_\ell) \le 1$. For $I \subseteq A_r := \{1, 2, \ldots, r\}$ we put $H(I) = \{(h_1, \ldots, h_\ell) \in C_\ell^*(N) : d_i(h_1 f_1 + \ldots + h_\ell f_\ell) \le 1$ if and only if $i \in I\}$. If $(h_1, \ldots, h_\ell) \in H(I)$ and $i \in A_r \backslash I$, then for any $h_{\ell+1} \in C_1(N)$ we have

$$d_i(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x) = d_i(h_1 f_1 + \ldots + h_\ell f_\ell) \ge 2.$$

Since $d_i(h_1 f_1 + \ldots + h_\ell f_\ell) \le d_i(\mathbf{F})$, it follows that

$$c_i'(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x) \le c_i(\mathbf{F}).$$

Using the trivial bound $c_i'(f) \le p_i^{1/2}$, we obtain

$$\prod_{i=1}^{r} c_i'(h_1 f_1 + \ldots + h_\ell f_\ell + h_{\ell+1}x) \le \prod_{i\in I} p_i^{1/2} \cdot \prod_{i\in A_r\backslash I} c_i(\mathbf{F})$$

for any $(h_1, \ldots, h_\ell) \in H(I)$ and $h_{\ell+1} \in C_1(N)$. Together with (5) and (6) this yields

$$D_N^{(\ell+1)} \le \frac{\ell+1}{N} + BN^{-1/2} \sum_{I\subseteq A_r} \prod_{i\in I} p_i^{1/2} \cdot \prod_{i\in A_r\backslash I} c_i(\mathbf{F}) \sum_{\mathbf{h}\in H(I)} \frac{1}{r(\mathbf{h}, N)} \sum_{h_{\ell+1}\in C_1(N)} \frac{1}{r(h_{\ell+1}, N)}.$$

Using the inequality

$$(7)\qquad \sum_{h\in C_1^*(m)} \frac{1}{r(h, m)} < \frac{2}{\pi}\log m + \frac{2}{5} \quad \text{for} \quad m \ge 2$$

from Niederreiter [15, Eq. (2.7)] this yields

$$D_N^{(\ell+1)} < \frac{\ell+1}{N} + BN^{-1/2}\left(\frac{2}{\pi}\log N + \frac{7}{5}\right) \sum_{I\subseteq A_r} \prod_{i\in I} p_i^{1/2} \cdot \prod_{i\in A_r\backslash I} c_i(\mathbf{F}) \sum_{\mathbf{h}\in H(I)} \frac{1}{r(\mathbf{h}, N)}.$$

By the assumption of the theorem, $(f_1, \ldots, f_\ell)$ is N-admissible. Therefore if $(h_1, \ldots, h_\ell) \in H(I)$ we get $h_k \equiv 0 \bmod p_i$ for $i \in I$ and $1 \le k \le \ell$; thus $h_k \equiv 0 \bmod \prod_{i\in I} p_i$ for

$1 \le k \le \ell$. Therefore with $L = \prod_{i\in I} p_i$ we obtain

$$\begin{aligned}
\sum_{\mathbf{h}\in H(I)} \frac{1}{r(\mathbf{h}, N)} &\le \sum_{\substack{\mathbf{h}\in C_\ell^*(N)\\ \mathbf{h}=\mathbf{0} \bmod L}} \frac{1}{r(\mathbf{h}, N)} = \sum_{\mathbf{h}\in C_\ell(N/L)} \frac{1}{r(L\mathbf{h}, N)} - 1\\
&= \left(\sum_{h\in C_1(N/L)} \frac{1}{r(Lh, N)}\right)^\ell - 1 = \left(1 + \sum_{h\in C_1^*(N/L)} \frac{1}{r(Lh, N)}\right)^\ell - 1\\
&= \left(1 + \frac{1}{L}\sum_{h\in C_1^*(N/L)} \frac{1}{r(h, N/L)}\right)^\ell - 1\\
&< \left(1 + \frac{1}{L}\left(\frac{2}{\pi}\log\frac{N}{L} + \frac{2}{5}\right)\right)^\ell - 1 \qquad \text{(by the inequality (7))}\\
&< \frac{\ell}{L}\left(\frac{2}{\pi}\log N + \frac{7}{5}\right)^\ell,
\end{aligned}$$

where we applied the mean-value theorem in the last step. It follows that

$$\begin{aligned}
D_N^{(\ell+1)} &< \frac{\ell+1}{N} + B\ell N^{-1/2}\left(\frac{2}{\pi}\log N + \frac{7}{5}\right)^{\ell+1} \sum_{I\subseteq A_r}\prod_{i\in I} p_i^{-1/2}\cdot \prod_{i\in A_r\setminus I} c_i(\mathbf{F})\\
&= \frac{\ell+1}{N} + B\ell N^{-1/2}\left(\frac{2}{\pi}\log N + \frac{7}{5}\right)^{\ell+1} \prod_{i=1}^r (c_i(\mathbf{F}) + p_i^{-1/2})\\
&< BN^{-1/2}(\log N)^{\ell+1}c(\mathbf{F})\left(\frac{\ell+1}{N^{1/2}(\log N)^{\ell+1}} + \ell\left(\frac{2}{\pi} + \frac{7}{5\log N}\right)^{\ell+1}\right)\\
&< BN^{-1/2}(\log N)^{\ell+1}c(\mathbf{F})\left(\frac{1}{60}(\ell+1)5^{-\ell} + \ell\left(\frac{2}{\pi} + \frac{7}{25}\right)^{\ell+1}\right)\\
&< 4BN^{-1/2}(\log N)^{\ell+1}c(\mathbf{F})
\end{aligned}$$

provided that $\log N \ge 5$, i.e., that $N > 148$. Together with Lemma 2.5 we get the result of Theorem 2.3. □

Proof of Theorem 2.1. Let N be an odd squarefree integer and $\overline{f}_j \in \mathbb{Z}[x]$ be polynomials such that $\overline{f}_j(x) = 2^{-m_j} f_j(x) (\bmod N)$ for $j = 1, \dots, \ell$. Application of Theorem 2.3 to $\overline{\mathbf{F}} = (\overline{f}_1, \dots, \overline{f}_\ell)$ shows that the discrepancy $\overline{D}_M^{(\ell)}$ of $(\{\overline{f}_1(k)/N\}, \dots, \{\overline{f}_\ell(k)/N\})$ for $k = 1, \dots, M$ satisfies

$$\overline{D}_M^{(\ell)} \le \frac{4}{M}\sqrt{N}(\log N)^{\ell+1} c(\mathbf{F}),$$

where we use that $c(\mathbf{F}) = c(\overline{\mathbf{F}})$. We apply to this inequality the equivalence

$$\{\overline{f}_j(x)/N\} \in [k_j 2^{-m_j}, (k_j+1)2^{-m_j}) \qquad \Longleftrightarrow$$

$$[f_j(x)]_N = -k_j N (\bmod 2^{m_j}) \quad \text{for} \quad j = 1, \dots, \ell,$$

where $[f_j(x)]_N$ is the residue of $f_j(x)$ mod N in $[0, N)$, and $0 \le k_j < 2^{m_j}$. To see the equivalence we note that $\{\overline{f}_j(x)/N\} \in [k_j 2^{-m_j}, (k_j+1)2^{-m_j})$ implies that there is an integer

y satisfying

$$k_jN \le y < (k_j+1)N, \quad y = f_j(x) \bmod N, \quad y = 0 \bmod 2^{m_j},$$

and thus $[f_j(x)]_N = -k_jN(\mathrm{mod} 2^{m_j})$. This proves one direction of the equivalence and the converse direction is an immediate consequence.

We see from the above inequality and the equivalence that for every $y \in \{0,1\}^m$

$$\left|\frac{1}{M}\#\{x \in [1,M] : \mathbf{F}^{\mathbf{m}}(x) = y\} - \frac{1}{2^m}\right| \le \frac{4}{M}\sqrt{N}(\log N)^{\ell+1}c(\mathbf{F}). \qquad \square$$

The proof of Theorem 2.1 extends to the following larger class of functions $\mathbf{F}^{\mathbf{u}}$. Let the polynomials $f_1,\dots,f_\ell \in \mathbb{Z}_N[x]$ be N-admissible and let $u_1,\dots,u_\ell$ be integers that are relatively prime to N, $\mathbf{F} = (f_1,\dots,f_\ell)$, and $\mathbf{u} = (u_1,\dots,u_\ell)$. Define $\mathbf{F}^{\mathbf{u}}$ as

$$\mathbf{F}^{\mathbf{u}} : [0, N-1] \ni x \mapsto \left((f_i(x) \bmod N) \bmod u_i \;\middle|\; \text{for } i = 1,\dots,\ell\right).$$

COROLLARY 2.6. *For $N > 148$, $1 \le M \le N$, and random $x \in [0, M-1]$ the max-norm difference between the distribution induced by $\mathbf{F}^{\mathbf{u}}(x)$ and the uniform distribution on $[0, u_1 - 1] \times \cdots \times [0, u_\ell - 1]$ is at most $((4/M)\sqrt{N}(\log N)^{\ell+1}c(\mathbf{F}))$.*

Theorem 2.1 deals with the particular case that the integers u_i are powers of 2. It is necessary that $u_1,\dots,u_\ell$ are relatively prime to N. The proof of the corollary uses the polynomials $\bar{f}_j = u_j^{-1}f_j(\mathrm{mod} N)$ and thus requires a division by u_j modulo N.

3. Random function generators that provide statistical local randomness. Let $H_{k,\ell} = I_\ell^{I_k}$ = "the set of functions $f : I_k \to I_\ell$." A *random function generator* F is an efficient algorithm that generates from names $x \in I_n$ a function $f_x = F(x,*) \in H_{k,\ell}$.

We call a probability distribution D on $H_{k,\ell}$ (ε, e)-*locally random* if for random f, $f \in_D H_{k,\ell}$, for any set of e distinct inputs $y_1,\dots,y_e \in I_k$ the concatenated output $f(y_1)f(y_2)\cdots f(y_e) \in I_{e\ell}$ is statistically random within ε.

The concept of (ε, e)-locally random distribution D on $H_{k,\ell}$ extends the concept of *universal hash functions* of Carter and Wegman [7]. If D is $(0,e)$-locally random, then for any distinct inputs $y_1,\dots,y_e \in I_k$ the bit string $f(y_1)f(y_2)\cdots f(y_e) \in I_e$ is truly random, i.e., D is the probability distribution of an e-universal family of hash functions in the sense of Carter and Wegman.

Carter and Wegman show how to generate an e-universal family of hash functions in $H_{k,k}$ from ke random bits. Let $K = GF(2^k)$ be the field with 2^k elements. If $(a_0,\dots,a_{e-1}) \in K^e$ is random, then the polynomial $P = \sum_{i=0}^{e-1} a_ix^i \in K[x]$ yields an e-universal family of hash functions in $H_{k,k}$.

Let N be a prime and $P \in \mathbb{Z}_N[x]$ be a polynomial with coefficients in the field $\mathbb{Z}_N$. We associate with P and $k, \ell \in \mathbb{N}$, $k, \ell \le \log_2 N$, the function

$$P^\ell : \mathbb{Z}_N \times [0, 2^k - 1] \to I_\ell, \qquad (z, y) \mapsto P(y+z)[\ell.$$

Here we let $P(y+z)[\ell$, for $\ell \le \log_2 N$, denote the bit string consisting of the ℓ least significant bits of the residue of $P(y+z) \bmod N$ that is in $\mathbb{Z}_N$. We let $P_z^\ell : I_k \to I_\ell$ denote the function $P^\ell(z,*)$.

THEOREM 3.1. *Let Nbe prime, $N > 148$, $P \in \mathbb{Z}_N[x]$, $k, \ell \le \log_2 N$, let $P_z^\ell : I_k \to I_\ell$ be as above and let $e + 1 \le \deg P < N$. Then for random $z \in \mathbb{Z}_N$ the family of functions $\{P_z^\ell\}$ is (ε, e)-locally random with $\varepsilon = N^{-1/2}(\log N)^{e+1}2^{e\ell+2}\deg P$.*

Proof. Let $d = \deg P$, let $y_1,\dots,y_e \in \mathbb{Z}_N$ be pairwise distinct, and let $f_i \in \mathbb{Z}_N[x]$ be the polynomial $f_i(x) = P(y_i + x)$ for $i = 1,\dots,e$. We next show that the polynomials

$f_1, \dots, f_e$ are linearly independent modulo the subspace of polynomials of degree ≤ 1 in $\mathbb{Z}_N[x]$. For suppose that there are $b_1, \dots, b_e \in \mathbb{Z}_N$ such that

$$\deg\left(\sum_{i=1}^{e} b_i P(y_i + x)\right) \le 1.$$

Then for $j = d - e + 1, \dots, d$ the jth derivative of this linear combination vanishes at $x = 0$, hence

$$\sum_{i=1}^{e} b_i P^{(j)}(y_i) = 0 \quad \text{for} \quad j = d - e + 1, \dots, d.$$

It is sufficient to prove that the coefficient matrix $[P^{(j)}(y_i)]_{\substack{1 \le i \le e \\ d-e+1 \le j \le d}}$ is nonsingular since this implies that $b_1 = \cdots = b_e = 0$. Suppose that there exist $h_{d-e+1}, \dots, h_d \in \mathbb{Z}_N$ such that

$$\sum_{j=d-e+1}^{d} h_j P^{(j)}(y_i) = 0 \quad \text{for} \quad 1 \le i \le e.$$

Put $g(x) = \sum_{j=d-e+1}^{d} h_j P^{(j)}(x)$, then $g(y_i) = 0$ for $1 \le i \le e$. Since $y_1, \dots, y_e$ are distinct and $\deg(g) \le d - (d - e + 1) = e - 1$ we have $g = 0$, so

$$\sum_{j=d-e+1}^{d} h_j P^{(j)}(x) = 0.$$

Comparing coefficients of x^{e-1} we get $h_{d-e+1} = 0$ (the coefficient of x^{e-1} in $P^{(d-e+1)}$ is nonzero since $d < N$). Continuing in this manner, we obtain $h_{d-e+1} = \cdots = h_d = 0$.

Since $f_1, \dots, f_e$ are linearly independent modulo $\mathbb{Z}_N + x\mathbb{Z}_N$, we can apply Theorem 2.1 to $\mathbf{F} = (f_1, \dots, f_e)$. Since $\prod_{j=1}^{e} f_j(z)[\ell \in I_{e\ell}$ the m in Theorem 2.1 is $e\ell$. The ℓ in Theorem 2.1 is e. Hence $\prod_{j=1}^{e} P(y_j + z)[\ell \in I_{e\ell}$ is statistically random within $\varepsilon = N^{-1/2}(\log N)^{e+1} d\, 2^{e\ell+2}$. Therefore $\{P_z^\ell\}$ is (ε, e)-locally random. □

REFERENCES

[1] W. ALEXI, B. CHOR, O. GOLDREICH, AND C. P. SCHNORR, RSA *and Rabin functions: certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–208.

[2] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomised parallel algorithm for the maximal independent set problem*, J. Algebra, 7 (1986), pp. 567–583.

[3] N. ALON, O. GOLDREICH, J. HASTAD, AND R. PERALTA, *Simple constructions of almost k-wise independent random variables*, Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 544–552.

[4] L. BLUM, M. BLUM, AND M. SHUB, *A simple unpredictable pseudo-random number generator*, SIAM J. Comput., 15 (1986), pp. 364–383.

[5] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982; SIAM J. Comput., 13 (1984), pp. 850–864.

[6] L. CARLITZ AND S. UCHIYAMA, *Bounds for exponential sums*, Duke Math. J., 24 (1957), pp. 37–41.

[7] L. CARTER AND M. WEGMAN, *Universal hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[8] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *How to construct random functions*, Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, 1984; also J. Assoc. Comput. Mach., 33 (1986), pp. 792–807.

[9] J. C. LAGARIAS AND J. A. REEDS, *Unique extrapolation of polynomial recurrences*, SIAM J. Comput., 17 (1988), pp. 342–362.

[10] R. LIDL AND H. NIEDERREITER, *Finite Fields*, Addison–Wesley, Reading, MA, 1983.

[11] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[12] U. M. MAURER AND J. L. MASSEY, *Perfect local randomness in pseudo-random sequences*, in Proceedings Crypto '89, Lecture Notes in Computer Science, Vol. 435, Springer-Verlag, 1990, pp. 100–112.

[13] S. MICALI AND C. P. SCHNORR, *Efficient, perfect polynomial random number generators*, J. Cryptology, 3 (1991), pp. 157–172.

[14] J. NAOR AND M. NAOR, *Small-bias probability spaces: efficient constructions and applications*, Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 213–223.

[15] H. NIEDERREITER, *Pseudo-random numbers and optimal coefficients*, Adv. Math., 26 (1977), pp. 99–181.

[16] N. NISAN, *Pseudorandom generators for space-bounded computation*, Proceedings of the 22nd ACM Symposium on Theory of Computing, 1990, pp. 204–208.

[17] C. P. SCHNORR, *On the construction of random number generators and random function generators*, Proc. EUROCRYPT '88, Lecture Notes in Computer Science, Vol. 330, Springer-Verlag, 1988, pp. 225–232.

[18] A. WEIL, *On some exponential sums*, Proc. Nat. Acad. Sci. USA, 34 (1948), pp. 204–207.

[19] A. C. YAO, *Theory and applications of trapdoor functions*, Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 695–704, August 1993

APPLYING CODING THEORY TO SPARSE INTERPOLATION*

A. DÜR[†] AND J. GRABMEIER[‡]

Abstract. Fast interpolation algorithms are presented for sparse sums of characters on the product monoid U^n with values in a field K in two special cases. In the first case, U is a finite cyclic group of order e, and K is a field that contains a root of unity of order e. Here linear codes over $\mathbb{Z}/e\mathbb{Z}$ can be used to construct sets of evaluation points that allow efficient interpolation by decoding Reed–Solomon codes. In the second case, K is the binary field $GF(2)$, and U is the multiplicative monoid of $GF(2)$. Here sparse sums of characters coincide with sparse Boolean polynomials and can be interpolated using the smallest set of evaluation points by decoding Reed–Muller codes.

Key words. interpolation, sparse multivariate polynomials, Boolean polynomials, coding theory, Reed–Muller codes, Reed–Solomon codes

AMS subject classifications. 11T55, 12E20, 20M99, 68C25, 94B05

1. Introduction.

The interpolation problem for k-sparse multivariate polynomials was studied intensively in recent years [BT88], [CDGK88], [KY88], [GKS88], [DG89], [GKS89], [Z90]. The most general approach was presented in [DG89] where, under the concept of sums of characters of Abelian monoids, most of the earlier results could be unified.

We briefly recall the problem and setting from [DG89]: Let A be an Abelian monoid with neutral element 1_A and let K be a field. According to the well-known Dedekind lemma the set $Hom(A, (K, *))$ of all *characters*, i.e., monoid homomorphisms with $1_A \mapsto 1_K$ from A into the multiplicative monoid $(K, *)$ of K, is a linearly independent subset of the K-space of all maps from A into K. For any subset $X \subseteq Hom(A, (K, *))$ of characters and for every positive integer k define the set X_k of *k-sums* of characters from X by

$$X_k := \left\{ f : A \to K \mid \exists\, f_1, \ldots, f_k \in K, \chi_1, \ldots, \chi_k \in X : \ f = \sum_{\kappa=1}^{k} f_\kappa \chi_\kappa \right\}.$$

In order not to have to discuss degenerate cases separately we assume $\#X \geq k$ and $\chi_\kappa \neq \chi_\sigma$ for $\kappa \neq \sigma$ in $f = \sum_{\kappa=1}^{k} f_\kappa \chi_\kappa$. For given X and k we are interested in procedures by which, for any such $f = \sum_\chi f_\chi \chi$ in X_k, its support

$$\operatorname{supp}(f) := \{\chi \in X \mid f_\chi \neq 0\}$$

and its coefficients f_χ can be determined from as few as possible evaluations of f. A first step to solve this *interpolation problem* is, of course, the study of (small) subsets T of A which allow us to distinguish any nontrivial k-sum of characters from X from the zero map; that is, subsets $T \subseteq A$ such that for any $f \in X_k \setminus \{0\}$ there exists some $a \in T$ with $f(a) \neq 0$. We will refer to such subsets as *zero-test sets* for X_k. Consequently, a subset T of A is suitable for interpolation, i.e., allows us to distinguish any two distinct k-sums of characters, if and only if T is a zero-test set for X_{2k}. However, there does not seem to exist a universally applicable interpolation algorithm that systematically reconstructs the support and the coefficients of f for any $f \in X_k$ from its restriction to T for any field K, any monoid A, any set $X \leq Hom(A, (K, *))$ of k-valued characters of A, and any zero-test set $T \subseteq A$ for X_{2K}. The exception is, of course, provided by the trivial, but surely inefficient, algorithm

*Received by the editors March 14, 1991; accepted for publication (in revised form) March 26, 1992.

[†]Institute of Mathematics, University of Innsbruck, Technikerstraße 26-A-6020, Innsbruck Austria. This work was prepared during a visit at the Scientific Center Heidelberg, IBM Germany, which is gratefully acknowledged.

[‡]Scientific Center Heidelberg, IBM Deutschland Informationssysteme 6mbH, Vangerowstraße 18, D-69020 Heidelberg, Germany (grabm@dhdibm1.bitnet or arne.duer@uibk.ac.at).

that, for all possible choices of $\chi_1, \dots, \chi_K \in X$ and $f_1, \dots, f_K \in K$, compares $f(a)$ with $\sum_{\kappa=1}^{k} f_\kappa \chi_\kappa(a)$ for all $a \in T$.

In the present paper we focus on the algorithmic aspects of interpolation on the product monoid U^n in two special cases. In the first case, U is a finite cyclic group of order e, and K is a finite field that contains a root of unity of order e. In §2 we show that the efficient interpolation algorithm for cyclic monoids presented in Theorem 1 of [DG89] can be applied also to the product U^n if a free linear code over $\mathbb{Z}/e\mathbb{Z}$ of dimension n with the following property can be found: For every selection of k distinct codewords, there exists a component where all codewords are different. We prove that codes with large minimum distance possess this property, and we prove the existence of such codes if e is the product of large primes. For general e, however, the problem of finding such codes or proving their nonexistence remains open.

In the second case, U is the multiplicative monoid of the binary field $K := GF(2)$. Here k-sums of characters can be identified with k-sparse Boolean polynomials in n indeterminates. In the paper [CDGK88] an interpolation algorithm was presented which uses $2k+1$ evaluations at elements from the extension field $GF(2^n)$, whereas the algorithm of the paper [GKS88] requires evaluations at $O(k^4n^2)$ elements from an extension field of $GF(2)$ of degree $O(\log kn)$. The most natural case of allowing no extension of the field can be dealt with by using the general method of Theorem 5 of [DG89], but at the cost of $O(k^2nt)$ evaluations at elements from $GF(2)$, where

$$t := \sum_{\nu=0}^{1+\lfloor \log k \rfloor} \binom{n}{\nu}.$$

Moreover, the algorithm of Theorem 5 of [DG89] in this case is adaptive, in the sense that the set of evaluation points is chosen during the execution of the algorithm and hence may depend on f. In §3 we present a solution of the problem which uses the smallest set of evaluation points. This set was determined in [CDGK88] and is of cardinality t. Our algorithm is based on the theory of Reed–Muller codes from algebraic coding theory and has complexity $O(kn^{2+\log k}(k + \log n))$. Unfortunately, we were not able to generalize this method to the case where U is the multiplicative monoid of a field of more than two elements.

The idea of using well-known algorithms from algebraic coding theory for interpolation is based on the following observation. For a zero-test set $T \subseteq A$ of X_k define a matrix

$$\mathcal{H}_T := (\chi(a))_{a \in T, \chi \in X} \subseteq K^{\#T \times \#X}.$$

Note that the row vectors of $\mathcal{H}_T$ are linearly independent if T is a minimal test set. We now view $\mathcal{H}_T$ as a parity-check matrix of a linear code C_T which is the kernel of the linear map $K^{\#X} \to K^{\#T}$ given by $\mathcal{H}_T$. Thus C_T is a linear code of length $\#X$ and dimension greater or equal to $\#X - \#T$. We claim that the minimum distance of C_T is at least $k+1$. To prove this, suppose that there exists a nonzero codeword $c \in K^{\#X}$ with at most k nonzero components. Then $f := \sum_{\chi \in X} c_\chi \chi$ is a nonzero element of X_k, but $f(a) = \sum_{\chi \in X} c_\chi \chi(a) = 0$ for all $a \in T$ because $c\mathcal{H}_T^t = 0$, which contradicts the assumption that T is a zero-test set.

Next assume that a codeword (of length $\#X$) is sent over a noisy channel and $f := (f_\chi)_{\chi \in X}$ is received. Then the syndrome of f is

$$f\mathcal{H}_T^t = (f(a))_{a \in T},$$

if f is identified with $\sum_{\chi \in X} f_\chi \chi$. Hence the task of interpolating f from black box evaluations amounts to reconstructing the error positions and magnitudes from the syndrome if the all-zero codeword has been sent.

For example, let A be a cyclic monoid. Then $T = \{a^i \mid 0 \le i < 2k\}$ is a zero-test set for X_{2k} where $X = Hom(A, (K, *))$ [DG89]. Here $\mathcal{H}_T$ is the parity-check matrix of a Reed–Solomon code, and decoding algorithms for Reed–Solomon codes implies interpolation algorithms for the cyclic monoid, e.g., Theorem 1 in [DG89].

A second example of this relation will be discussed in detail in §3.

2. Interpolation on products of cyclic groups. Let U be a finite cyclic group of order e with generator η, written multiplicatively, and let K be a field that contains a primitive eth root of unity ω. Identifying η with ω, we can assume, without loss of generality, that U is contained in K. Let A be the product group U^n, and let X be the set of all characters of A with values in K. The characters in X admit the parametrization

$$\chi^\alpha = \chi^{(\alpha_1,\alpha_2,\ldots,\alpha_n)} : U^n \to K$$

where

$$\chi^\alpha(u_1,\ldots,u_n) := u_1^{\alpha_1}\ldots u_n^{\alpha_n}$$

for $\alpha_1,\ldots,\alpha_n \in \{0,1,\ldots,e-1\}$.

Interpolation of k-sparse sums of characters from X can be dealt with by using the general method of Theorem 5 in [DG89]. In this section we study the question of whether the efficient interpolation algorithm presented in Theorem 1 of [DG89] for the cyclic case $n = 1$ can be generalized to the case $n \ge 1$. In [DG89] a subset $D \subseteq A$ is called a *k-distinction set* for X if for any subset $Y \subseteq X$ of cardinality at most k there exists some $a \in D$ such that $\chi(a) \ne \xi(a)$ for all $\chi, \xi \in Y$ with $\chi \ne \xi$. If D is a k-distinction set of X, then $D^{[k]} := \{1\} \cup \{a^i \mid a \in D,\ 0 < i < k\}$ is a zero-test set D for X_k. In the sequel we show how to construct k-distinction sets from linear codes over rings. Then, in Theorem 2.4, we present our algorithm to interpolate functions in X_{2k} from their values on $D^{[2k]}$.

LEMMA 2.1. *Let C be a free linear block code over $\mathbb{Z}/e\mathbb{Z}$ of dimension n and length r with the following property:*

(π) *For every collection of k distinct codewords, there exists a component where all codewords are different.*

Then a k-distinction set for X is given by

$$D := \{d_\rho := (\eta^{c_{\nu,\rho}})_{1\le\nu\le n} \mid 1 \le \rho \le r\},$$

where $(c_{\nu,\rho})_{1\le\nu\le n,1\le\rho\le r}$ is an arbitrary generator matrix of C with linearly independent rows.

Proof. For a character $\chi^\alpha \in X$ and $1 \le \rho \le r$, we have $\chi^\alpha(d_\rho) = \prod_{\nu=1}^n (\eta^{c_{\nu,\rho}})^{\alpha_\nu} = \eta^{\sum_{\nu=1}^n \alpha_\nu c_{\nu,\rho}}$. Furthermore, as the rows of the matrix $(c_{\nu,\rho})$ are linearly independent, to each given subset Y of X of cardinality at most k there correspond k different codewords $(\sum_{\nu=1}^n \alpha_\nu c_{\nu,\rho})_{1\le\rho\le n}$. According to property (π), there exists a position ρ such that the ρ-components of the k codewords are different. Hence, for $a := d_\rho \in D$, we have $\chi(a) \ne \xi(a)$ for all $\chi, \xi \in Y$ with $\chi \ne \xi$. □

Lemma 2.1 suggests looking for codes with property (π). However, codes with property (π) do not exist for all values of e, n, and k, e.g., for $e = n = 2$ and $k = 3$. A sufficient condition for property (π) that can be dealt with more easily is given by the following lemma.

LEMMA 2.2. *Let C be a free linear block code over $\mathbb{Z}/e\mathbb{Z}$ of length r, dimension n, and minimum distance greater or equal to d such that*

$$d/r > 1 - 1/\binom{k}{2}.$$

Then C enjoys property (π).

Proof. Let S be a set of k distinct codewords of C, and let T denote the set of subsets of S with two elements. For $\{v, w\} \in T$, the set $\{\rho \in \{1, 2, \dots, r\} | \ v_\rho = w_\rho\}$ has at most $r - d$ elements. Consequently, the set $\{\rho | \ v_\rho = w_\rho \text{ for some } \{v, w\} \in T\}$ has at most $\binom{k}{2}(r - d)$ elements. Since $\binom{k}{2}(r - d) < r$ by assumption, there exists some ρ such that $v_\rho \neq w_\rho$ for all $\{v, w\} \in T$. □

Lemmas 2.1 and 2.2 show how to construct k-distinction sets from free linear codes over $\mathbb{Z}/e\mathbb{Z}$ of dimension n with large minimum distance. Finding such codes is the main problem of algebraic coding theory and has been studied intensively for a finite field $GF(q)$ instead of the finite ring $\mathbb{Z}/e\mathbb{Z}$. For square-free e, the problem can be reduced to the case where e is prime by the Chinese Remainder Theorem. Let p denote the smallest prime factor of e. If $p \geq \binom{k}{2}(n - 1) - 1$, a Reed–Solomon code over $GF(p)$ of length $r = \binom{k}{2}(n - 1)$ and minimum distance $d = r + 1 - n$ can be used. The following theorem proves the existence of good codes under a weaker condition for p but does not provide a construction of the codes. For general e, only a few constructions of good codes over $\mathbb{Z}/e\mathbb{Z}$ are known [B72], [B75], [S77], [S78].

THEOREM 2.3. *Assume that e is square-free, and let p be the smallest prime factor of e. If $p > \binom{k}{2}$, there exists a free linear code over $\mathbb{Z}/e\mathbb{Z}$ of length r, dimension n, and minimum distance d, such that*

$$d/r > 1 - 1/\binom{k}{2}.$$

Proof. For brevity, we call a free linear code of length r, dimension n, and minimum distance greater or equal to d an $[r, n, \geq d]$ code. Let $e = \prod_{i=1}^{m} p_i$ be the prime factor decomposition of e. Invoking the Chinese Remainder Theorem, an $[r, n, \geq d]$ code over $\mathbb{Z}/e\mathbb{Z}$ can be constructed from $[r, n, \geq d]$ codes over $\mathbb{Z}/p_i\mathbb{Z}$ by applying the ring isomorphism between $\mathbb{Z}/e\mathbb{Z}$ and $\prod_{i=1}^{m} \mathbb{Z}/p_i\mathbb{Z}$ to the entries of the generator matrices. Therefore, it suffices to prove the existence of an $[r, n, \geq d]$ code over $\mathbb{Z}/p\mathbb{Z}$ such that $d/r > 1 - \epsilon$, where $\epsilon = 1/\binom{k}{2}$. Let $d = \lfloor (1 - \epsilon)r \rfloor + 1$. To apply the theorem of Gilbert and Varshamov [LN83], we must choose r such that

$$p^{r-n} > \sum_{i=0}^{d-2} \binom{r-1}{i} (p-1)^i,$$

or, equivalently, such that

$$\sum_{i=r-d+1}^{r-1} \binom{r-1}{i} \beta^i (1 - \beta)^{r-1-i} < \beta^{n-1},$$

where $\beta := 1/p$. Introducing a random variable B, which obeys the binomial distribution with parameters $r - 1$ and β, the left-hand side equals $\text{Prob}\{B \geq \epsilon r\}$ and can be bounded from above by

$$\text{Prob}\{B \geq \epsilon r\} \leq \text{Prob}\{| \ B - (r-1)\beta \ | \geq (\epsilon - \beta)r + \beta\} \leq \frac{(r-1)\beta(1-\beta)}{[(\epsilon - \beta)r + \beta]^2},$$

where, in the last step, we have used Tschebyscheff's inequality. Consequently, the condition for the theorem of Gilbert and Varshamov can be satisfied if r is chosen sufficiently large. □

The big advantage of zero-test sets derived from distinction sets is that sparse sums of characters can be interpolated almost as efficiently as in the cyclic case.

THEOREM 2.4. *Let D be a k-distinction set of X, and let $D^{[2k]} = \{1\} \cup \{a^i \mid a \in D,\ 0 < i < 2k\}$ be the associated zero-test set for X_{2k}. Then a function f in X_k can be interpolated from its values on $D^{[2k]}$ as follows:*

- *Choose an element a in D such that the rank of the matrix $(f(a^{i+j}))_{0 \leq i,j<k}$ is maximal, say, $\tilde{k}$, which equals* $\#\mathrm{supp}(f)$.
- *Interpolate f from its values on $1, a, a^2, .., a^{2\tilde{k}-1}$ as in the cyclic case described in Theorem* 1 *of* [DG89].

Proof. We prove that, for an element a in D, the rank of the matrix $(f(a^{i+j}))_{0\leq i,j<k}$ is maximal if and only if a distinguishes the characters in the support of f, and that the maximum is equal to $\#\mathrm{supp}(f)$. Write $f = \sum_{\alpha\in I} f_\alpha \chi^\alpha$, where $I = \{\alpha \mid \chi^\alpha \in \mathrm{supp}(f)\}$. Then, for arbitrary a in D, the matrix $(f(a^{i+j}))_{0\leq i,j<k}$ can be written as the matrix product $(\chi^\alpha(a^i))D_I(\chi^\alpha(a^i))^t$, where the square matrix D_I is the diagonal matrix $\mathrm{diag}((f_\alpha)_{\alpha\in I})$. As $rank D_I = \#\mathrm{supp}(f)$ and D contains an element a that distinguishes the characters χ^α, $\alpha \in I$, we conclude that the maximal rank of $\{(f(a^{i+j}))_{0\leq i,j<k} | a \in D\}$ is $\tilde{k} := \#\mathrm{supp}(f)$. Now suppose that the element a distinguishes the characters χ^α, $\alpha \in I$. Then the matrix $(\chi^\alpha(a^i))$ has maximal rank, and we find that $rank\ (f(a^{i+j}))_{0\leq i,j<k} = \#\mathrm{supp}(f)$. Conversely, suppose that the rank of the matrix $(f(a^{i+j}))_{0\leq i,j<k}$ is maximal and hence equal to $\#\mathrm{supp}(f)$. Then also $rank(\chi^\alpha(a^i)) = \#\mathrm{supp}(f)$, and we conclude that the element a distinguishes the characters $\chi_\alpha, \alpha \in I$. □

3. Interpolation of sparse Boolean polynomials. Let $U = \{0, 1\}$ be the multiplicative monoid of $K := GF(2)$, and let X be the set of all characters of U^n with values in K. These characters are parametrized by the subsets V of $\{1, ..., n\}$:

$$x_V := \prod_{v\in V} pr_v\ ,$$

where pr_v maps an element to its vth component. We can index the elements in U^n by the same parameter set by setting

$$\chi_W := ([\nu \in W])_{1\leq\nu\leq n}$$

for $W \subseteq \{1, ..., n\}$. Here we have used Iverson's symbol

$$[\ \ ] : \{True, False\} \to \{0, 1\}, True \mapsto 1, False \mapsto 0,$$

from [GKP89]. For $0 \leq r \leq n$, we define

$$\mathcal{T}(r) := \{W \subseteq \{1, ..., n\} \mid \#W \geq n - r\}.$$

In [CDGK88], Theorem 3.2, it was shown that $\mathcal{T}(1 + \lfloor \log k \rfloor)$ parametrizes the smallest zero-test set for X_{2k}, where log denotes the logarithm to the base 2. Hence the matrix

$$\mathcal{H}_r := (x_V(\chi_W))_{W\in\mathcal{T}(r), V\subseteq\{1,...,n\}}$$

is a parity-check matrix of a binary linear code of length 2^n that has dimension $\sum_{r+1\leq i\leq n} \binom{n}{i}$ because the test set is minimal.

THEOREM 3.1. *The linear code defined by the smallest zero-test set of the interpolation problem of k-sparse sums of characters of* $\{0, 1\}^n$ *is the Reed–Muller code of length* 2^n *and order* $n - 2 - \lfloor \log k \rfloor$.

Proof. For $V \subseteq \{1, ..., n\}$ and $W \in \mathcal{T}(r)$, we have $x_V(\chi_W) = [V \subseteq W] = [\bar{W} \subseteq \bar{V}] = x_{\bar{W}}(\chi_{\bar{V}})$. Thus the rows of the matrix $\mathcal{H}_r$ are the evaluations of the monomial characters of total degree at most r at the elements of $\{0, 1\}^n$. By the theory of Reed–Muller codes (see [MS77, Chap. 13, §7]), the matrix $\mathcal{H}_r$ is a generator matrix of the Reed–Muller code of order r and, therefore, is also the parity-check matrix of the Reed–Muller code of order $n - 1 - r$. Setting $r := 1 + \lfloor \log k \rfloor$ now implies the result. □

Theorem 3.1 suggests interpolating k-sparse Boolean polynomials

$$f = \sum_{S \subseteq \{1,...,n\}} f_S x_S$$

by decoding Reed–Muller codes. In technical applications, Reed–Muller codes of low dimension that can efficiently correct a large number of bit errors are used. The decoding algorithm is based on majority voting, which gives, for every position in the received word, an estimate of whether or not an error has occurred (see [MS77, Chap. 13, §7]). Applying majority voting to interpolation we get, for every subset S of $\{1, ..., n\}$, whether the coefficient of x_S is 1 or 0. This is feasible if many errors should be corrected, but inefficient in our situation where the sparsity k is small compared to 2^n.

Nevertheless, we shall derive an efficient interpolation algorithm for k-sparse Boolean polynomials from the observation that the Reed–Muller code of order r is a subcode of the extended BCH code of designed distance $2^{n-r} - 1$. More precisely, we choose a primitive element $\omega \in GF(2^n)$ and a $GF(2)$-linear map

$$\Theta : GF(2)^n \to GF(2^n).$$

For example, Θ could map an element of $GF(2^n)$, constructed as a residue class ring, to its coordinates with respect to the canonical basis consisting of monomials. Then we order the subsets of $\{1, ..., n\}$ via Θ and χ according to

$$\omega^0 < \omega^1 < \ldots < \omega^{2^n-2} < 0.$$

This permutation transforms our Reed–Muller code into an extended cyclic code with zeros

$$\{\omega^i \mid 1 \leq i \leq 2^n - 2, w_2(i) \leq n - r - 1\},$$

where w_2 is the number of nonzero digits in the 2-adic expansion of i (see [MS77, Chap. 13, §5]). We remark that, although not explicitly stated in the proof of Theorem 10, the $GF(2)$-linearity of Θ is essential. Consequently, another parity-check matrix of our Reed–Muller code is given by

$$\begin{pmatrix} 1 & \ldots & 1 & 1 \\ & & & 0 \\ & \omega^{ij} & & \vdots \\ & & & 0 \end{pmatrix}_{\substack{1 \leq i \leq 2^n-2, w_2(i) \leq n-r-1 \\ 0 \leq j \leq 2^n-2}} .$$

Choosing $r = n - 2 - \lfloor \log k \rfloor$, we see that, in particular, the elements $\{\omega^i \mid 1 \leq i \leq 2k - 1\}$ are zeros, and hence the code is a subcode of the extended BCH code with designed distance $2k + 1$. Therefore, we can use the decoding methods for extended BCH codes to interpolate k-sparse sums of characters of $GF(2)^n$ from the $2k$ syndrome components

$$s_{f,i} = \begin{cases} \sum_{S \subseteq \{1,...,n\}} f_S = f(1, 1, \ldots, 1) & \text{for } i = 0 \\ \sum_{0 \leq j \leq 2^n-2} f_{\Theta^{-1}(\omega^j)} \omega^{ij} = \sum_{\emptyset \neq S \subseteq \{1,...,n\}} f_S \Theta(\chi_S)^i & \text{for } 1 \leq i \leq 2k - 1 \end{cases} .$$

However, only in the case $i = 0$ do we get the syndrome component directly from the black box. In the other cases, the syndrome components must be calculated from the evaluations of f on $\mathcal{T}(1 + \lfloor \log k \rfloor)$. The following lemma shows how this can be accomplished by basic arithmetic operations in $GF(2^n)$. To see the relation to the method of Theorem 4.2 of [CDGK88], note that in [CDGK88], BCH-decoding was also used. In [CDGK88], however, the points in $GF(2)^n$ were labelled by the 2-adic expansion which does not give a connection to Reed–Muller codes. Consequently, the algorithm there must get the syndromes directly from the oracle, which requires that the oracle be capable of accepting elements from an extension of $GF(2)$ of degree n.

LEMMA 3.2. *For $1 \le i \le 2k - 1$ we have*

$$s_{f,i} = \sum_{T \in \mathcal{T}(1+\lfloor \log k \rfloor)} f(\chi_T) l_{T,i},$$

where

$$l_{T,i} = \sum_{T \subseteq S} \Theta(\chi_S)^i .$$

Proof. The linear map

$$\begin{array}{ccc} GF(2)^{\{0,1\}^n} & \to & GF(2)^{\mathcal{T}(1+\lfloor \log k \rfloor)} \\ f & \mapsto & (f(\chi_T))_{T \in \mathcal{T}(1+\lfloor \log k \rfloor)} \end{array}$$

is surjective and has the Reed–Muller code of order $n - 2 - \lfloor \log k \rfloor$ as its kernel. Furthermore, for $1 \le i \le 2k - 1$, the map sending an $f \in GF(2)^{\{S | S \subseteq \{1,\dots,n\}\}}$ to the syndrome component $s_{f,i}$ is linear in f and vanishes on the code. Therefore there exists a $GF(2)$-linear map

$$\begin{array}{ccc} GF(2)^{\mathcal{T}(1+\lfloor \log k \rfloor)} & \to & GF(2^n)^{2k-1} \\ (f(\chi_T))_{T \in \mathcal{T}(1+\lfloor \log k \rfloor)} & \mapsto & (s_{f,i})_{1 \le i \le 2k-1} \end{array}$$

given by the matrix $(l_{T,i})_{1 \le i \le 2k-1, T \in \mathcal{T}(1+\lfloor \log k \rfloor)}$, such that

$$s_{f,i} = \sum_{T \in \mathcal{T}(1+\lfloor \log k \rfloor)} l_{T,i} f(\chi_T).$$

Specializing to $f^T := \prod_{\nu=1}^{n}([\nu \notin T] + x_{\{\nu\}}) = \sum_{T \subseteq V} x_V$ and observing that $f^T(\chi_S) = [T = S]$, we obtain the formula

$$l_{T,i} = s_{f^T,i} = \sum_{T \subseteq S} \Theta(\chi_S)^i . \qquad \square$$

Summarizing the above considerations and invoking the theory of BCH codes [B84, Chap. 10], we get the following algorithm:

Interpolation algorithm for k-sparse n-ary Boolean polynomials f. *Let $f = \sum_{S \in \mathcal{F}} x_S$ be a k-sparse n-ary Boolean polynomial, i.e., $\mathcal{F}$ is a subset of the power set of $\{1, \dots, n\}$ with cardinality at most k.*

INPUT: Oracle for f.

Step 1. *Construct the test set $\mathcal{T}(1 + \lfloor \log k \rfloor)$.*

Step 2. *Ask the oracle for the* $t := \sum_{\nu=0}^{1+\lfloor \log k \rfloor} \binom{n}{\nu}$ *values* $(f(\chi_T))_{T \in \mathcal{T}(1+\lfloor \log k \rfloor)}$.

Step 3. *Provide arithmetic of* $GF(2^n)$ *by constructing an irreducible polynomial of degree* n *over* $GF(2)$, *and choose a* $GF(2)$-*linear map* $\Theta : GF(2)^n \to GF(2)^n$.

Step 4. *Calculate the syndrome transformation matrix* $(l_{T,i})_{1 \le i \le 2k-1, T \in \mathcal{T}(1+\lfloor \log k \rfloor)}$.

Step 5. *Calculate the syndrome* $(s_{f,i})_{1 \le i \le 2k-1}$.

Step 6. *Determine the shortest linear-feedback shift register* $((\Lambda_i)_{0 \le i \le 2k}, l)$, *where* $\Lambda_i \in GF(2^n)$, $\Lambda_0 = 1$, $l \in \mathbb{N}$, *which generates the sequence* $(s_{f,0} = f(1, 1, \dots, 1), s_{f,1}, \dots, s_{f,2k-1})$, *i.e.*, $s_{f,i} = -\sum_{1 \le \iota \le i} \Lambda_\iota s_{f,i-\iota}$ *for all* $l + 1 \le i \le 2k - 1$.

Step 7. *Calculate the roots* $\{\lambda_i | 1 \le i \le l\}$ *of the polynomial* $\sum_{0 \le i \le l} \Lambda_{l-i} x^i$.

OUTPUT: $\mathcal{F} = \{\Theta^{-1}(\lambda_i) | 1 \le i \le l\}$

Note in particular that a primitive element of $GF(2^n)$ is required only for theoretical considerations and not in the algorithm. In step 5 one can use the Galois relations

$$s_{f,2i} = s_{f,i}{}^2,$$

where the second subscripts are to be read modulo $2^n - 1$, e.g., $s_{f,3} = s_{f,5}{}^2$ if $n = 3$. The length l of the shift-register determined in step 6 equals $\#\mathcal{F}$. Using l as the degree of $\sum_{0 \le i \le l} \Lambda_{l-i} x^i$ guarantees that the location 0 corresponding to the extension position (constant term of f) is a zero of this polynomial as well. Indeed l and $\max\{\iota | \Lambda_\iota \ne 0\}$ differ by one if and only if f has a constant term, and are equal otherwise.

We next analyze the computational complexity of the algorithm in a serial model of computation. The steps 1, 3, and 4 of the algorithm depend only on k and n, but not on f and, therefore, can be precomputed. The determination of the test set requires $O(t) \le O(n^{1+\log k})$ steps, and the questioning of the oracle in step 2 is counted for free. The irreducible polynomial of degree n over $GF(2)$ required in step 3 can be found rapidly by probabilistic algorithms, see, e.g., [B81], the expected number of operations in $GF(2)$ being $O(n^2 \log^2 n \log \log n)$. A deterministic algorithm to find an irreducible polynomial of degree n is given in [AL86]. The proof that this algorithm runs in polynomial time assumes the extended Riemann hypothesis [K86]. To calculate the syndrome transformation matrix we need at most $O(tk^2)$ additions and $O(tk)$ multiplications in $GF(2^n)$. The calculation of the syndrome in $GF(2^n)$ requires at most $O(tk)$ additions of elements in $GF(2^n)$.

The shift register in step 6 can be computed by the Berlekamp–Massey algorithm, which requires $O(k^2)$, in its recursive form only slightly more than $O(k \log k)$ operations in $GF(2^n)$, cf. [B83, §§7.4, 7.5, 7.6, and 11.7].

In the last step we must find all roots of a univariate polynomial of degree at most k, which happen to be pairwise different and all in $GF(2^n)$. This can be done by Berlekamp's trace algorithm [LN83] if $k \ge n$, and by the *affine method* of Van Oorschot and Vanstone [OV89] otherwise. The precomputations require $O(nk^2)$, respectively, $O(k^3 + nk)$ multiplications in $GF(2^n)$. The remaining costs are $O(nk^2 + n^2k)$. Summing up and counting operations in $GF(2)$, the precomputations require on average $O(kn^{2+\log k}(k + \log n))$ operations in $GF(2)$. Here we used the estimates for the probabilistic algorithm in step 3. For the bit complexity of multiplication in $GF(2^n)$ see [LSW83]. The remaining costs for a specific black box are $O(kn^{2+\log k})$ operations in $GF(2)$.

The complexity in a parallel model of computation depends heavily on the open questions of whether the construction of irreducible polynomials and factoring polynomials are in NC, see [KR90], respectively, [G84] and [G85]. A probabilistic solution of the second problem is given in [G84], where a parallel version of the Cantor–Zassenhaus algorithm is presented with time $O(\log^4(n))$, the number of processors being polynomial in n. Solving a system of linear equations can be done within $O(k^{4.5})$ arithmetic processors and $O(\log^2 k)$ parallel time [M86], and this can be used to solve step 6. We do not know whether there are better parallel algorithms that exploit the special structure of the matrix of the coefficients as does the Berlekamp–Massey algorithm in the sequential case.

Finally, we remark that the approach presented above for the binary field $GF(2^n)$ does not generalize to arbitrary finite fields $GF(q)$ in a straightforward way. For instance, consider the special case $k = 1$ and $q > 2$. In Theorem 3.5 of [CDGK88] it has been proved that the minimum number of points in a zero-test set for X_2 is $2n + 1$. However, for any zero-test set with $2n + 1$ elements, the associated linear code is no generalized Reed–Muller code [L71] because generalized Reed–Muller codes over $GF(q)$ cannot have codimension $2n + 1$.

We have implemented the algorithm in the computer algebra system Scratchpad.

Acknowledgment. We thank K. Werther for implementing parts of the algorithm.

REFERENCES

[AL86] L. M. Adleman and H. W. Lenstra Jr, *Finding irreducible polynomials over finite fields*, Proc. ACM STOC, 8 (1986), pp. 350–355.

[B72] I. F. Blake, *Codes over certain rings*, Inform. and Control, 20 (1972), pp. 396–404.

[B75] ———, *Codes over integer residue rings*, Inform. and Control, 29 (1975), pp. 295–300.

[B81] M. Ben-Or, *Probabilistic algorithms in finite fields*, Proc. 22nd IEEE FOCS (1981), pp. 394–398.

[B83] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, MA, 1983.

[B84] E. R. Berlekamp, *Algebraic Coding Theory*, Revised Ed., Aegean Park Press, Laguna Hills, CA, 1984.

[BT88] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, Proc. 20th ACM STOC 10 (1988), pp. 301–309.

[CDGK88] M. Clausen, A. Dress, J. Grabmeier, and M. Karpinski, *On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields*, Theoret. Comput. Sci., 84 (1991), pp. 151–164.

[DG89] A. Dress and J. Grabmeier, *The interpolation problem for k-sparse polynomials and character sums*, Adv. Appl. Math., 12 (1991), pp. 57–75.

[G84] J. von zur Gathen, *Parallel algorithm for algebraic problems*, SIAM J. Comput., 13 (1984), pp. 808–824.

[G85] J. von zur Gathen, *Irreducible polynomials over finite fields*, Univ. Zürich, Zürich, Switzerland, preprint, 1985.

[GKP89] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.

[GKS88] D. Y. Grigoriev, M. Karpinski, and M. F. Singer, *Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields*, SIAM J. Comput., 19 (1990), pp. 1059–1063.

[GKS89] ———, *The interpolation problem for k-sparse sums of eigenfunctions of operators*, Adv. Appl. Math., 12 (1991), pp. 76–81.

[KR90] R. M. Karp and V. L. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, in Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, Chap. 17, J. van Leuwen, ed., North-Holland, Amsterdam, 1990.

[K86] E. Kranakis, *Primality and Cryptography*, Wiley-Teubner, Stuttgart, Germany, 1986.

[KY88] E. Kaltofen and L. Yagati, *Improved sparse multivariate polynomial interpolation algorithms*, Report 88-17, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1988.

[L71] J. H. van Lint, *Coding Theory*, Lecture Notes in Math. 201, Springer-Verlag, Berlin, 1971.

[LN83] H. Lidl and H. Niederreiter, *Finite Fields*, Addison-Wesley, Reading, MA, 1983.

[LSW83] A. Lempel, G. Seroussi, and S. Winograd, *On the complexity of multiplication in finite fields*, Theoret. Comput. Sci., 22 (1983), pp. 285–296.

[MS77] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1977.

[M86] K. Mulmuley, *A Fast Parallel Algorithm to Compute the Rank of a Matrix over an Arbitrary Field*, Proc. ACM STOC, (1986), pp. 338–339.

[OV89] P. C. van Oorschot and S. A. Vanstone, *A geometric approach to root finding in $GF(q^m)$*, IEEE Trans. Inform. Theory, IT-35 (1989), pp. 444–453.

[S77] E. Spiegel, *Codes over $\mathbb{Z}_m$*, Inform. and Control, 35 (1977), pp. 48–51.

[S78] ———, *Codes over $\mathbb{Z}_m$, revisited*, Inform. and Control 37, (1978), pp. 100–104.

[Z90] R. Zippel, *Interpolating polynomials from their values*, J. Symbolic Comput., 9 (1990), pp. 375–403.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 705–726, August 1993

EXACT IDENTIFICATION OF READ-ONCE FORMULAS USING FIXED POINTS OF AMPLIFICATION FUNCTIONS*

SALLY A. GOLDMAN[†], MICHAEL J. KEARNS[‡], AND ROBERT E. SCHAPIRE[§]

Abstract. In this paper a new technique is described for *exactly identifying* certain classes of read-once Boolean formulas. The method is based on sampling the input-output behavior of the target formula on a probability distribution that is determined by the *fixed point* of the formula's *amplification function* (defined as the probability that a one is output by the formula when each input bit is one independently with probability p). By performing various statistical tests on easily sampled variants of the fixed-point distribution, it is possible to efficiently infer *all* structural information about any logarithmic-depth formula (with high probability). Results are applied to prove the existence of short *universal identification sequences* for large classes of formulas. Also described are extensions of these algorithms to handle high rates of noise, and to learn formulas of unbounded depth in Valiant's model with respect to specific distributions.

Key words. computational learning theory, machine learning, learning with queries, Boolean formulas, read-once formulas, amplification functions, learning with noise

AMS subject classifications. 68Q25, 68T05

1. Introduction.

In this paper we describe efficient algorithms for *exactly identifying* certain classes of read-once Boolean formulas by observing the target formula's behavior on examples drawn randomly according to a fixed and simple distribution that is related to the formula's *amplification function*. The class of read-once Boolean formulas is the subclass of Boolean formulas in which each variable appears at most once. The amplification function $A_f(p)$ for a function $f : \{0, 1\}^n \to \{0, 1\}$ is defined as the probability that the output of f is one when each of the n inputs to f is one independently with probability p. Amplification functions were first studied by Valiant [24] and Boppana [4], [5] in obtaining bounds on monotone formula size for the majority function.

The method used by our algorithms is of central interest. For several classes of formulas, we show that the behavior of the amplification function is *unstable* near the fixed point; that is, the value of $A_f(p)$ varies greatly with a small change in p. This in turn implies that small but easily sampled perturbations of the fixed-point distribution (that is, the distribution where each input is one with probability p, where $A_f(p) = p$) reveal structural information about the formula. For instance, a typical perturbation of the fixed-point distribution hard-wires a single variable to one and sets the remaining variables to one with probability p.

We apply this method to obtain efficient algorithms for exact identification of classes of read-once formulas over various bases. These include the class of logarithmic-depth read-once formulas constructed with NOT gates and three-input majority gates (for which the fixed-point distribution is the uniform distribution), as well as the class of logarithmic-depth read-once formulas constructed with NAND gates (for which the fixed-point distribution assigns one to each input independently with probability $1/\phi \approx 0.618$, where $\phi = (1 + \sqrt{5})/2$ is the golden

*Received by the editor July 11, 1991; accepted for publication (in revised form) March 9, 1992. Most of this research was carried out while all three authors were at the Massachusetts Institute of Technology, Laboratory for Computer Science with support provided by Army Research Office grant DAAL03-86-K-0171, Defense Advanced Research Projects Agency contract N00014-89-J-1988, National Science Foundation grant CCR-88914428, and a grant from the Siemens Corporation.

[†]Department of Computer Science, Washington University, St. Louis, Missouri 63130 (sg@cs. wustl.edu). This author is supported by a General Electric Foundation Junior Faculty grant and National Science Foundation grant CCR-9110108.

[‡]AT&T Bell Laboratories, Murray Hill, New Jersey 07974 (mkearns@research.att.com).

[§]AT&T Bell Laboratories, Murray Hill, New Jersey 07974 (schapire@research.att.com). Support was received from Air Force Office of Scientific Research grant 89-0506 while at Harvard University.

ratio). Thus, for these classes, since the fixed point of the amplification function is the same for all formulas in the class, for each class we obtain a simple product distribution under which the class is learnable. As proved by Kearns and Valiant [16], [13], these same classes of formulas cannot be even weakly approximated in polynomial time when no restriction is placed on the target distribution; thus, our results may be interpreted as demonstrating that while there are some distributions that, in a computationally bounded setting, reveal essentially *no* information about the target formula, there are natural and simple distributions that reveal *all* information.

For Boolean read-once formulas (a superset of the class of formulas constructed from NAND gates) there is an efficient, exact-identification algorithm using membership and equivalence queries due to Angluin, Hellerstein, and Karpinski [1], [11]. The class of read-once majority formulas can also be exactly identified using membership and equivalence queries, as proved by Hancock and Hellerstein [9] and Bshouty et al. [6]. Briefly, in the query model, the learner attempts to infer the target formula by asking questions, or *queries*, of a "teacher." For instance, the learner might ask the teacher what the formula's output would be for a specific assignment to the input variable; this is called a *membership query*. On an *equivalence query*, the learner asks if a given conjectured formula is equivalent to the target formula.

Note that our algorithms' use of a *fixed* distribution can be regarded as a form of "random" membership queries, since this fixed and known distribution can be easily simulated by making random membership queries. Thus, our algorithms are the first efficient procedures for exact identification of logarithmic-depth majority and NAND formulas using only membership queries. Furthermore, the queries used are *nonadaptive* in the sense that they do not depend upon the answers received to previous queries. In contrast, all previous algorithms for exact identification, including the algorithms mentioned above, require highly adaptive queries. As a consequence, our results can be applied to prove the existence of polynomial-length *universal identification sequences* for large classes of formulas; these are fixed sequences of instances for which every unique formula in the class induces a different labeling.

We also prove that our algorithms are *robust* against a large amount of *random misclassification noise*, similar to, but slightly more general than, that considered by Sloan [23] and Angluin and Laird [2]. Specifically, if η_0 and η_1 represent the respective probabilities that an output of 0 or 1 is misclassified, then a robust version of our algorithm can handle any noise rate for which $\eta_0 + \eta_1 \neq 1$; the sample size and computation time required increase only by an inverse quadratic factor in $|1 - \eta_0 - \eta_1|$. Again regarding our algorithms as using "random" membership queries, these are the first efficient procedures performing exact identification in some reasonable model of *noisy queries*. Our algorithms can also tolerate a modest rate of *malicious noise*, as considered by Kearns and Li [14].

Finally, we present an algorithm that learns *any* (not necessarily logarithmic-depth) read-once majority formula in Valiant's model against the uniform distribution. To obtain this result we first show that the target formula can be well approximated by truncating the formula to have only logarithmic depth. We then generalize our algorithm for learning logarithmic-depth read-once formulas to handle such truncated formulas. A similar result also holds for read-once NAND formulas of unbounded depth.

The problem of learning Boolean formulas against special distributions has been considered by a number of other authors. In particular, our technique closely resembles that used by Kearns et al. [15] for learning the class of read-once formulas in disjunctive normal form (DNF) against the uniform distribution. A similar result, though based on a different method, was obtained by Pagallo and Haussler [18]. These results were extended by Hancock and Mansour [10], and by Schapire [21] as described in the following.

Also, Linial, Mansour, and Nisan [17] used a technique based on Fourier spectra to learn the class of constant-depth circuits (constructed from gates of unbounded fan-in) against the uniform distribution. Furst, Jackson, and Smith [7] generalized this result to learn this same class against any *product distribution* (i.e., any distribution in which the setting of each variable is chosen independently of the settings of the other variables). Verbeurgt [25] gives a different algorithm for learning DNF formulas against the uniform distribution. However, all three of these algorithms require quasi-polynomial ($n^{\text{polylog}(n)}$) time, though Verbeurgt's procedure only requires a polynomial-size sample.

Finally, Schapire [21] has recently extended our technique to handle a probabilistic generalization of the class of all read-once Boolean formulas constructed from the usual basis {AND, OR, NOT}. He shows that an arbitrarily good approximation of such formulas can be inferred in polynomial time against any product distribution.

2. Preliminaries. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, Boppana [4], [5] defines its *amplification function* A_f as follows: $A_f(p) = \mathbf{Pr}[f(X_1, \ldots, X_n) = 1]$, where $X_1, \ldots, X_n$ are independent Bernoulli variables that are each one with probability p. The quantity $A_f(p)$ is called the *amplification of f at p*. Valiant [24] uses properties of the amplification function to prove the existence of monotone Boolean formulas of size $O(n^{5.3})$ for the majority function on n inputs. We denote by $D^{(p)}$ the distribution over $\{0,1\}^n$ induced by having each variable independently set to one with probability p.

For $q_j \in \{0,1\}$ and $i_j \in \{1, \ldots, n\}$, $1 \leq j \leq r$, we write $f|x_{i_1} \leftarrow q_1, \ldots, x_{i_r} \leftarrow q_r$ to denote the function obtained from f by fixing or hard wiring each variable x_{i_j} to the value q_j. If each $q_j = q$ for some value q, we abbreviate this by $f|x_{i_1}, \ldots, x_{i_r} \leftarrow q$.

In our framework, the learner is attempting to infer an unknown *target concept c* chosen from some known *concept class $\mathcal{C}$*. In this paper, $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$ is parameterized by the number of variables n, and each $c \in \mathcal{C}_n$ represents a Boolean function on the domain $\{0,1\}^n$. A polynomial-time learning algorithm achieves *exact identification* of a concept class (from some source of information about the target, such as examples or queries) if it can infer a concept that is equal to the target concept on all inputs. A polynomial-time learning algorithm achieves *exact identification with high probability* if, for any $\delta > 0$, it can with probability at least $1 - \delta$ infer a concept that is equal to the target concept on all inputs. In this setting polynomial time means polynomial in n and $1/\delta$. Our algorithms achieve exact identification with high probability when the example source is a particular, fixed distribution.

In the *distribution-free* or *probably approximately correct* (PAC) learning model, introduced by Valiant [24], the learner is given access to labeled (positive and negative) examples of the target concept, drawn randomly according to some unknown *target distribution D*. The learner is also given as input positive real numbers ϵ and δ. The learner's goal is to output with probability at least $1 - \delta$ a *hypothesis h* that has probability at most ϵ of disagreeing with c on a randomly drawn example from D (thus, the hypothesis has *accuracy* at least $1 - \epsilon$). In the case of a randomized hypothesis h, the probability that h disagrees with c is taken over both the random draw from D and the random coin flips used by h. If such a learning algorithm exists (that is, a polynomial-time algorithm meeting the goal for any $n \geq 1$, any $c \in \mathcal{C}_n$, any distribution D, and any ϵ, δ), we say that $\mathcal{C}$ is PAC *learnable*. In this setting, polynomial time means polynomial in n, $1/\epsilon$, and $1/\delta$. In this paper, we are primarily interested in a variant of Valiant's model in which the target distribution is known a priori to belong to a specific restricted class of distributions. This distribution-specific model has also been studied by Benedek and Itai [3].

In this paper we give efficient algorithms that, with high probability, exactly identify the following classes of formulas:

- **The class of logarithmic-depth read-once majority formulas:** This class consists of Boolean formulas of logarithmic depth constructed from the basis {MAJ, NOT} where a MAJ gate computes the majority of three inputs, and each variable appears at most once in the formula. Without loss of generality, we assume all NOT gates are at the input level.

- **The class of logarithmic-depth read-once positive NAND formulas:** This is the class of read-once formulas of logarithmic depth constructed from a basis {NAND} where a NAND gate computes the negation of the logical AND of two inputs. Note that each input appears at most once in the formula and all variables are positive (unnegated). Observe also that the class of read-once positive NAND formulas is equivalent to the class of read-once formulas constructed from alternating levels of OR/AND gates, starting with an OR gate at the top level, with the additional condition that each variable is negated if and only if it enters an OR gate. This observation is easily proved by repeated application of DeMorgan's law.

  Note that by a symmetry argument we can also handle read-once formulas constructed over the {NOR} basis.

In both cases above, our results depend on the restriction on the fan-in of the gates, as well as the requirement that only variables (and not constants) appear in the leaves of the target formula.

Note that because we consider only read-once formulas, there is a unique path from any gate or variable to the output. We define the *level* or *depth of a gate* λ to be the number of gates (not including λ itself) on the path from λ to the output. Thus, the output gate is at level 0. Likewise, we define the *level* or *depth of an input variable* to be the number of gates on the path from the variable to the output. The *depth* of the entire formula is the maximum level of any input, and the *bottom level* consists of all gates and variables of maximum depth.

An input x_i, or a gate λ, *feeds* a gate λ' if the path from x_i or λ to the output goes through λ'. If x_i or λ is an input to λ', then we say that x_i or λ *immediately feeds* λ'. For any two input bits x_i and x_j we define $\Gamma(x_i, x_j)$ to be the deepest gate λ fed by both x_i and x_j. Likewise, $\Gamma(x_i, x_j, x_k)$ is the deepest gate λ fed by x_i, x_j, and x_k. We say that a pair of variables x_i and x_j *meet* at the gate $\Gamma(x_i, x_j)$. Also, if $\Gamma(x_i, x_j) = \Gamma(x_i, x_k) = \Gamma(x_j, x_k) = \Gamma(x_i, x_j, x_k)$, then we say that the variables x_i, x_j, and x_k *meet* at gate $\Gamma(x_i, x_j, x_k)$; otherwise, the triple does not meet in the formula. (Note that this only makes sense if there are gates with more than two inputs, such as a three-input majority gate.)

All logarithms in this paper are base two.

3. Exact identification of read-once majority formulas. In this section we use properties of amplification functions to obtain a polynomial-time algorithm that, with high probability, exactly identifies any read-once majority formula of logarithmic depth from random examples drawn according to a uniform distribution.

This type of formula is used by Schapire [20] in his proof that a concept class is weakly learnable in polynomial time if and only if it is strongly learnable in polynomial time. That is, the hypothesis output by his boosting procedure can be viewed as a majority formula whose inputs are the hypotheses output by the weak learning algorithm. We also note that a read-once majority formula cannot in general be converted into a read-once Boolean formula over the usual {AND, OR, NOT} basis. (For example, a single three-input majority gate cannot be converted into a read-once Boolean formula over this basis; this can be proved, for instance, by enumerating all three-input read-once Boolean formulas.)

It can be shown that the class of logarithmic-depth read-once majority formulas is not learnable in the distribution-free model, modulo assorted cryptographic assumptions. depend on the assumed intractability of factoring Blum integers, inverting RSA functions, and recognizing quadratic residues. See Kearns and Valiant [16] for details. Briefly, this can be proved using a Pitt and Warmuth-style "prediction-preserving reduction" [19] to show that learning read-once majority formulas is at least as hard as learning general Boolean formulas. Our reduction starts with a given Boolean formula which we can assume, without loss of generality, has been converted using standard techniques into an equivalent formula of logarithmic depth. The main idea of the reduction is to replace each OR gate (respectively, AND gate) occurring in this formula with a MAJ gate, one of whose inputs is wired to a distinct variable that, under the target distribution, always has the value 1 (respectively, 0). The resulting majority formula can further be reduced to one that is read-once using the substitution method of Kearns et al. [15]. Finally, combined with Kearns and Valiant's result that Boolean formulas are not learnable (modulo cryptographic assumptions), this shows that majority formulas are also not learnable.

Despite the hardness of this class in the general distribution-free framework, we show that the class is nevertheless exactly identifiable when examples are chosen from the uniform distribution. The algorithm consists of two phases. In the first phase, we determine the relevant variables (i.e., those that occur in the formula), their signs (i.e., whether they are negated or not), and their levels. To achieve this goal, for each variable, we hard wire its value to 1 and estimate the amplification of the induced function at $\frac{1}{2}$ using examples drawn randomly from the uniform distribution on the remaining variables. Here, by "hard wiring" a variable to 1, we really mean that we apply a *filter* that only lets through examples for which that variable is 1. We prove that if the variable is relevant, then with high probability this estimate will be significantly smaller or greater than $\frac{1}{2}$, depending on whether the variable occurs negated or unnegated in the formula; otherwise, this estimate will be near $\frac{1}{2}$. Furthermore, the level of a relevant variable can be determined from the amount by which the amplification of the induced function differs from $\frac{1}{2}$.

In the second phase of the algorithm, we construct the formula. More precisely, we first construct the bottom level of the formula, and then recursively construct the remaining levels. To construct the bottom level of the formula, we begin by finding triples of variables that are inputs to the same bottom-level gate. To do this, for each triple of relevant variables that have the largest level number, we hard wire the three variables to one and again estimate the amplification of the induced function from random examples. We show that we can determine whether the three variables all enter the same bottom-level gate based on this estimate.

Briefly, the recursion works as follows. When constructing level t of the formula, if we find that x_i, x_j, and x_k are inputs to the same level-t gate, then in the recursive call we replace x_i, x_j, and x_k by a level-t metavariable $y \equiv \text{MAJ}(x_i, x_j, x_k)$. Since y is a known subformula, its output on any example can be easily computed and y can be treated like an ordinary variable. Furthermore, since $\frac{1}{2}$ is the fixed point for the amplification function of any read-once majority formula, it follows that y is 1 with probability $\frac{1}{2}$. Thus, for the recursive call we replace all triples of variables that enter level-t gates with metavariables, and we easily obtain our needed source of random examples drawn according to the uniform distribution on the new variable set from the original source of examples.

For the remainder of this section, we explore some of the properties of the amplification function of read-once majority formulas, leading eventually to a proof of the correctness of this algorithm.

LEMMA 3.1. *Let X_1, X_2, and X_3 be three independent Bernoulli variables, each* 1 *with probability p_1, p_2, and p_3, respectively. Then* $\Pr[\text{MAJ}(X_1, X_2, X_3) = 1] = p_1p_2 + p_1p_3 + p_2p_3 - 2p_1p_2p_3$.

Proof. The stated probability is exactly the chance that at least two of the three variables are 1. □

Lemma 3.1 implies that $A_f(\frac{1}{2}) = \frac{1}{2}$ for any read-once majority formula f. Thus, $\frac{1}{2}$ is a fixed point of A_f.

Our approach depends on the fact that the first derivative of A_f is large at $\frac{1}{2}$, meaning that a slight perturbation of $D^{(1/2)}$ (i.e., the uniform distribution) tends to perturb the statistical behavior of the formula sufficiently to allow exact identification. See Fig. 1 for a graph showing the amplification function for balanced read-once majority formulas of various depths.

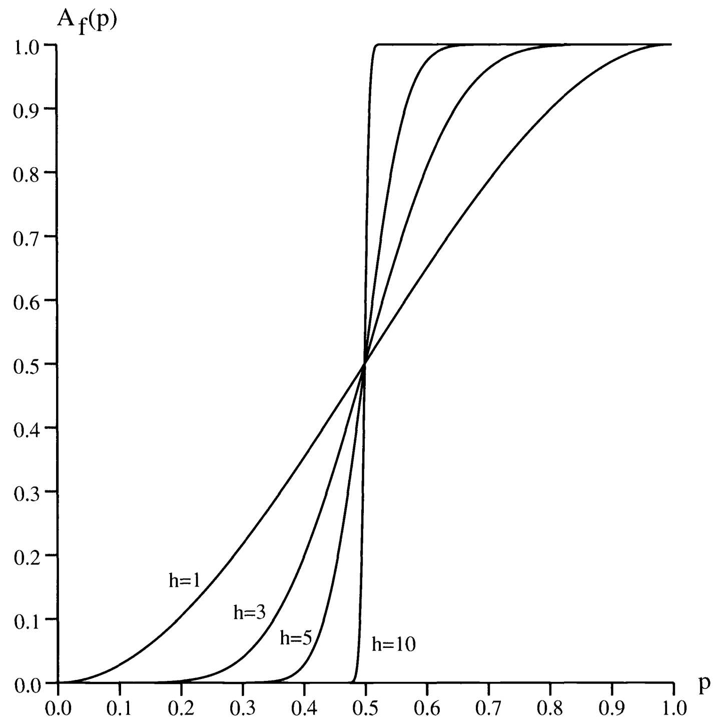


FIG. 1. *The amplification function for read-once majority formulas for complete ternary trees of depth h.*

We perturb $D^{(1/2)}$ by hard wiring a small number of variables to be 1; such perturbations can always be efficiently sampled by simply waiting for the desired variables to be simultaneously set to 1 in a random example from $D^{(1/2)}$.

We begin by considering the effect on the function's amplification of altering the probability with which one of the variables x_j is set to 1. This will be important in the analysis that follows.

LEMMA 3.2. *Let f be a read-once majority formula, and let t be the level of an unnegated variable x_j. Then for $q \in \{0, 1\}$, $A_{f|x_j \leftarrow q}(\frac{1}{2}) = \frac{1}{2} + (\frac{1}{2})^t (q - \frac{1}{2})$.*

Proof. The proof is by induction on t. When $t = 0$, the formula consists of just the variable x_j, and the lemma holds. For the inductive step, let f_1, f_2, and f_3 be the functions computed by the three subformulas obtained by deleting the output gate of f; thus, f is just the majority of f_1, f_2, and f_3. Note that x_j occurs in exactly one of these three subformulas—assume it occurs in the first. Since x_j occurs at level $t - 1$ of this subformula, by the inductive hypothesis, $A_{f_1|x_j \leftarrow q}(1/2) = (1/2) + (1/2)^{t-1}(q - (1/2))$, and since x_j does not occur in the other subformulas, $A_{f_i|x_j \leftarrow q}(\frac{1}{2}) = \frac{1}{2}$ for $i = 2, 3$. From Lemma 3.1, it follows that $A_{f|x_j \leftarrow q}(\frac{1}{2})$ has the stated value, completing the induction. □

It can now be seen how we use the amplification function to determine the relevant variables of f: If x_j is relevant, then the statistical behavior of the output of f changes significantly when x_j is hard wired to 1. Similarly, the sign and the level of each variable can be readily determined in this manner.

THEOREM 3.3. *Let f be a read-once majority formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_j \leftarrow 1}(\frac{1}{2})$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \tau \leq (1/2)^{h+2}$. Then:*

- *x_j is relevant if and only if $|\hat{\alpha} - \frac{1}{2}| > \tau$;*
- *if x_j is relevant, then it occurs negated if and only if $\hat{\alpha} < \frac{1}{2}$;*
- *x_j occurs at level t if and only if $\left||\hat{\alpha} - (1/2)| - (1/2)^{t+1}\right| < \tau$.*

Proof. This proof follows from straightforward calculations using Lemma 3.2. □

Thus, if one estimates the value of the amplification function from a sample whose size is polynomial in 2^h, then with high probability we can determine which variables are relevant, as well as the sign and level of every relevant variable. Specifically, we can apply Chernoff bounds [12] to derive a sample size sufficient to ensure that all the above information is properly computed with high probability. We therefore assume henceforth that the level of every variable has been determined and that, (without loss of generality), all variables are relevant and unnegated.

More problematic is determining exactly how the variables are combined in f. A natural approach is to try hard wiring *pairs* of variables to 1, and to again estimate the amplification of the induced function in the hopes that some structural information will be revealed. The following lemma, that is useful at a later point, shows that this approach fails.

LEMMA 3.4. *Let f be a read-once majority formula, and let x_i and x_j be distinct, unnegated variables that occur at levels t_1 and t_2, respectively. Then*

$$A_{f|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1+1} + \left(\tfrac{1}{2}\right)^{t_2+1},$$

regardless of the depth d of $\lambda = \Gamma(x_i, x_j)$.

Proof. By induction on d. Let f_1, f_2, and f_3 be the three subformulas of f that are inputs to the output gate so that $f = \text{MAJ}(f_1, f_2, f_3)$.

If $d = 0$, then λ is the output gate, and x_i and x_j occur in two of the subformulas (say, f_1 and f_2, respectively). From Lemma 3.2, it follows that

$$A_{f_k|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_k}$$

for $k = 1, 2$, and, since neither x_i nor x_j is relevant to f_3, $A_{f_3|x_i,x_j \leftarrow 1}(\frac{1}{2}) = \frac{1}{2}$. The stated value for $A_{f|x_i,x_j \leftarrow 1}(\frac{1}{2})$ follows then from Lemma 3.1.

If $d > 0$, then λ is a gate occurring in one of the subformulas (say, f_1) at level $d - 1$ of the subformula. By inductive hypothesis,

$$A_{f_1|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1} + \left(\tfrac{1}{2}\right)^{t_2}.$$

Also, $A_{f_k|x_i,x_j \leftarrow 1}(\frac{1}{2}) = \frac{1}{2}$ for $k = 2, 3$. The stated value for $A_{f|x_i,x_j \leftarrow 1}(\frac{1}{2})$ again follows from Lemma 3.1. □

Thus, if two relevant variables are hard-wired to 1, no information is obtained by knowing the value of the amplification function. That is, the amplification function is independent of the level at which the two variables meet.

Therefore, we instead consider what happens when three relevant variables of the same level are fixed to 1. In fact, it turns out to be sufficient to do so for triples of variables, all of which occur at the bottom level of the formula. We show that by doing so we can determine the full structure of the formula.

For each triple x_i, x_j, and x_k all occurring at level t, there are essentially two cases to consider; either

1. the triple x_i, x_j, x_k does not meet in the formula; or
2. the three variables x_i, x_j, and x_k meet at the gate $\Gamma(x_i, x_j, x_k)$. We divide this case into two subcases:
   (a) x_i, x_j, and x_k are inputs to the same gate so that $\Gamma(x_i, x_j, x_k)$ occurs at level $t-1$; or
   (b) $\Gamma(x_i, x_j, x_k)$ occurs at some level $d < t-1$.

We are interested in separating Case 2(a) from the other cases by estimating the amplification of the function when all three variables are hard wired to 1. This is sufficient to reconstruct the structure of the formula: If we can find three variables that are inputs to some gate λ (and there must always exist such a triple), then we can essentially replace the subformula consisting of the three variables and the gate λ by a new metavariable whose value can easily be determined from the values of the original three variables. Furthermore, since $\frac{1}{2}$ is a fixed point for all read-once majority formulas, the metavariables' statistics will be the same as those of the original variables. Thus, the total number of variables is reduced by two, and the rest of the formula's structure can be determined recursively.

The following two lemmas analyze the amplification of the function when three variables are hard wired to 1 in both of the above cases. We begin with Case 2.

LEMMA 3.5. *Let f be a read-once majority formula. Let x_i, x_j, and x_k be three distinct, unnegated inputs that occur at levels t_1, t_2, and t_3, respectively, and which meet at gate $\lambda = \Gamma(x_i, x_j, x_k)$. Let d be the level of λ. Then*

$$A_{f|x_i,x_j,x_k \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1+1} + \left(\tfrac{1}{2}\right)^{t_2+1} + \left(\tfrac{1}{2}\right)^{t_3+1} - \left(\tfrac{1}{2}\right)^{t_1+t_2+t_3-2d-1}.$$

Proof. The proof is by induction on d. As in the preceding lemmas, suppose $f = \mathrm{MAJ}(f_1, f_2, f_3)$. If $d = 0$, then λ is the output gate of f and, without loss of generality, x_i, x_j, and x_k occur one each in f_1, f_2, and f_3, respectively. From Lemma 3.2, $A_{f_r|x_i,x_j,x_k \leftarrow 1}(1/2) = (1/2) + (1/2)^{t_r}$, for $r = 1, 2, 3$. The stated value for $A_{f|x_i,x_j,x_k \leftarrow 1}(\frac{1}{2})$ follows from Lemma 3.1.

If $d > 0$, then one of the subformulas (say, f_1) contains λ at depth $d-1$. By inductive hypothesis,

$$A_{f_1|x_i,x_j,x_k \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1} + \left(\tfrac{1}{2}\right)^{t_2} + \left(\tfrac{1}{2}\right)^{t_3} - \left(\tfrac{1}{2}\right)^{t_1+t_2+t_3-2d-2},$$

and of course, $A_{f_r|x_i,x_j,x_k \leftarrow 1}(\frac{1}{2}) = \frac{1}{2}$ for $r = 2, 3$. The proof is completed by again applying Lemma 3.1. □

So, unlike the situation in which only two variables are hard wired to 1, here the value of the amplification function depends on the level of the formula at which the three variables meet. However, it may be the case that x_i, x_j, and x_k do not meet at all (i.e., we may be in Case 1). The next lemma considers this case.

LEMMA 3.6. *Let f be a read-once majority formula. Let x_i, x_j, and x_k be three distinct, unnegated inputs that occur at levels t_1, t_2, and t_3, respectively, and for which $\lambda' = \Gamma(x_i, x_j) \neq \Gamma(x_i, x_j, x_k) = \lambda$. Then*

$$A_{f|x_i,x_j,x_k \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1+1} + \left(\tfrac{1}{2}\right)^{t_2+1} + \left(\tfrac{1}{2}\right)^{t_3+1},$$

regardless of the levels d and d' of gates λ and λ'.

Proof. The proof is by induction on d. As before, assume that $f = \mathrm{MAJ}(f_1, f_2, f_3)$. If $d = 0$, then λ is the output gate, λ' occurs (say) in f_1, and x_k in f_2. From Lemma

3.4, $A_{f_1|x_i,x_j,x_k\leftarrow 1}(1/2) = (1/2) + (1/2)^{t_1} + (1/2)^{t_2}$, and from Lemma 3.2, $A_{f_2|x_i,x_j,x_k\leftarrow 1}(1/2) = (1/2) + (1/2)^{t_3}$. Also, $A_{f_3|x_i,x_j,x_k\leftarrow 1}(1/2) = (1/2)$. Lemma 3.1 then implies the stated value for $A_{f|x_i,x_j,x_k\leftarrow 1}(1/2)$.

If $d > 0$, then λ occurs, say, in f_1. By the inductive hypothesis, $A_{f_1|x_i,x_j,x_k\leftarrow 1}(1/2) = (1/2) + (1/2)^{t_1} + (1/2)^{t_2} + (1/2)^{t_3}$, and clearly $A_{f_r|x_i,x_j,x_k\leftarrow 1}(1/2) = (1/2)$ for $r = 2, 3$. An application of Lemma 3.1 completes the induction. □

Combining these lemmas, we can show that Case 2(a) can be separated from the other cases by estimating the function's amplification with triples of variables hard wired to 1.

THEOREM 3.7. *Let f be a read-once majority formula. Let x_i, x_j, and x_k be three distinct, unnegated, level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x,x_j,x_k\leftarrow 1}(1/2)$ for which $|\hat{\alpha} - \alpha| < \tau \leq 3(1/2)^{t+4}$. Then x_i, x_j, and x_k are inputs to the same gate of f if and only if $\hat{\alpha} < (1/2) + (1/2)^t + \tau$.*

Proof. If Case 2(a) applies, then Lemma 3.5 implies that $\alpha = (1/2) + (1/2)^t$. Otherwise, if either Case 1 or 2(b) applies, then Lemmas 3.5 and 3.6 imply that $\alpha \geq (1/2) + (1/2)^t + 3(1/2)^{t+3}$. The theorem follows immediately. □

We are now ready to state the main result of this section.

THEOREM 3.8. *There exists an algorithm with the following properties: Given h, n, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0, 1\}^n$ and labeled by any read-once majority formula f of depth at most h on n variables, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O(4^h \cdot \log(n/\delta))$, and its time complexity is $O(4^h \cdot (r^3 + n) \cdot \log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Proof. First, for each variable x_i, estimate the function's amplification with x_i hard-wired to 1. (We will ensure that, with high probability, this estimate is within $(1/2)^{h+2}$ of the true amplification.) It follows from Theorem 3.3 that after this phase of the algorithm, with high probability we know which variables are relevant, and we know the sign and depth of each relevant variable. (So, we assume from now on that the formula is monotone.)

In the second phase of the algorithm, we build the formula level by level from bottom to top. To build the bottom level, for all triples of variables x_i, x_j, x_k that enter the bottom level, we estimate the amplification with x_i, x_j, and x_k hard wired to 1. (We will ensure that, with high probability, this estimate is within $3(1/2)^{h+4}$ of the true amplification.) It follows from Theorem 3.7 that we can determine which variables enter the same bottom-level gates.

We want to recurse to compute the other levels; however, we cannot hard wire too many variables without the filter requiring too many examples. The key observation is that on examples drawn from the uniform distribution, the output of any subformula is 1 with probability $\frac{1}{2}$. Thus, the inputs into any level are in fact distributed according to a uniform distribution. Since we compute the formula from bottom to top, the filter can just compute the value for the *known* levels to determine the inputs to the level currently being learned. Our algorithm is described in Fig. 2.

Given that the estimates for the formula's amplification have the needed accuracy, the proof of correctness follows from Theorems 3.3 and 3.7. To compute the actual sample size needed to make these estimates, we use Hoeffding's inequality [12] (also known as a form of Chernoff bounds) as stated in the following lemma.

LEMMA 3.9 (Hoeffding's Inequality). *Let $X_1, \ldots, X_m$ be a sequence of m independent Bernoulli trials, each succeeding with probability p. Let $S = X_1 + \cdots + X_m$ be the random variable describing the total number of successes. Then for $0 \leq \gamma \leq 1$, the following holds:*

$$\Pr[|S - pm| > \gamma m] \leq 2e^{-2m\gamma^2}.$$

```
LearnMajorityFormula($n, h, \delta$)
    $m \leftarrow 228 \cdot 4^h \ln(8n^3/\delta)$
    $E \leftarrow \{m$ labeled examples from $D^{(1/2)}\}$
    $X \leftarrow \emptyset$
    for $1 \leq i \leq n$
        $E' \leftarrow$ examples from $E$ for which $x_i = 1$
        $\hat{\alpha} \leftarrow$ fraction of $E'$ that are positive
        if $|\hat{\alpha} - \frac{1}{2}| > (\frac{1}{2})^{h+2}$ then
            if $\hat{\alpha} > \frac{1}{2}$ then $X \leftarrow X \cup \{x_i\}$
                     else $X \leftarrow X \cup \{\bar{x}_i\}$
            $t(x_i) \leftarrow$ compute-level($\hat{\alpha}, h$)
    BuildFormula($h, X, E$)

BuildFormula($t, X, E$)
    if $t = 0$ then the target formula is the only variable in $X$
    else
        for all distinct triples $x_i, x_j, x_k \in X$ for which $t(x_i) = t(x_j) = t(x_k) = t$
            $E' \leftarrow$ examples from $E$ for which $x_i = x_j = x_k = 1$
            $\hat{\alpha} \leftarrow$ fraction of $E'$ that are positive
            if $\hat{\alpha} < \frac{1}{2} + (\frac{1}{2})^t + 3(\frac{1}{2})^{t+4}$ then
                let $y \equiv \text{MAJ}(x_i, x_j, x_k)$ be a new variable
                $t(y) \leftarrow t - 1$
                $X \leftarrow (X \cup \{y\}) - \{x_i, x_j, x_k\}$
        BuildFormula($t - 1, X, E$)
```

FIG. 2. *Algorithm for exactly identifying read-once majority formulas of depth h. Procedure* compute-level($\hat{\alpha}, h$) *computes the level associated with $\hat{\alpha}$ as given by Theorem* 3.3.

In the first phase of the algorithm, for each variable x_i, we need a good estimate $\hat{\alpha}$ of $\alpha = A_{f|x_i \leftarrow 1}(\frac{1}{2})$; specifically, we require that the chosen sample be sufficiently large that $|\alpha - \hat{\alpha}| < 2^{-(h+2)}$ with probability at least $1 - \delta/2n$. Then every such estimate for the n variables will have the needed accuracy with probability at least $1 - \delta/2$.

Using Hoeffding's inequality it can be shown that a filtered sample (i.e., the sample after all examples are removed for which $x_i = 0$) of size $m_1 > 8 \cdot 4^h \ln(8n/\delta)$ is sufficiently large to ensure that the estimate $\hat{\alpha}$ has the needed accuracy with probability at least $1 - \delta/4n$. Again using Hoeffding's inequality, it can be shown that if we draw an (unfiltered) sample of size at least $\max\{4m_1, 8\ln(4/\delta)\}$ then with probability at least $1 - \delta/4n$, at least one quarter of the examples chosen will be such that $x_i = 1$, and thus, the filtered sample will have size at least m_1. So by using a sample of size exceeding $32 \cdot 4^h \ln(8n/\delta)$, all of the estimates satisfy the requirements for the first phase of the algorithm with probability at least $1 - \delta/2$.

In the second phase, we require good estimates of the formula's amplification when triples of variables are hard wired. In fact, we need such estimates not only when ordinary variables are hard wired, but also when we hard wire metavariables. Note that, assuming all estimates have the needed accuracy, every (meta-)variable added to the set X in Fig. 2 in fact computes some subformula g of f. Thus, for every triple of subformulas g_1, g_2, and g_3 of f, our algorithm requires an estimate $\hat{\alpha}$ of α, the amplification of f at $\frac{1}{2}$, given that the output of each subformula g_1, g_2, and g_3 is fixed to the value 1. Since a read-once majority formula on n variables has at most $3n/2$ subformulas (since it has at most $n/2$ MAJ gates), we require a sample sufficiently large that $|\alpha - \hat{\alpha}| < 3(1/2)^{h+4}$ with probability at least $1 - 4\delta/27n^3$. The chance that all of the (at most $(3n/2)^3$) estimates have the needed accuracy is then at least $1 - \delta/2$. The analysis is similar to that given above, yielding the sample size stated in the figure.

Finally, the time complexity follows from the fact that `LearnMajorityFormula` makes n estimates from the sample and `BuildFormula` makes $O(r^3)$ estimates from the sample. □

Note that our algorithm's sample complexity has only a logarithmic dependence on the number of irrelevant attributes. Also, it follows immediately from Theorem 3.8 that any read-once majority formula of depth $O(\log n)$ can be exactly identified in polynomial time.

Finally, we note that our algorithm can be modified to work without receiving a bound for the height of the formula as input; the time and sample complexity only increase by a factor of two. The idea is to guess an initial value of $h = 1$ and to increment our guess each time the algorithm fails; it can be shown that, if the formula's height is greater than our current guess, then this fact will become evident by our algorithm's inability to successfully construct a formula. (Specifically, the algorithm `BuildFormula` in Fig. 2 will reach a point at which there remain level-t variables in X, but no three remaining level-t variables are immediate inputs to the same gate.)

4. Exact identification of read-once positive NAND formulas. In this section we use the properties of the amplification function to obtain a polynomial-time algorithm that, with high probability, exactly identifies any read-once positive NAND formula of logarithmic depth from $D^{(\psi)}$ where ψ is the constant $(\sqrt{5}-1)/2 \approx 0.618$. Note that $\psi = 1/\phi = \phi - 1$, where ϕ is the golden ratio.

As noted earlier, the class of read-once positive NAND formulas is equivalent to the class of read-once formulas constructed from alternating levels of OR/AND gates, starting with an OR gate at top level, and with the additional condition that each variable is negated if and only if it enters an OR gate.

We show that this class of formulas is learnable when examples are chosen from a distribution in which each variable is 1 with probability ψ. The basic structure of the algorithm is just like that of the preceding algorithm for identifying read-once majority formulas. In the first phase of the algorithm, we determine the relevant variables and their depths by hard wiring each variable to 0, and estimating the amplification of the induced function at ψ using random examples from $D^{(\psi)}$. In the second phase of the algorithm, we construct the formula by finding pairs of variables that are direct inputs to a bottom-level gate. Here, we show that this is possible by hard wiring pairs of variables to 0 and estimating the function's amplification. After learning the structure of the bottom level of the formula, we again are able to construct the remaining levels recursively.

Since the techniques used in this section are so similar to those in § 3, the proofs of the lemmas and theorems have been omitted. Most of the lemmas can be proved by simple induction arguments as before.

We turn now to a discussion of some of the properties of the amplification function of read-once positive NAND formulas; these lead to a proof of the correctness of our algorithm.

LEMMA 4.1. *Let X_1 and X_2 be independent Bernoulli variables, each* 1 *with probability p_1 and p_2, respectively. Then* $\Pr[\text{NAND}(X_1, X_2) = 1] = 1 - p_1 p_2$.

It is easily verified that $1 - \psi^2 = \psi$, and thus that ψ is a fixed point of the amplification function A_f whenever f is a read-once positive NAND formula. Once again, our approach depends on the fact that slight perturbations of $D^{(\psi)}$ tend to perturb the statistical behavior of the formula sufficiently to allow exact identification.

LEMMA 4.2. *Let f be a read-once positive* NAND *formula, and let t be the level of some variable x_j. Then $A_{f|x_j \leftarrow q}(\psi) = \psi + (q - \psi)(-\psi)^t$.*

Thus, hard wiring an even-leveled input to 0 decreases the amplification while hard wiring an odd-leveled input to 0 increases the amplification. To give some intuition explaining this behavior, consider the correspondence described above between read-once positive NAND formulas and leveled OR/AND formulas. An even-leveled input corresponds to an input to an AND gate and thus hard wiring that input to 0 clearly decreases the amplification. However, an odd-leveled input corresponds to an input that is first negated and then fed to an OR gate; thus, this case corresponds to hard wiring the input to an OR gate to 1 which clearly increases the amplification function.

As we saw in the last section, the amplification function can be used to determine the relevant variables of f: if x_j is relevant then the statistical behavior of the output of f changes significantly when x_j is hard wired to 0. Similarly, the level of each variable can be computed in this manner.

THEOREM 4.3. *Let f be a read-once positive* NAND *formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_j \leftarrow 0}(\psi)$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \tau \leq \psi^{h+1}/2$. Then*:

- *x_j is relevant if and only if $|\hat{\alpha} - \psi| > \tau$;*
- *x_j occurs at level t if and only if $|\psi + (-\psi)^{t+1} - \hat{\alpha}| < \tau$.*

We next consider the effect on the amplification function of hard wiring two inputs. Unlike the case of majority formulas, measuring the amplification of the function when pairs of variables are hard wired to 0 reveals a great deal of information about the structure of the formula. In particular, the value of the amplification function when two level-t variables x_i and x_j are hard wired to 0 depends critically on the depth of $\Gamma(x_i, x_j)$.

LEMMA 4.4. *Let f be a read-once positive* NAND *formula, and let x_i and x_j be two distinct variables that occur at levels t_1 and t_2, respectively, and for which $\lambda = \Gamma(x_i, x_j)$ is at level d. Then*

$$A_{f|x_i,x_j \leftarrow 0}(\psi) = \psi + (-\psi)^{t_1+1} + (-\psi)^{t_2+1} - (-\psi)^{t_1+t_2-d}.$$

Using the same ideas as in the last section, it can now be proved that, given a good estimate of the amplification function, we can determine which variables meet at bottom-level gates.

THEOREM 4.5. *Let f be a read-once positive* NAND *formula. Let x_i and x_j be two level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_i,x_j \leftarrow 0}(\psi)$ for which $|\hat{\alpha} - \alpha| < \tau \leq \psi^{t+3}/2$. Then x_i and x_j are inputs to the same level-$(t-1)$ gate of f if and only if $|\psi + (-\psi)^{t+1} - \hat{\alpha}| < \tau$.*

We are now ready to state the main result of this section.

THEOREM 4.6. *Let $\psi = 1/\phi = (\sqrt{5} - 1)/2$. Then there exists an algorithm with the following properties: Given h, n, $\delta > 0$, and access to examples drawn from the distribution $D^{(\psi)}$ on $\{0, 1\}^n$, and labeled by any read-once positive* NAND *formula f of depth at most h on n variables, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O(\phi^{2h} \cdot \log(n/\delta))$, and its time complexity is $O(\phi^{2h} \cdot (r^2 + n) \cdot \log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Our algorithm is modifying both `BuildFormula` and `LearnMajorityFormula` in a straightforward fashion. The proof that this algorithm is correct follows from the preceding lemmas and theorems, and is similar to the proof of Theorem 3.8.

As before, it follows immediately that any read-once positive NAND formula of depth at most $O(\log n)$ can be exactly identified in polynomial time.

5. Short universal identification sequences. In this section we describe an interesting consequence of our results. Observe that if we regard our algorithms' use of a *fixed* distribution as a form of "random" membership queries, then it is apparent that these queries are *nonadaptive*; each query is independent of all previous answers. In other words, our algorithms pick all membership queries before seeing the outcome of any. From this observation we can apply our results to prove the existence of polynomial-length *universal identification sequences* for classes of formulas; that is, fixed sequences of instances that distinguish all concepts from one another.

More formally, we define an *instance sequence* to be an unlabeled sequence of instances, and an *example sequence* to be a labeled sequence of instances. Let $\mathcal{C}_n$ be a concept class. We say an instance sequence S *distinguishes* a concept c if the example sequence obtained by labeling S according to c distinguishes c from all other concepts in $\mathcal{C}_n$; that is, c is the only concept consistent with the example sequence so obtained. A *universal identification sequence* for a concept class $\mathcal{C}_n$ is an instance sequence that distinguishes *every* concept $c \in \mathcal{C}_n$.

In the language used in the related work of Goldman and Kearns [8], a sequence that distinguishes c would be called a *teaching sequence* for c. Thus, in their terminology, a universal identification sequence is one that acts as a teaching sequence for every $c \in \mathcal{C}_n$. See also the related work of Shinohara and Miyano [22].

The following theorem gives general conditions for when a deterministic exact identification algorithm implies the existence of a polynomial-length universal identification sequence. We then apply this theorem to our algorithms of the previous sections.

THEOREM 5.1. *Let $\mathcal{C}_n$ be a concept class with cardinality $|\mathcal{C}_n| \leq 2^{p(n)}$, where $p(n)$ is some polynomial. Let A be a deterministic algorithm that, given $\delta > 0$ and random, labeled examples drawn from some fixed distribution D, exactly identifies any $c \in \mathcal{C}_n$ with probability exceeding $1 - \delta$. Furthermore, suppose that the sample complexity of A is $q(n) \log^k(1/\delta)$ for some polynomial $q(n)$ and constant k. Then there exists a polynomial-length universal identification sequence for $\mathcal{C}_n$. Specifically, there exists such a sequence of length $q(n) \cdot (p(n))^k$.*

Proof. The proof uses a standard probabilistic argument. Fix $c \in \mathcal{C}_n$, and let S be the random example sequence drawn by algorithm A. Since algorithm A achieves exact identification of c with probability exceeding $1 - \delta$ we have that the probability, taken over all random example sequences for c, that A fails to exactly identify the *particular* target concept c is less than δ. Letting $\delta = 2^{-p(n)}$ it follows that the probability, taken over all random instance sequences, that A fails to identify *any* target concept in $\mathcal{C}_n$ is less than $|\mathcal{C}_n|\delta \leq 1$. Thus, with positive probability, an instance sequence of length $q(n)(p(n))^k$ drawn randomly from D causes A to exactly identify any $c \in \mathcal{C}_n$. Call such a sequence *good*. Therefore there must exist some good instance sequence S of this length.

We now show that a good S *distinguishes* all $c \in \mathcal{C}_n$. For $c \in \mathcal{C}_n$, let S_c be the example sequence obtained by labeling S according to c. Suppose some $c' \in \mathcal{C}_n$ is consistent with S_c so that $S_c = S_{c'}$. Since S is a good sequence and since A is a deterministic exact identification algorithm, it follows that on input S_c, A must output c and on input $S_{c'} = S_c$, A must output c'. Clearly this can only be true if $c = c'$. Thus c must be the only concept in $\mathcal{C}_n$ that is consistent with S_c. $\square$

Applying this theorem to our exact identification algorithms, we obtain the following corollary.

COROLLARY 5.2. *There exist polynomial-length universal identification sequences for the classes of logarithmic-depth read-once majority formulas and logarithmic-depth read-once positive* NAND *formulas.*

6. Handling random misclassification noise. Because the algorithms described in §§3 and 4 are statistical in nature, they are easily modified to handle a considerable amount of noise. In this section, we describe a robust version of our algorithm for learning logarithmic-depth read-once majority formulas. Although omitted, a similar (though slightly more involved) algorithm can be derived for NAND formulas.

Our algorithm is able to handle a kind of random misclassification noise that is similar, but slightly more general than that considered by Angluin and Laird [2] and Sloan [23]. Specifically, the output of the target formula is "flipped" with some fixed probability that may depend on the formula's output. Thus, if the true, computed output of the formula is 0, then the learner sees 0 with probability $1 - \eta_0$, and 1 with probability η_0, for some quantity η_0. Similarly, a true output of 1 is observed to be 0 with probability η_1 and 1 with probability $1 - \eta_1$. When $\eta_0 = \eta_1$, this noise model is equivalent to that considered by Angluin and Laird, and Sloan. Note that when $\eta_0 + \eta_1 = 1$, outputs of 0 or 1 are entirely indistinguishable in an information-theoretic sense. Moreover, we can assume without loss of generality that $\eta_0 + \eta_1 \leq 1$ by symmetry of the behavior of the formula f with its negation $\neg f$.

If we regard our algorithm's use of a fixed distribution as a form of membership query, we can also handle large rates of misclassification noise in the queries. Here the formulation of a meaningful noise model is more problematic. In particular, we wish to disallow the uninteresting technique of repeatedly querying a particular instance in order to obtain its true classification with overwhelming probability. Thus, we consider a model in which noisy labels are *persistent:* For each instance x, on the first query to x, the true output of the target concept is computed and is reversed with probability η_0 or η_1, according to whether the true output is 0 or 1 (as described above). However, on all subsequent queries to x, the label returned is the same as the label returned with the first query to x. A natural interpretation of such persistent noise is that of a teacher who is simply wrong on certain instances, and cannot be expected to change his mind with repeated sampling. This kind of persistent noise is not a problem for our algorithms because, when n is large, the algorithm is extremely unlikely to query the same instance twice.

Our algorithm assumes that $\eta_0 + \eta_1$ is bounded away from 1 so that $\eta_0 + \eta_1 \leq 1 - \rho$ for some known positive quantity ρ. The error rates themselves, η_0 and η_1, are assumed to be unknown. Our algorithm exactly identifies the target formula with high probability in time polynomial in all of the usual parameters, and $1/\rho$.

Our robust algorithm has a similar structure to that of the algorithm described previously for the noise-free case: The algorithm begins by determining the relevance and sign of each variable. However, it is not clear at this point how the level of each variable might be ascertained in the presence of noise. Nevertheless, it turns out to be possible to find three bottom-level variables that are inputs to the same gate. As before, once such a triple has been discovered, the remainder of the formula can be identified recursively.

To start with, note that if p is the probability that a 1 is output by the target formula f under some distribution on $\{0, 1\}^n$, then the probability that a 1 is observed by the learner is

$$p(1 - \eta_1) + (1 - p)\eta_0 = p(1 - \eta_0 - \eta_1) + \eta_0.$$

Thus, $\tilde{A}_f(p) = A_f(p) \cdot (1 - \eta_0 - \eta_1) + \eta_0$ is the probability that a 1 is observed when each input is 1 with probability p. Under the uniform distribution, a 1 is observed with probability $\xi = \tilde{A}_f(\frac{1}{2})$. Since η_0 and η_1 are unknown, ξ is unknown as well. However, an accurate estimate $\hat{\xi}$ (say, within $\Theta(\rho/2^h)$ of ξ) can be efficiently obtained in the usual manner by sampling.

The next lemma shows that a variable x_i's relevance and sign can be determined by hard wiring it to 1 and comparing $\hat{\xi}$ to an estimate of the value $\alpha = \tilde{A}_{f|x_i \leftarrow 1}(\frac{1}{2})$.

LEMMA 6.1. *Let f be read-once majority formula of depth h. Let $\hat{\xi}$ and $\hat{\alpha}$ be estimates of $\xi = \tilde{A}_f(\frac{1}{2})$ and $\alpha = \tilde{A}_{f|x_j \leftarrow 1}(\frac{1}{2})$, for some variable x_j. Assume $|\alpha - \hat{\alpha}| < \tau$ and $|\xi - \hat{\xi}| < \tau$ for some $\tau \leq \rho/2^{h+3}$. Then*:

- *x_j is relevant if and only if $|\hat{\alpha} - \hat{\xi}| > 2\tau$;*
- *if x_j is relevant, then it occurs negated if and only if $\hat{\alpha} < \hat{\xi}$.*

Proof. Note that $\alpha - \xi = (1 - \eta_0 - \eta_1)(A_{f|x_j \leftarrow 1}(\frac{1}{2}) - \frac{1}{2})$. The lemma then follows from Lemma 3.2, and by noting that $1 - \eta_0 - \eta_1 \geq \rho$. □

More difficult is the problem of determining the level of each variable since η_0 and η_1 are unknown. Nevertheless, it turns out to be possible to identify the formula without first determining the level of each variable. In particular, we can determine a triple of variables that are inputs to the same bottom-level gate. As described in §3, once this is done, the three variables can be replaced by a metavariable, and the rest of the formula can be constructed recursively. Thus, to complete the algorithm, we need only describe a technique for finding such a triple.

From the comments above, we can assume, without loss of generality, that all variables are relevant and unnegated. The key point, to be proved, is the following: $\tilde{A}_{f|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2})$ is minimized over triples x_i, x_j, and x_k whenever the three variables are inputs to the same bottom-level gate. Thus, such a triple can be found by estimating $\tilde{A}_{f|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2})$ for each triple and choosing the one with the smallest estimated value.

LEMMA 6.2. *Let f be a monotone, read-once majority formula of depth h. For all triples of distinct indices i, j, and k, let $\hat{\alpha}_{ijk}$ be an estimate of $\alpha_{ijk} = \tilde{A}_{f|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2})$, and assume that $|\alpha_{ijk} - \hat{\alpha}_{ijk}| < \tau \le 3\rho/2^{h+4}$. Suppose that $\hat{\alpha}_{qrs} = \min\{\hat{\alpha}_{ijk} : i, j, k \text{ distinct}\}$. Then x_q, x_r and x_s are bottom-level variables that are inputs to the same gate.*

Proof. From Lemma 3.5, if x_i, x_j, and x_k are bottom-level variables that are inputs to the same gate, then

$$\alpha_{ijk} = (1 - \eta_0 - \eta_1)\left(\tfrac{1}{2} + \left(\tfrac{1}{2}\right)^h\right) + \eta_0.$$

Otherwise, Lemmas 3.5 and 3.6 imply that

$$\alpha_{ijk} \ge (1 - \eta_0 - \eta_1)\left(\tfrac{1}{2} + \left(\tfrac{1}{2}\right)^h + 3\left(\tfrac{1}{2}\right)^{h+3}\right) + \eta_0.$$

Furthermore, since $1 - \eta_0 - \eta_1 \ge \rho$ we get that

$$\alpha_{ijk} \ge (1 - \eta_0 - \eta_1)\left(\tfrac{1}{2} + \left(\tfrac{1}{2}\right)^h\right) + \eta_0 + 3\rho/2^{h+3}.$$

Since each $\hat{\alpha}_{ijk}$ is accurate to within $3\rho/2^{h+4}$, it follows that $\hat{\alpha}_{qrs}$ can be minimal only if x_q, x_r, and x_s are bottom-level inputs to the same gate. □

Thus, Lemma 6.2 gives a technique for finding bottom-level inputs to the same gate and, as previously mentioned, the remainder of the formula can be constructed recursively as in § 3. We thus obtain the main result of this section.

THEOREM 6.3. *There exists an algorithm with the following properties: Given h, n, $\rho > 0$, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0, 1\}^n$, labeled by a read-once majority formula f of depth at most h on n variables, and misclassified with probabilities η_0 and η_1 (as described above) for $\eta_0+\eta_1 \le 1-\rho$, the algorithm exactly identifies f with probability at least $1-\delta$. The algorithm's sample complexity is $O((4^h/\rho^2)\cdot\log(n/\delta))$, and its time complexity is $O((4^h/\rho^2)\cdot(n + r^3)\cdot\log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Finally, we comment that our algorithms can be extended to handle a modest amount of *malicious noise*. In this model, first considered by Kearns and Li [14], an adversary is allowed to corrupt each example in any manner he chooses (both the labels and the variable settings) with probability η. We can show that the algorithm described in § 3 for majority formulas can handle malicious error rates as large as $\Theta(2^{-h})$ where h is the height of the target formula. Thus, for logarithmic-depth formulas, we can handle malicious error rates up to an inverse polynomial in the number of relevant variables. Similar results also hold for NAND formulas.

The extension of the algorithm to handle malicious noise is straightforward. The algorithm of § 3 depends only on accurate estimates of the amplification of f when some of f's inputs are hard wired. For instance, in the first phase of the algorithm, we need to estimate, for each input x_i, the amplification $\alpha = A_{f|x_i \leftarrow 1}(\frac{1}{2})$. This quantity is really just the conditional probability

$$\Pr[f = 1 \mid x_i = 1] = \frac{\Pr[f = 1 \wedge x_i = 1]}{\Pr[x_i = 1]} = 2 \cdot \Pr[f = 1 \wedge x_i = 1]$$

where the probabilities are computed with respect to the uniform distribution. Thus, we can compute an estimate of α using an estimate for $\beta = \Pr[f = 1 \wedge x_i = 1]$. Note that malicious noise can affect this probability β by at most an additive factor of η; that is, the chance that $f = 1$ and $x_i = 1$ (in the presence of malicious noise) is at least $\beta - \eta$ and at most $\beta + \eta$.

Thus, using Hoeffding's inequality (Lemma 3.9), an estimate of β that is accurate to within $\eta + \tau$ can be obtained from a sample of size polynomial in $1/\tau$ (with high probability). Such an estimate for β yields an estimate for α that is accurate to within $2(\eta + \tau)$.

A similar argument can be made for computing the estimates required in the second phase of the algorithm. Since Theorems 3.3 and 3.7 show that the required estimates need only be accurate to within $3/2^{h+4}$, it follows that a malicious error rate of, say, one-sixteenth this amount can be tolerated without increasing the algorithm's complexity by more than constant factors.

7. Learning unbounded-depth formulas. In this final section, we describe extensions of our algorithms to learn formulas of unbounded depth in Valiant's PAC model with respect to specific distributions. As in the last section, we focus only on majority formulas, omitting the similar application of these techniques to NAND formulas.[1]

For formulas of unbounded depth, exact identification from the uniform distribution in polynomial time is too much to ask; for purely information-theoretic reasons, at least $\Omega(2^h)$ examples must be drawn from the uniform distribution to exactly identify a majority formula of depth h. This can be proved by showing (say, by induction on h) that if x_i occurs at level h of formula f, then 2^{-h} is the probability that an instance is chosen for which the output of f depends on x_i (i.e., for which f's output changes if x_i is flipped). Thus, $\Omega(2^h)$ random examples are needed simply to determine, for example, whether x_i occurs negated or unnegated.

Therefore, to handle arbitrarily deep formulas, we must relax our requirement of exact identification. Instead, we adopt Valiant's criterion of obtaining a good *approximation* of the target concept (with high probability). As before, our algorithms do not work for all distributions, just the fixed-point distribution. We describe an algorithm that, given $\epsilon, \delta > 0$, and access to random examples of the target majority formula drawn from the uniform distribution, outputs with probability $1 - \delta$ an ϵ-*good* hypothesis; that is, one that agrees with the target formula on a randomly chosen instance from the uniform distribution with probability at least $1 - \epsilon$. Furthermore, the running time is polynomial in $1/\delta$, $1/\epsilon$, and the number of variables n.

We begin by briefly discussing the main ideas of the algorithm. First, as noted above, variables that occur deep in the formula are unimportant in the sense that their values are unlikely to influence the formula's output on a randomly chosen instance. Intuitively, we would like to take advantage of this fact by somehow treating such variables as irrelevant. However, they cannot be simply deleted from the formula without leaving "holes" that must in some way be handled.

We therefore introduce the notion of a *partially visible function*. This is a function on a set of *visible* variables whose values can be observed by the learner, and a set of *hidden*

[1]Alternatively, the algorithm recently described by Schapire [21] could be used to PAC learn unbounded-depth NAND formulas. His algorithm is more general but less efficient than the approach described in this section.

variables that are not observable. With respect to a distribution on the set of assignments to the hidden variables, we say that two partially visible Boolean functions are *equivalent* if, for all assignments to the visible variables, the probabilities are the same that each function evaluates to 1 (where the probabilities are taken over random assignments to the hidden variables). In other words, the behaviors of the two functions are indistinguishable with respect to the visible variables.

Thus, we handle all deep variables by regarding them as hidden variables, and the target formula as one that is partially visible. In particular, *insignificant* variables—those that occur below level $h = \lceil \log(n/2\epsilon) \rceil$—are considered hidden and their actual values ignored. We call the partially visible formula obtained from the target formula f in this manner the *truncated target*.

Our algorithm works by exactly identifying the truncated target; that is, by constructing a partially visible formula f' that is equivalent to it (in the sense described above, with respect to the uniform distribution). It will be shown that f and f' agree on a randomly chosen instance with probability at least $1 - \epsilon$, and therefore f' is an ϵ-good hypothesis satisfying the PAC criterion.

It remains then only to show how f' can be constructed. First, observe that by Lemma 3.2 all significant variables occurring in f can be detected (and their signs and levels determined) in polynomial time. Moreover, by arguments similar to those given in § 3, it can be shown that if some triple of significant variables meet at a gate, then the level of that gate can be detected from the amplification function by hard wiring the three variables to 1. We call this information (the level and sign of each significant variable, and the level at which each triple of significant variables meet, if at all) the formula's *schedule*. It turns out that the schedule alone is sufficient to fully reconstruct the partially visible formula f', as is shown below.

These then are the main ideas of the algorithm. What follows is a more detailed exposition.

A *partially visible function* $f(x : y)$ is a Boolean function f on a set of *visible variables* $x = x_1 \cdots x_r$, and a set of *hidden variables* $y = y_1 \cdots y_s$. Two partially visible functions $f(x : y)$ and $g(x : z)$ on the same set of visible variables are *equivalent* with respect to distributions D and E on the domains of y and z if, for all x, $\Pr[f(x : Y) = 1] = \Pr[g(x : Z) = 1]$, where Y and Z are random variables representing a random assignment to y and z according to D and E. In the discussion that follows, we will only be interested in uniform distributions.

As described above, our algorithm regards variables that occur deep in the target formula as hidden variables. The next two lemmas show that two partially visible read-once majority formulas that are identical, except for some deep hidden variables, are very likely to produce the same output on randomly chosen inputs.

LEMMA 7.1. *Let f be a read-once majority formula on n variables. Let t be the level of x_n in f. Let $X_1, \ldots, X_{n-1}$, Y and Z be independent Bernoulli variables, each* 1 *with probability* 1/2. *Then* $\Pr[f(X_1, \ldots, X_{n-1}, Y) \neq f(X_1, \ldots, X_{n-1}, Z)] = 2^{-t-1}$.

Proof. The proof is by induction on t. If $t = 0$, then f is the function x_n and since $\Pr[Y \neq Z] = 1/2$, the lemma holds. If $t > 0$, then let $f = \text{MAJ}(f_1, f_2, f_3)$, and suppose that f_1 is the subformula in which x_n occurs. Since x_n does not also occur in f_2 or f_3, we will regard these as functions only on the remaining $n - 1$ variables. It is not hard to see then that $f(X_1, \ldots, X_{n-1}, Y) \neq f(X_1, \ldots, X_{n-1}, Z)$ if and only if $f_2(X_1, \ldots, X_{n-1}) \neq f_3(X_1, \ldots, X_{n-1})$ and $f_1(X_1, \ldots, X_{n-1}, Y) \neq f_1(X_1, \ldots, X_{n-1}, Z)$. Since f_2 and f_3 each output 1 independently with probability 1/2, we have

$$\Pr[f_2(X_1, \ldots, X_{n-1}) \neq f_3(X_1, \ldots, X_{n-1})] = 1/2.$$

Also, by inductive hypothesis,

$$\Pr[f_1(X_1, \ldots, X_{n-1}, Y) \neq f_1(X_1, \ldots, X_{n-1}, Z)] = 2^{-t}.$$

The lemma then follows by independence. □

LEMMA 7.2. *Let f be a read-once majority formula on n variables. Let t_i be the level of variable x_i in f. Let $X_1, \ldots, X_n, X'_1, \ldots, X'_r$, $r \leq n$, be independent Bernoulli variables, each* 1 *with probability* 1/2. *Then*

$$\Pr[f(X_1, \ldots, X_n) \neq f(X'_1, \ldots, X'_r, X_{r+1}, \ldots, X_n)] \leq \sum_{i=1}^{r} 2^{-t_i-1}.$$

Proof. The proof is by induction on r. If $r = 0$, then the lemma holds trivially. For $r > 0$, we have

$$\begin{aligned}
&\Pr[f(X_1, \ldots, X_n) \neq f(X'_1, \ldots, X'_r, X_{r+1}, \ldots, X_n)] \\
&\quad \leq \Pr[f(X_1, \ldots, X_n) \neq f(X'_1, \ldots, X'_{r-1}, X_r, \ldots, X_n)] \\
&\qquad + \Pr[f(X'_1, \ldots, X'_{r-1}, X_r, \ldots, X_n) \neq f(X'_1, \ldots, X'_r, X_{r+1}, \ldots, X_n)] \\
&\quad \leq \sum_{i=1}^{r-1} 2^{-t_i-1} + 2^{-t_r-1}
\end{aligned}$$

where the last inequality follows from our inductive hypothesis as well as from the preceding lemma. □

As proved below, Lemma 7.2 implies that any partially visible formula is an ϵ-good hypothesis if it is equivalent to the truncated target, the partially visible formula obtained from the target formula by regarding all variables at or below level $h = \lceil \log(n/2\epsilon) \rceil$ as hidden variables. Given an assignment to the visible variables, such a hypothesis is evaluated in the obvious manner by choosing a random assignment to the hidden variables and computing the output of the formula on the combined assignments to the hidden and visible variables. (Thus, the hypothesis is likely to be randomized.)

LEMMA 7.3. *Let $\epsilon > 0$, and let f be a read-once majority formula on n variables. Let $x = x_1 \cdots x_r$ be the variables occurring above level $h = \lceil \log(n/2\epsilon) \rceil$, and let $y = y_1 \cdots y_{n-r}$ be the remaining variables. Let $g(x : z)$ be any partially visible formula equivalent to the partially visible formula $f(x : y)$. Then $Pr[f(X : Y) \neq g(X : Z)] \leq \epsilon$, where X, Y, and Z are random variables representing the uniformly random choice of assignments to x, y, and z; that is, $g(x : z)$ is an ϵ-good hypothesis for f.*

Proof. Let Y' be a random variable representing a random assignment to y, chosen independently of Y. Since $f(x : y)$ is equivalent to $g(x : z)$, we have

$$\Pr[f(X : Y) \neq g(X : Z)] = \Pr[f(X : Y) \neq f(X : Y')].$$

By Lemma 7.2, the right-hand side of this equation is bounded by ϵ, since each of the $n - r \leq n$ variables y_i occurs at or below level h in f. □

For $\epsilon > 0$ and target formula f, we will henceforth say that variables occurring above level $h = \lceil \log(n/2\epsilon) \rceil$ are *significant*. Note that Theorem 3.3 implies that the significance, sign, and level of any variable x_j can be determined by hard wiring that variable to 1, as usual. More specifically, if $\hat{\alpha}$ is an estimate of $\alpha = A_{f|x_j \leftarrow 1}(\frac{1}{2})$ for which

$$|\hat{\alpha} - \alpha| < \tau \leq \left(\tfrac{1}{2}\right)^{h+2}$$

then x_j is significant if and only if

$$\left|\hat{\alpha} - \tfrac{1}{2}\right| > \left(\tfrac{1}{2}\right)^{h+1} + \tau,$$

and, if it is significant, then its sign and level can be determined as in Theorem 3.3.

Similar to Theorem 3.7, we can show that, for any triple of significant variables, we can determine the level of the gate at which the triple meets, if at all. More precisely, if x_i, x_j, and x_k are three unnegated variables occurring at levels t_1, t_2, and t_3, and if $\hat{\alpha}$ is an estimate of $\alpha = A_{f|x_i,x_j,x_k \leftarrow 1}(\frac{1}{2})$ for which $|\hat{\alpha} - \alpha| < \tau \leq 2^{-3h}$, then it follows from Lemmas 3.5 and 3.6 that x_i, x_j, and x_k meet at a level-d gate if and only if

$$\left|\tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1+1} + \left(\tfrac{1}{2}\right)^{t_2+1} + \left(\tfrac{1}{2}\right)^{t_3+1} - \left(\tfrac{1}{2}\right)^{t_1+t_2+t_3-2d-1} - \hat{\alpha}\right| < \tau.$$

As mentioned above, we call the sum total of this information—the significance of each variable, the level and sign of each significant variable, and the level of the gate at which each triple of significant variables meet, if at all—the formula's *schedule*. It remains then only to show how an ϵ-good hypothesis can be constructed from the schedule. Specifically, we show how to construct a partially visible formula that is equivalent to the truncated target $f(x : y)$. (Here, x is the vector of visible (i.e., significant) variables, and y is the vector of hidden (insignificant) variables.)

Suppose first that no three visible variables meet in f. Such a formula is said to be *unstructured*. Although, strictly speaking, f cannot be unstructured (by our choice of h), this special case turns out nevertheless to be important in handling the more general case since *subformulas* of f may be unstructured.

Lemma 7.4 below shows that an unstructured formula $f(x : y)$ is equivalent to any other unstructured partially visible formula whenever each visible variable occurs at the same level with the same sign in both formulas. Thus, unstructured formulas are not changed when visible variables are moved around within the same level. This fact makes the identification of unstructured formulas from their schedules quite easy. For any partially visible read-once majority formula $f(x : y)$, let $p_f(x) = \Pr[f(x : Y) = 1]$ where Y represents a random assignment to y.

LEMMA 7.4. *Let $f(x : y)$ and $g(x : z)$ be unstructured read-once majority formulas on s visible variables. Suppose that each visible variable x_j is relevant and occurs at the same level t_j with the same sign in both formulas. Then the two partially visible formulas are equivalent.*

Proof. It suffices to prove the lemma when no visible variable is negated since negated variables can simply be replaced by unnegated metavariables.

To prove the lemma, we show that

$$p_f(x) = \tfrac{1}{2} + \sum_{i=1}^{s} 2^{-t_i}(x_i - \tfrac{1}{2}). \tag{1}$$

Since this statement applies to any unstructured formula, it follows immediately that $p_f(x) = p_g(x)$ and the two partially visible formulas are equivalent.

We prove equation (1) by induction on the height h of f. If $h = 0$, then f consists of a single visible or hidden variable. If f is the formula x_j, where x_j is some visible variable, then $p_f(x) = x_j$, satisfying (1). If f is the formula y_j, where y_j is a hidden variable, then $p_f(x) = \frac{1}{2}$, also satisfying (1).

If $h > 0$, then let $f = \text{MAJ}(f_1, f_2, f_3)$ where f_1, f_2, and f_3 are partially visible subformulas. Since f is unstructured, one of these (say, f_3) contains no visible variables, and thus $p_{f_3}(x) = \frac{1}{2}$. Suppose, without loss of generality, that $x_1, \ldots, x_r$ are the visible variables relevant to f_1. Then, by inductive hypothesis,

$$p_{f_1}(x) = \tfrac{1}{2} + \sum_{i=1}^{r} 2^{-t_i+1}(x_i - \tfrac{1}{2})$$

and

$$p_{f_2}(x) = \tfrac{1}{2} + \sum_{i=r+1}^{s} 2^{-t_i+1}(x_i - \tfrac{1}{2}).$$

Applying Lemma 3.1, it is easily verified that (1) is satisfied, completing the induction. □

Thus, if $f(x : y)$ is unstructured, then an equivalent unstructured formula can be constructed from f's schedule. For instance, here is an efficient algorithm: Let t be the depth of the deepest visible variable in f. Break the set of all level-t variables into pairs. Replace each such pair x_i, x_j by a level-$(t-1)$ visible metavariable $w \equiv \text{MAJ}(x_i, x_j, y)$, where y is a new hidden variable. If an odd level-t variable x_i remains, replace it with a level-$(t-1)$ visible metavariable $w \equiv \text{MAJ}(x_i, y, y')$, where y and y' are new hidden variables. Repeat for levels $t-1, t-2, \ldots, 1$. It is not hard to show that this algorithm results in a formula that is unstructured, and that is consistent with f's schedule (and so is equivalent).

With these tools in hand for dealing with unstructured formulas, we are now ready to describe an algorithm for handling the general case, i.e., for reconstructing any (not necessarily unstructured) formula from its schedule.

Let $f(x : y)$ be the truncated target. If f is unstructured, then the previous algorithm applies. Otherwise, we can find from the schedule three visible variables x_i, x_j, and x_k that meet at some maximum-depth gate λ of f; that is, they meet at a level-d gate, and no triple of visible variables meet at any gate of depth exceeding d. Then the subformula g subsumed by λ computes the majority of three subformulas g_1, g_2, and g_3, each containing one of x_i, x_j, and x_k (say, in that order). Let x_ℓ be some other visible variable. Then it is easily verified that x_ℓ is relevant to g_1 if and only if x_ℓ, x_j, and x_k meet at a level-d gate (namely, λ). Thus, all of the visible variables relevant to g_1 (and likewise for g_2 and g_3) can be determined from the schedule. Moreover, note that each of these subformulas is unstructured since λ is of maximum depth. Thus, each subformula can be identified using the previous algorithm for unstructured formulas, and therefore, the entire subformula subsumed by (and including) λ can be identified.

The rest of the formula can be identified recursively: We replace subformula g by a new metavariable w, and update the schedule appropriately.

This completes the description of the algorithm. We thus have the following theorem.

THEOREM 7.5. *There exists an algorithm with the following properties: Given n, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0,1\}^n$, and labeled by any read-once majority formula f on n variables, the algorithm outputs an ϵ-good hypothesis for f with probability at least $1-\delta$. The algorithm's sample complexity is $O((n/\epsilon)^6 \cdot \log(n/\delta))$, and its time complexity is $O((n^9/\epsilon^6) \cdot \log(n/\delta))$.*

Proof. As before, let $h = \lceil \log(n/2\epsilon) \rceil$. To implement the procedure outlined above, we must be able to compute from a random sample: (1) each variable's significance, (2) each significant variable's sign and level, and (3) the level of the gate (if any) at which each triple of significant variables meet. As noted above, to compute (1) and (2), we need to find, for each variable x_i, an estimate $\hat{\alpha}$ of $\alpha = A_{f|x_i \leftarrow 1}(\frac{1}{2})$ for which $|\alpha - \hat{\alpha}| < (1/2)^{h+2}$. Hoeffding's inequality (Lemma 3.9) implies that such estimates can be derived, with high probability, from a sample of size $O(2^{2h} \log(n/\delta))$. We also noted above that we can compute (3) given, for each triple of significant variables x_i, x_j, x_k, an estimate $\hat{\alpha}$ of $\alpha = A_{f|x_i,x_j,x_k \leftarrow 1}(\frac{1}{2})$ for which $|\hat{\alpha} - \alpha| < 2^{-3h}$. Again applying Hoeffding's inequality, we see that such estimates can be computed (with high probability) for all triples of variables from a sample of size $O(2^{6h} \log(n/\delta))$. This proves the stated sample size.

The running time of the procedure is dominated by the computation from the sample of the $O(n^3)$ estimates described above. □

8. Summary. In this paper, we described a general technique for inferring the structure of read-once formulas over various bases. We have shown how this technique can be applied to achieve exact identification when the formula is of logarithmic depth, even in the presence of noise. We have also described how to use this technique to infer good approximations of formulas with unbounded depth.

Acknowledgments. We are very grateful to Ron Rivest for his comments on this material, and for a careful reading he made of an earlier draft. We also thank Avrim Blum for suggesting the notion of a universal identification sequence. Finally, we thank the anonymous referees for their comments.

REFERENCES

[1] D. Angluin, L. Hellerstein, and M. Karpinski, *Learning read-once formulas with queries*, J. Assoc. Comput. Mach., 40(1993), pp. 185–210.

[2] D. Angluin and P. Laird, *Learning from noisy examples*, Machine Learning, 2 (1988), pp. 343–370.

[3] G. M. Benedek and A. Itai, *Learnability with respect to fixed distributions*, Theoret. Comput. Sci., 86 (1991), pp. 377–389.

[4] ———, *Amplification of probabilistic Boolean formulas*, in Advances in Computing Research 5: Randomness and Computation, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 27–45.

[5] ———, *Lower bounds for monotone circuits and formulas*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1986.

[6] N. Bshouty, T. Hancock, L. Hellerstein, and M. Karpinski, *Read-once formulas, justifying assignments, and generic transformations*, unpublished manuscript, 1991.

[7] M. Furst, J. Jackson, and S. Smith, *Improved learning of AC^0 functions*, in Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Aug. 1991, pp. 317–325.

[8] S. A. Goldman and M. J. Kearns, *On the complexity of teaching*, in Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Aug. 1991, pp. 303–314.

[9] T. Hancock and L. Hellerstein, *Learning read-once formulas over fields and extended bases*, in Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Aug. 1991, pp. 326–336.

[10] T. Hancock and Y. Mansour, *Learning monotone $k\mu$ DNF formulas on product distributions*, in Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Aug. 1991, pp. 179–183.

[11] L. Hellerstein, *On characterizing and learning some classes of read-once formulas*, Ph.d. thesis, University of California, Berkeley, CA, 1989.

[12] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[13] J. Kearns, *The Computational Complexity of Machine Learning*, MIT Press, Cambridge, MA, 1990.

[14] M. Kearns and M. Li, *Learning in the presence of malicious errors*, in Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, May 1988, pp. 267–280.

[15] M. Kearns, M. Li, L. Pitt, and L. Valiant, *On the learnability of Boolean formulae*, in Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, May 1987, pp. 285–295.

[16] M. Kearns and L. G. Valiant, *Cryptographic limitations on learning Boolean formulae and finite automata*, in Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, May 1989, pp. 433–444; J. Assoc. Comput. Mach., to appear.

[17] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Oct. 1989, pp. 574-579.

[18] G. Pagallo and D. Haussler, *A greedy method for learning μ DNF functions under the uniform distribution*, Tech. Report UCSC-CRL-89-12, Computer Research Laboratory, University of California, Santa Cruz, CA, June 1989.

[19] L. Pitt and M. K. Warmuth, *Prediction-preserving reducibility*, J. Comput. System Sci., 41 (1990), pp. 430–467.

[20] R. E. Schapire, *The strength of weak learnability*, Machine Learning, 5 (1990), pp. 197–227.

[21] ———, *Learning probabilistic read-once formulas on product distributions*, in Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Aug. 1991, pp. 184–198, Machine Learning, to appear.

[22] A. Shinohara and S. Miyano, *Teachability in computational learning*, New Generation Computing, 8 (1991), pp. 237–247.

[23] R. H. SLOAN, *Types of noise in data for concept learning*, in Proceedings of the 1988 Workshop on Computational Learning Theory, Aug. 1988, pp. 91–96.

[24] L. G. VALIANT, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[25] K. VERBEURGT, *Learning* DNF *under the uniform distribution in quasi-polynomial time*, in Proceedings of the Third Annual Workshop on Computational Learning Theory, Aug. 1990, pp. 314–326.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 727–750, August 1993

ASYNCHRONOUS FAULT-TOLERANT TOTAL ORDERING ALGORITHMS *

LOUISE E. MOSER[†], P. M. MELLIAR-SMITH[†], AND VIVEK AGRAWALA[‡]

Abstract. Two novel efficient algorithms for placing a total order on messages in an asynchronous fault-tolerant distributed system are presented. The algorithms are resilient to fewer than $n/3$ and $n/2$ faulty processes in an n-process system. Partial correctness and probabilistic termination are demonstrated; it is also shown that there does not exist a total ordering algorithm that is guaranteed to terminate. A comparison of the complexity of the algorithms is given.

Key words. distributed computing, fault tolerance, asynchronism, probabilistic algorithm, partial order, total order

AMS subject classifications. 68M10, 68N25, 68Q10, 68Q20

1. Introduction.

As computers have become less expensive, many computer systems are now being designed as distributed networks of computers cooperating to provide an overall service. One advantage of a network of computers is the fault tolerance made possible by the use of multiple processes executing on different computers. Fault tolerance requires not only extra processes but also replication of data so that the application can continue unaffected by the loss of a process. Operation of a fault-tolerant distributed system requires careful coordination to ensure that inconsistencies do not develop between the data located at different processes. The problem of ensuring that the various processes of a distributed system make consistent decisions, despite process and communication faults, is an important area of research, of great intellectual interest, and of practical importance.

The problem of achieving distributed consensus in synchronous and partially synchronous systems is relatively easy, and effective fault-tolerant algorithms for reaching consensus have been developed [5], [6], [10]; synchronous systems are, however, limited in flexibility. In asynchronous systems, the consensus problem is much more difficult; it is, in fact, impossible to construct an asynchronous fault-tolerant consensus algorithm that is guaranteed to terminate [8]. Consequently, probabilistic algorithms have been devised for which the probability of not reaching consensus diminishes asymptotically to zero as more messages are processed [1], [3], [7], [12], [14]; such algorithms typically involve the exchange of many messages for each consensus decision and, thus, are expensive to operate.

An alternative approach to achieving consensus in a distributed system involves placing a total order on messages broadcast or multicast to process groups; such an order allows distributed consensus decisions to be reached using simple sequential algorithms. Several systems have been developed that use a total ordering of messages [2], [4], [13]. These systems share the characteristic that their ordering algorithms are not inherently fault tolerant. Rather, in the event of a process fault, the algorithms stop until an underlying fault-detection and system-reconfiguration algorithm has removed the faulty processes and formed a new configuration. Services requiring consensus decisions cannot be provided until this reconfiguration activity is completed.

We describe here the first truly fault-tolerant total ordering algorithms that continue to form the total order even in the presence of faulty processes so that system operation continues uninterrupted; this property is very desirable, if not essential, for many fault-tolerant distributed

*Received by the editors October 24, 1990; accepted for publication (in revised form) May 10, 1992. This research was supported by National Science Foundation grants CCR-8908515 and NCR-9016361.

[†]Department of Electrical and Computer Engineering, University of California, Santa Barbara, California 93106.

[‡]Siemens Corporate Research, 755 College Road East, Princeton, New Jersey 08540.

systems. The total ordering problem in an asynchronous fault-tolerant distributed system is as hard as the consensus problem; in fact, we show that it is impossible to construct a fault-tolerant total ordering algorithm that is guaranteed to terminate. The difficulty is that messages may be lost and may have to be retransmitted; thus, different processes may receive the same messages in different orders. Communication faults can cause arbitrary delays in the delivery of a message, but a process cannot wait forever for a message from another process because that process may have failed and the message may never be delivered.

The Total algorithms presented here are fault-tolerant algorithms for incrementally converting a partial order on messages into a total order in such a way that every nonfaulty process constructs the same total order and all processes construct consistent total orders. Because of the impossibility of constructing a terminating fault-tolerant total ordering algorithm, the Total algorithms are necessarily probabilistic algorithms. There are basically two ways to incorporate probabilities into the possible executions of an algorithm. In the first approach [1], explicit random steps are introduced into the algorithm. The second approach [3], which is the one we adopt, assumes probabilistic behavior of the communication mechanism.

The partial order on which the Total algorithms are based can be constructed from almost any underlying communication mechanism. Typically, each message contains an acknowledgment or context field that defines directly, or indirectly through the transitive closure, the messages that it follows in the partial order. When the total algorithms are implemented using broadcast communication, the partial order is "narrow" and high efficiency can be achieved. The Total algorithms convert the partial order into a total order without the need to broadcast additional messages. On average, each message broadcast extends the total order by one message plus any retransmissions that may be required due to communication faults. We have implemented the Total algorithms on a network of Sun workstations using the Trans protocol [11] to provide a partial order on broadcast messages.

The Total algorithms are genuinely distributed algorithms that are executed independently and symmetrically by each process. The algorithms are completely transparent to process and communication faults and continue to operate without the need to identify such faults. Because they operate in an asynchronous fault-tolerant environment, termination cannot be guaranteed; probabilistic termination is, however, guaranteed. Algorithms with these properties are not easy to construct and may contain subtle errors; thus, detailed proofs of correctness are required. The proofs also help in understanding the characteristics of the algorithms.

This paper is organized as follows. In §2 we define our model of computation, and in §3 we introduce the basic idea of the Total algorithms. In §4 we present a Total algorithm that is resilient to fewer than $n/3$ faulty processes along with an example and the proof of partial correctness and probabilistic termination. In §5 we present a more complex Total algorithm that is resilient to fewer than $n/2$ faulty processes. In §6 we compare the complexity of the two algorithms, and in §7 we establish the impossibility result. The Appendix contains the proof for the $n/2$ resilient Total algorithm.

2. The model. We consider an asynchronous fault-tolerant distributed system with n processes, where $n \geq 2$. The system is asynchronous in that no bound can be placed on the time required for a computation or for communication of a message. The system is subject to process and communication faults that are not malicious. Messages may be lost or arbitrarily delayed. We assume that the maximum number of faulty processes (the resilience) is k, where $k < n/2$. If the number of faulty processes is k or less, the fault-tolerant total ordering algorithms operate to extend the total order; if the number of faulty processes exceeds k, the algorithms stop.

A distributed total ordering algorithm executes on each process independently; thus, the execution of such an algorithm is modeled as a single process. The input to the algorithm is an infinite partial order on messages, common to all processes. The output of the algorithm is an infinite total order. The algorithm cannot, of course, examine the entire partial order but accepts the partial order, and extends the total order, incrementally. Only information derived from the partial order is used to construct the total order; no other communication between processes is allowed.

We consider the set M of messages of a particular execution of the system and assume that M is partially ordered by a partial order relation, called *follows*. Being a partial order, the follows relation is reflexive, antisymmetric, and transitive. A total order relation satisfies the additional property that every two elements are comparable. We let m_{pi} denote the ith message in the sequence of messages broadcast by process p and assume that m_{pi} follows only a finite number of messages in the partial order and that $m_{p,i+1}$ follows m_{pi}.

The partial order M can be derived from the acknowledgments of messages typically used in broadcast communication. A message m' from process q follows a message m from process p if m' acknowledges m or a message m'' that follows m, where $m'' \neq m$ and $m'' \neq m'$. Some care is required to ensure that messages are received as well as ordered [11]. Thus, m' follows m only if process q, when broadcasting message m', received every message that process p had received when broadcasting message m. The follows relation is a subset of Lamport's causality relation [9]. Because messages may be lost or delayed, some of the edges of the causality relation may be excluded from the follows relation.

A *prefix A of a partial order M* is a subset of the messages of M together with the partial order relations between them such that, if m' is an element of A and m' follows m, then m is also an element of A. The *size* of a partial order prefix A is the number of messages in A.

A total order relation on a set of messages is, of course, reflexive, antisymmetric, and transitive and, most importantly, satisfies the property that every two elements are comparable. A *total order prefix* is a finite set of messages that is totally ordered and that can, therefore, be represented as a finite sequence $m_{p_1 i_1}, \ldots, m_{p_t i_t}$.

An *internal state* of a process consists of a prefix of the partial order and a prefix of the total order. The *initial state* of a process consists of an empty prefix of the partial order and an empty prefix of the total order.

In a *step*, a process's partial order prefix A is extended by one message m, together with the partial order relations between m and the messages in A, and its total order prefix is extended by zero or more messages. A step is *applicable* to an internal state if the only messages that m follows are in the partial order prefix of that state. Hereafter, we only consider steps that are applicable.

A process p is *nonfaulty* in a partial order M if at least one message from p is included in M and if each message from p is followed by a message from every nonfaulty process; otherwise, process p is *faulty*. This implies that each nonfaulty process has infinitely many messages in M.

The asynchronous nature and faulty behavior of the processes and the communication mechanism are reflected in the partial order that is the input to the algorithm and in the order in which messages are supplied to a process to extend its prefix. A process has no control over which message extends its partial order prefix in a step, and it cannot determine whether any future extension of its partial order prefix will involve a message that follows a particular message. A message from a faulty process may be followed by no other message and, thus, may be supplied to one process at an arbitrary number of steps after it has been supplied to another process, if at all.

A partial order is *admissible* if it satisfies the following liveness and fairness properties. Partial orders with these properties are readily obtained from the acknowledgments in broadcast messages [11]. It is easy to construct partial orders which do not satisfy these properties and which are, therefore, not admissible and to which our algorithms do not apply.

Liveness

1. For each message m from a nonfaulty process p, process p eventually obtains a prefix A such that $m \in A$.

Let p and q be distinct nonfaulty processes and let m be a message from p. We define $\mathrm{E}(m, p, q)$ to be the event that there exists a message m' from q that follows the message m from p and that there does not exist a message m'' such that m' follows m'' and m'' follows m, where $m'' \neq m$ and $m'' \neq m'$.

Fairness

1. $0 < \epsilon < \mathrm{pr}(\mathrm{E}(m, p, q)) < 1$.
2. $\mathrm{E}(m, p, q)$ and $\mathrm{E}(m', r, s)$ are independent events, where $m \neq m'$ or $q \neq s$.

The above fairness properties formalize an assumption of no bias in the selection of processes to broadcast or in the choice of processes to receive messages. The intuition underlying the first fairness property is that, for each process q, the event that message m from process p is directly followed by a message from process q is neither guaranteed nor precluded from happening. Similarly, the second fairness property requires that two such events neither force nor preclude each other.

The validity of the total ordering algorithms depends on the following properties.

Partial Correctness

1. The total orders determined by any two processes are consistent, i.e., if any process determines that m is the ith message of the total order, then no process determines that m' is the ith message, where $m' \neq m$.
2. The total order is consistent with the partial order, i.e., if m' follows m in the total order, then m does not follow m' in the partial order, where $m' \neq m$.

Probabilistic Termination

1. The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.
2. For each message m broadcast by a nonfaulty process q, the probability that a nonfaulty process p places m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.

Termination is unfortunately precluded by the impossibility result.

3. The total ordering algorithms. The Total algorithms operate within the context of a system in which processes communicate by broadcasting messages. The objective of the algorithms is to place a total order on messages in such a way that nonfaulty processes construct identical total order prefixes and all processes construct consistent total order prefixes. The total order prefixes are constructed incrementally and independently by each of the processes. A total order on messages is useful for applications that require distributed consensus decisions.

The input to the Total algorithms is a partial order prefix, i.e., a finite set of messages that is partially ordered. As each process receives messages, it builds its partial order prefix (a process is assumed to receive each message it broadcasts). Sometimes receipt of a message may result in no addition to the partial order prefix, while receipt of another message may lead to the inclusion of several messages in the partial order prefix. The partial order construction is not, however, the subject of this paper.

In our description of the algorithms, an execution step consists of adding exactly one message to the partial order prefix and executing one of the Total algorithms described in

§§4 and 5. The algorithms are executed independently and concurrently by each process. Execution of these algorithms results in a process's total order prefix being extended by zero or more messages.

The messages in the partial order prefix that have already been advanced to the total order play no part in the further extension of the total order prefix. Some of the messages in the partial order prefix follow other messages that have not yet been advanced to the total order; such a message cannot be the next message to extend the total order prefix. Thus, for each partial order prefix, we define a *candidate message* to be a message that is not yet in the total order prefix and that follows only messages that are already in the total order prefix. A set of candidate messages is referred to as a *candidate set.*

In each execution step, the algorithms consider all candidate sets that can be constructed from the candidate messages in the partial order prefix. A step may result in a decision to extend the total order prefix. Alternatively, a step may result in no candidate set's obtaining a favorable decision and, thus, no extension to the total order prefix. In the next step, with an additional message in the partial order prefix, another attempt will be made to extend the total order prefix.

The lth decision of a process p in favor of a candidate set for inclusion in the total order determines a set S_p^l of candidate messages by which p extends its total order prefix T_p^{l-1} to obtain T_p^l. The initial candidate set S_p^0 and the initial total order prefix T_p^0 are empty. The messages of the candidate set S_p^l are included in the total order, following the messages in T_p^{l-1}, in any arbitrary but deterministic order, such as lexicographic order of message identifiers.

A decision by a process to advance a candidate set to the total order is determined by the votes of the messages in its partial order prefix. These votes are not contained explicitly in the messages but are deduced from the partial order relationships between messages in the partial order prefix, as we will describe. Each process makes its own decisions independently without reference to the decisions being made by other processes.

In an execution step, each candidate set is voted on separately and independently; messages in the partial order prefix provide votes on the candidate sets. Voting on a candidate set takes place in a sequence of stages; each candidate set has its own sequence of stages. In a step, voting proceeds sequentially through the stages for a particular candidate set but concurrently on all of the candidate sets, since it cannot be predetermined which message is able to vote for which candidate set at which stage. A decision in favor of a candidate set cannot be made until decisions against all of its proper subsets have been made.

At stage 0, the vote of a message on a candidate set depends on which candidate messages that message follows. At stage $i > 0$, the vote of a message on a candidate set depends on whether that message follows "enough" (defined below) messages that vote at stage $i-1$. A message may not vote at a stage at which a previous message from its source votes. A message may be unable to vote at any stage on a candidate set if the voting criteria (given below) are not satisfied. Conversely, a message may be able to vote at several stages if the voting criteria are satisfied for each of those stages and no previous message from its source has voted at those stages. Since each message follows itself in the partial order, a message can include itself in the number of messages required to vote.

In §§4 and 5 we present the two alternative Total algorithms that are resilient to $n/3$ and $n/2$ faulty processes. The second algorithm involves an intermediate form of voting and is more complex than the first but provides higher resilience. In the presentation, each sequence of stages refers to a particular candidate set S, but we write, for example, "votes for S at stage i" as an abbreviation for "votes for S at stage i of the sequence of stages for S." We use "follows" to mean "follows in the partial order," and "for the lth extension" to mean "for the lth extension of the total order." If a lemma is stated in terms of "for (against) a candidate set," we prove the statement for one alternative only, provided that the proof for the other alternative is similar.

4. The $n/3$ resilient algorithm. For this algorithm, we assume that the resilience $k < n/3$. The algorithm is defined by the following voting and deciding criteria; these criteria determine which candidate set is chosen for inclusion in the total order at the lth extension.

- Each process, before determining the vote of a message m on a candidate set S for the lth extension of the total order, must obtain a prefix A of the partial order such that $m \in A$.
- A message can vote on a candidate set S for the lth extension of the total order at stage i only if no previous message from its source has voted on S for the lth extension at stage i.
- The number of votes required for a decision and the number of votes required for a further vote are at least N_d and N_v, where

$$N_d = (n + k + 1)/2 \quad \text{and} \quad N_v = (n - k)/2.$$

The Criteria for Voting on a Candidate Set S for the lth Extension

At stage 0,

- A message votes for S if that message follows every message in S and it follows no other candidate message. (A candidate message votes for the set containing only itself.)
- A message votes against S if that message follows any candidate message other than those in S. (A candidate message votes against all sets of which it is not a member.)

At stage i, where $i > 0$,

- A message votes for S if
  - it follows at least two messages that vote on S at stage $i - 1$,
  - it follows at least N_v messages that vote for S at stage $i - 1$, and
  - it follows fewer messages that vote against S than vote for S at stage $i - 1$.
- A message votes against S if
  - it follows at least two messages that vote on S at stage $i - 1$,
  - it follows at least N_v messages that vote against S at stage $i - 1$, and
  - it does not vote for S at stage i.

The Criteria for Deciding on a Candidate Set S for the lth Extension

At stage i, where $i \geq 0$,

- A process decides for S if
  - it determines that at least N_d messages vote for S at stage i, and
  - for each proper subset of S, it decides against that proper subset.
- A process decides against S if
  - it determines that at least N_d messages vote against S at stage i, or
  - it decides for a proper subset of S.

The numbers of votes, which determine the values of N_d and N_v, are based on the following properties: Decisions do not conflict ($N_d + N_v > n$ and $2 * N_d > n$), decisions are feasible ($N_d \leq n - k$), and stages of voting advance ($2 * N_v - 1 \leq n - k$). Analysis of these inequalities readily yields the requirements given above and also yields the constraint $k < n/3$.

4.1. An example. Consider the messages of the partial order prefix that have not been advanced to the total order shown in Fig. 1. Here there are $n = 6$ processes and the resilience $k = 1$. Thus, $N_d = 4$ and $N_v = 2.5$, requiring at least four votes for a decision and at least three votes for a further vote.

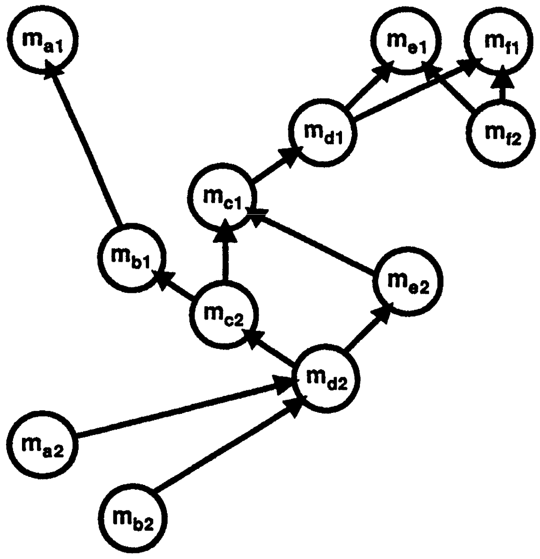


FIG. 1. *A partial order prefix in an $n = 6$ process system with resilience $k = 1$, $N_d = 4$, and $N_v = 2.5$, requiring at least four votes for a decision and at least three votes for a further vote. Here the candidate sets $\{m_{a1}\}$, $\{m_{e1}\}$, and $\{m_{f1}\}$ obtain too many unfavorable votes at stage 0 and, thus, are decided against, but the set $\{m_{e1}, m_{f1}\}$ obtains four favorable votes at stage 0 from m_{d1}, m_{c1}, m_{e2}, and m_{f2}, enough for a favorable decision. Even if message m_{f2} is lost, there remain three favorable votes at stage 0, but there are four favorable votes at stage 1 from m_{e2}, m_{d2}, m_{a2}, and m_{b2}; again, enough for a favorable decision.*

We assume initially that the process executing the algorithm has the entire partial order prefix. The partial order prefix contains three candidate messages m_{a1}, m_{e1}, and m_{f1}. The candidate sets $\{m_{e1}\}$ and $\{m_{f1}\}$ are voted for only by the messages themselves. Messages m_{a1} and m_{b1} vote for the set $\{m_{a1}\}$ but messages m_{c2} and m_{d2} do not because they follow other candidate messages in the partial order. Messages m_{d1}, m_{c1}, m_{e2}, and m_{f2} vote for the set $\{m_{e1}, m_{f1}\}$, a sufficient number of votes for a decision. Note that the candidates in the set $\{m_{a1}, m_{e1}, m_{f1}\}$ precede the four messages m_{c2}, m_{d2}, m_{a2}, and m_{b2}. A process cannot, however, decide immediately in favor of the set $\{m_{a1}, m_{e1}, m_{f1}\}$ since it must first decide on the set $\{m_{e1}, m_{f1}\}$.

Next we consider the possibility that a process may fail. In particular, suppose that process f fails some time after broadcasting message m_{f1} and before broadcasting message m_{f2}. The partial order prefix seen by the other processes does not include message f_2 and, thus, they do not know whether f had received messages m_{e1}, m_{d1}, m_{c1}, and m_{e2} and, thus, had decided in favor of the set $\{m_{e1}, m_{f1}\}$. Nor can they be confident that f had indeed failed; f may be trying to broadcast but may be blocked by contention for the bus, or may be working on an urgent task, or may be taking a short siesta from which it will awake to announce that it has indeed received those messages and decided for $\{m_{e1}, m_{f1}\}$ or, alternatively, that it has received message m_{a1}, and, thus, has decided against $\{m_{e1}, m_{f1}\}$, as the case may be.

However, even without knowledge of process f's vote, three messages m_{d1}, m_{c1}, and m_{e2} vote in favor of the set $\{m_{e1}, m_{f1}\}$ at stage 0, and four messages m_{e2}, m_{d2}, m_{a2}, and m_{b2} follow those three messages and, therefore, vote in favor of $\{m_{e1}, m_{f1}\}$ at stage 1. Consequently, messages m_{e2}, m_{d2}, m_{a2}, and m_{b2} suffice for the decision to include the set $\{m_{e1}, m_{f1}\}$ in the total order.

4.2. The proof for the $n/3$ resilient algorithm.

Partial Correctness

The proof of partial correctness involves showing that if processes p and q make lth extensions to their total order prefixes, then the candidate sets they choose for those extensions are identical and, thus, their lth total order prefixes are identical, i.e., $S_p^l = S_q^l$ and $T_p^l = T_q^l$. The proof is by induction on l. The base case is trivial since, if $l = 0$, these sets are all empty. The bulk of the proof constitutes establishing the inductive step.

Lemma 4.1 is important because it stipulates that the only information used by the algorithm to determine the vote of a message is the partial order on messages. By "determine the vote of a message" we mean determine if the message voted for or against a candidate set, or was unable to vote because it did not follow enough messages that voted on the candidate set.

LEMMA 4.1. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p and q each determine the vote of a message m on a candidate set S for the lth extension of the total order at stage i, then they determine the same vote of m on S at stage i.*

Proof. The proof is by induction on i. If process p determines the vote of a message m on a candidate set S at stage 0, then p has obtained a partial order prefix A that contains message m. Since A is a prefix, all of the messages that m follows are contained in A. Process q likewise has obtained a partial order prefix A' that contains message m and all of the messages that m follows. Since the vote of a message m on a candidate set S at stage 0 is determined by the messages that m follows, p and q determine the same vote of m.

We now assume that the statement holds for $i - 1$, where $i > 0$. If process p determines the vote of a message m on a candidate set S at stage i, then p has obtained a partial order prefix A that contains m and all of the messages that m follows. Process q likewise obtains a partial order prefix A' that contains m and all of the messages that m follows. The vote of a message at stage i, where $i > 0$, is determined by the votes of the messages that m follows at stage $i - 1$. By the induction hypothesis, p and q determine the same votes of the messages that m follows at stage $i - 1$. Thus, they determine the same vote of m. □

LEMMA 4.2. *Each process p broadcasts at most one message that votes on a candidate set S at stage i; that message does not vote both for and against the candidate set S at stage i.*

Proof. The first statement follows from the requirement that a message votes on a candidate set S at stage i only if no previous message from its source has voted on S at stage i. The second statement follows from the voting criteria. □

Lemmas 4.1 and 4.2 will be used in subsequent proofs without reference.

LEMMA 4.3. *If a message follows at least N_d messages that vote for (against) a candidate set S at stage $i - 1$, then no message votes against (for) S at stage i, where $i > 0$.*

Proof. If a message follows $N \geq N_d = (n + k + 1)/2$ messages that vote for S at stage $i - 1$, then at most $n - N \leq n - (n + k + 1)/2 < (n - k)/2 = N_v$ messages vote against S at stage $i - 1$. If a message m' votes against S at stage i, then m' follows at least N_v messages that vote against S at stage $i - 1$, which is a contradiction. □

LEMMA 4.4. *If message m votes against (for) a candidate set S at stage j then, for each i such that $0 \leq i \leq j$, m follows a message that votes against (for) S at stage i.*

Proof. The proof is by a simple induction on j using the voting criteria. □

LEMMA 4.5. *If no message votes against (for) a candidate set S at stage i, then no message votes against (for) S at stage j, where $j \geq i$. Likewise, if process p decides for (against) a candidate set S at stage i, then no message votes against (for) S at stage j, where $j > i$.*

Proof. To prove the first statement, we note that if a message m votes against a candidate set S at stage j then, by Lemma 4.4, m follows a message that votes against S at stage i, which is a contradiction. The second statement follows from the decision criteria, Lemma 4.3, and the first statement. □

Propositions 4.1–4.5 constitute a major part of the proof and are used in proving Theorem 4.1, which establishes consistency of decisions among the processes.

PROPOSITION 4.1. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p decides for a candidate set S for the lth extension of the total order, then q does not decide for a proper subset S' of S for the lth extension.*

Proof. The proof is by induction on the cardinality c of S. If $c = 1$, the statement holds since, by the voting and decision criteria and a simple induction, no process decides on the empty candidate set.

Assume that the statement holds for candidate sets S of cardinality less than c. We argue by contradiction, assuming further that p decides for S and that q decides for a proper subset S' of S at some stage j. Thus, q determines that $N' \geq N_d$ messages vote for S' at stage j.

By the inductive assumption, since q decides for a proper subset S' of S, p does not decide for a proper subset S'' of S'. Thus, since p decides for S, p decides against S' at some stage i for the reason that p determines that at least $N \geq N_d$ messages vote against S' at stage i. By Lemma 4.1, p and q determine the same vote of any message at each stage i of the voting on S.

If $i = j$ then, by Lemma 4.2, since no message votes both for and against S', $N + N' \geq N_d + N_d = n + k + 1 > n$, which is a contradiction. If $j > i$ then, by Lemma 4.5, since p decides against S' at stage i, no message votes for S' at stage j, which is a contradiction. Similarly, if $i > j$, we obtain a contradiction by applying Lemma 4.5 to process q with i and j interchanged. □

PROPOSITION 4.2. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p decides for (against) a candidate set S for the lth extension of the total order, then q does not decide against (for) S for the lth extension.*

Proof. Assume that p decides for S at stage i and that q decides against S at stage j. Since p decides for S at stage i, p determines that at least $N \geq N_d$ messages vote for S at stage i. Furthermore, by Proposition 4.1, q does not decide for a proper subset S' of S. Thus, q decides against S at stage j for the reason that q determines that at least $N' \geq N_d$ messages vote against S at stage j. By Lemma 4.1, p and q determine the same vote of any message at each stage i of the voting on S.

If $i = j$ then, by Lemma 4.2, since no message votes both for and against S at stage j, $N + N' \geq N_d + N_d = n + k + 1 > n$, which is a contradiction. Otherwise, we reach a contradiction by Lemma 4.5. □

LEMMA 4.6. *Let S and S' denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$, and $s' \in S'$, $s' \notin S$. If message m from process p votes for S at stage i, then m or a previous message from p votes against S' at stage i.*

Proof. The proof is by induction on i. If message m from process p votes for S at stage 0, then it follows the candidate message $s \in S$. Since m follows s and $s \notin S'$, by the voting criteria, m or a previous message from p votes against S' at stage 0, which establishes the base case.

We now assume the statement for $i - 1$. Let m follow N messages that vote for S at stage $i - 1$, $\bar{N}$ messages that vote against S at stage $i - 1$, N' messages that vote for S' at stage $i - 1$, and $\bar{N}'$ messages that vote against S' at stage $i - 1$. Since m votes for S at stage i, $N \geq N_v$,

$N > \bar{N}$, and $N + \bar{N} \geq 2$. Thus, $N \geq 2$. By the inductive assumption, if a message votes for S at stage $i - 1$, that message or a previous message from its source votes against S' at stage $i - 1$. Thus, $\bar{N'} \geq N$. Similarly, if a message votes for S' at stage $i - 1$, that message or a previous message from its source votes against S at stage $i - 1$. Thus, $\bar{N} \geq N'$. Consequently, $\bar{N'} \geq N_v$, $\bar{N'} > N'$, and $N' + \bar{N'} \geq 2$. Thus, m or a previous message from p votes against S' at stage i unless a previous message from p votes for S' at stage i.

Now suppose that there exists such a message m' from p that votes for S' at stage i. As above, we conclude that m' or a previous message from p votes against S at stage i unless a previous message from p votes for S at stage i. Since m votes for S at stage i, we have a contradiction by Lemma 4.2. □

PROPOSITION 4.3. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$, and let S and S' denote candidate sets for the lth extension of the total order such that there exist messages $s \in S$, $s \notin S'$, and $s' \in S'$, $s' \notin S$. If p decides for S for the lth extension of the total order, then q does not decide for S' for the lth extension.*

Proof. If process p decides for S at stage i, then p determines that at least N_d messages vote for S at stage i and thus p has obtained a prefix of the partial order containing those messages and all of the messages that they follow. Let m denote any message that votes for S at stage i. By Lemma 4.6, m or a previous message from its source votes against S' at stage i. Thus, p determines that at least N_d messages vote against S' at stage i and, therefore, p decides against S' at stage i. By Proposition 4.2, q does not decide for S'. □

PROPOSITION 4.4. *If process p makes the lth extension of its total order prefix T_p^{l-1} by including the candidate set S_p^l, and if process q makes the lth extension of its total order prefix T_q^{l-1} by including the candidate set S_q^l, and if $T_p^{l-1} = T_q^{l-1}$, then $S_p^l = S_q^l$ and, consequently, $T_p^l = T_q^l$.*

Proof. By Proposition 4.2, if process p decides for S_p^l, then process q does not decide against S_p^l and, by Propositions 4.1 and 4.3, q does not decide for S', where $S' \neq S_p^l$. Thus, $S_p^l = S_q^l$. Since $T_p^{l-1} = T_q^{l-1}$ and, since p and q order the elements of S_p^l in the same deterministic order, $T_p^l = T_q^l$. □

PROPOSITION 4.5. *If process p makes the lth extension of the total order and process q makes the lth extension of the total order, then $S_p^l = S_q^l$ and $T_p^l = T_q^l$.*

Proof. The proof is by induction on l. If $l = 0$, then $S_p^0 = S_q^0 = \phi$ and $T_p^0 = T_q^0 = \phi$. We now assume the statement for $l - 1$. If process p makes the lth extension of the total order, it must have made the $(l - 1)$st extension, and similarly for q. By the inductive assumption, $T_p^{l-1} = T_q^{l-1}$. Consequently, by Proposition 4.4, $S_p^l = S_q^l$ and $T_p^l = T_q^l$. □

The main theorem, Theorem 4.1, guarantees that the total orders determined by different processes, even faulty processes, are consistent.

THEOREM 4.1. *If process p determines that m is the ith message in the total order, then process q does not determine that m' is the ith message, where $m' \neq m$.*

Proof. If process p determines that m is the ith message in the total order, then there exists an $l > 0$ such that m is in T_p^l and m is not in T_p^{l-1}. Similarly, if process q determines that m' is the ith message in the total order, then there exists an $l' > 0$ such that m' is in $T_p^{l'}$ and m' is not in $T_p^{l'-1}$. Without loss of generality, assume that $l' \geq l$. By Proposition 4.5, $T_p^l = T_q^l$. Thus, m is the ith message in $T_p^l = T_q^l$ and also the ith message in $T_q^{l'}$; so $m = m'$.

Theorem 4.2 guarantees that no message that follows a message in the partial order precedes that message in the total order. Thus, the total order is not arbitrary but is consistent with the partial order.

THEOREM 4.2. *If m' follows m in process p's total order prefix, where $m' \neq m$, then m does not follow m' in the partial order.*

Proof. Before process p can advance m to its total order prefix, m must become a candidate message. As a candidate message, m only follows messages in the partial order that are already in p's total order prefix. Since m' follows m in the total order, m' is not in process p's total order prefix when m becomes a candidate message. Thus, m does not follow m' in the partial order. □

Probabilistic Termination

To prove probabilistic termination we use the following definition.

A *deciding pattern* consists of three *ranks*, as shown in Fig. 2, such that:

1. Each rank contains $n - k$ messages from distinct nonfaulty processes, and all ranks contain messages from the same $n - k$ processes.
2. Each message in the second rank directly follows every message in the first rank and no other message.
3. Each message in the third rank directly follows every message in the second rank and no other message.

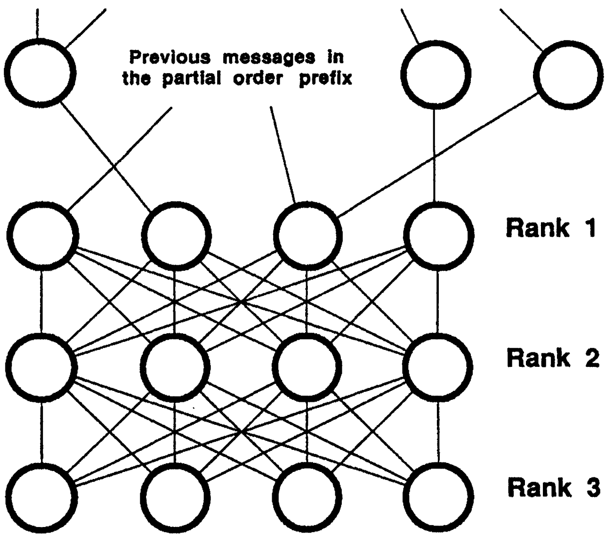


FIG. 2. *A deciding pattern of three ranks. Each rank contains $n - k$ messages from distinct nonfaulty processes, and all three ranks contain messages from the same $n - k$ processes. Here $n = 5$ and $k = 1$.*

A deciding pattern is only one of many possible partial order patterns that lead to a decision to extend the total order. This particular pattern was chosen because it is easy to analyze, for every message in the second rank follows exactly the same set of messages, aside from itself, and every message in the third rank follows exactly the same set of messages, aside from itself. This pattern is, however, quite improbable. Other partial order patterns exist that are much more likely and that lead to a decision to extend the total order just as quickly but that yield more complex proofs.

In Proposition 4.6 we show that there always exists a candidate set that can be advanced to the total order at the lth extension, i.e., a process cannot decide against all available candidate sets. However, in unfavorable circumstances, a process may take an arbitrarily large number of steps without reaching a decision, as the impossibility result requires.

PROPOSITION 4.6. *Let S^l be the largest set of candidate messages in the partial order for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. Then $S^l \neq \phi$ and, if process p makes the lth extension of its total order prefix T_p^{l-1}, then p cannot decide against S^l unless p has decided for a proper subset of S^l.*

Proof. For each process q consider the message m_{qj} in the partial order with smallest sequence number j that is not in process p's total order prefix T_p^{l-1} and that is followed by a message from a nonfaulty process. By the definition of nonfaulty, there exist at least $n - k$ such messages. Furthermore, there exists one or more such messages that do not follow any messages other than those in T_p^{l-1}. Thus, $S^l \neq \phi$.

If process p decides against S^l at stage i and p has not decided for a proper subset of S^l, then p determines that at least N_d messages vote against S^l at stage i. Now $N_d = (n+k+1)/2 > k$ and, therefore, at least one of those messages is from a nonfaulty process. Let m be such a message. By Lemma 4.4, m follows a message m' that votes against S^l at stage 0. Thus, m' follows a candidate message s such that $s \notin S^l$. But m also follows s, which contradicts the definition of S^l. □

The next three lemmas are used to prove Lemma 4.10, which states that there exists a largest i at which a message m or a previous message from its source votes on a candidate set S.

LEMMA 4.7. *If message m votes on a candidate set S at stage i, where $i > 0$, then m follows a message that votes on S at stage $i - 1$ but does not vote on S at stage i.*

Proof. If message m votes on S at stage i then, by the voting criteria, it follows at least two messages that vote on S at stage $i - 1$. Consider the set of all messages that m follows and that vote on S at stage $i - 1$. This set is finite (of cardinality at most n) and the follows relation is acyclic (except for cycles of length one). Thus, this set contains a message m' that follows no message that votes on S at stage $i - 1$, aside from itself. Consequently, m' does not vote on S at stage i. □

LEMMA 4.8. *If message m from process p votes on a candidate set S at stage j then, for each i such that $0 \leq i \leq j$, m or a previous message from p votes on S at stage i.*

Proof. The proof is by induction on j. If $j = 0$, then the statement holds since m follows itself. If $j = 1$, then m follows at least $N_v > 0$ messages that vote for S at stage 0 or at least $N_v > 0$ messages that vote against S at stage 0. Each of these messages follows all messages in S or a message not in S. Thus, m follows all messages in S or a message not in S. Consequently, m or a previous message from p votes on S at stage 0.

Now assume that the statement holds for $j - 1$, where $j > 1$. If m votes on S at stage j, then m follows at least $N_v > 0$ messages that vote for S at stage $j - 1$ or at least $N_v > 0$ messages that vote against S at stage $j - 1$. Each of these messages follows at least two messages that vote on S at stage $j-2$ and at least N_v messages that vote for S at stage $j-2$ or at least N_v messages that vote against S at stage $j-2$. Thus, m follows at least two messages that vote on S at stage $j - 2$ and at least N_v messages that vote for S at stage $j - 2$ or at least N_v messages that vote against S at stage $j - 2$. Consequently, m or a previous message from p votes on S at stage $j - 1$. The inductive assumption now gives the result. □

LEMMA 4.9. *If message m votes on a candidate set S at stage i, then m follows at least i distinct messages that vote on S at stages 0 through $i - 1$.*

Proof. The proof is by a simple induction on i using Lemmas 4.7 and 4.8. □

LEMMA 4.10. *If message m from process q follows each message in a candidate set S, then there exists an i such that m or a previous message from q votes on S at stage i and neither m nor any previous message from q votes on S at stage $i + 1$.*

Proof. The number of messages that precede message m in the partial order is finite, say x, and the number of messages that precede any previous message from process q is less than x. By Lemma 4.9, there exists a j (for example, $j = x + 1$) such that neither m nor any previous message from q votes on S at stage j. Since m follows each message in S, m or a previous message from q votes on S at stage 0. Thus, there exists an i, $0 \leq i < j$, such that m or a previous message from q votes on S at stage i and neither m nor any previous message from q votes on S at stage $i + 1$. □

The above lemma is used in the proof of Lemma 4.11, which demonstrates that a deciding pattern is indeed a deciding pattern.

LEMMA 4.11. *If the partial order prefix of process p contains a deciding pattern such that each message in its first rank follows all of the messages in a candidate set S, then process p decides on S.*

Proof. Every message in the second rank of the deciding pattern follows exactly the same set of messages, aside from itself. Thus, if any message in the second rank or a previous message from its source votes on S at stage i, then every message in the second rank or a previous message from its source votes on S at stage i. Consequently, by Lemma 4.10, there exists a largest stage i at which every message in the second rank or a previous message from its source votes on S.

Now every message in the third rank follows the same set of messages, aside from itself. In particular, each of these messages follows exactly the same set of at least $n - k$ messages that vote on S at stage i and follows no message that votes on S at stage $i + 1$, aside from itself. Consequently, by the voting criteria, all $n - k$ third rank messages vote for S at stage $i + 1$ or all vote against S at stage $i + 1$. By the decision criteria and induction on the cardinality of the candidate sets, process p decides on S at stage $i + 1$. □

Lemma 4.12 shows that the probability that a process decides on a candidate set increases with the size of its partial order prefix, while Proposition 4.7 shows that the probability that a process decides on a candidate set increases with the number of steps the process takes.

LEMMA 4.12. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process q. The probability that a partial order prefix of size x obtained by a nonfaulty process p contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as x tends to infinity.*

Proof. Consider a substructure of process p's partial order prefix consisting of three consecutive messages from each of $n - k$ nonfaulty processes. By the fairness properties, for each such substructure, the probability that the substructure is a deciding pattern is greater than some positive constant. As x tends to infinity, the number of such substructures tends to infinity. Thus, the probability that the partial order prefix contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as x tends to infinity. □

PROPOSITION 4.7. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process q and let p be a nonfaulty process. The probability that p's partial order prefix at step t contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as t tends to infinity.*

Proof. As the number t of steps taken by process p tends to infinity, the size x of p's partial order prefix tends to infinity. The statement now follows from Lemma 4.12. □

Lemma 4.13 provides the inductive step for Proposition 4.8, which shows that the probability that a process constructs the lth extension of the total order increases with the number of steps taken by that process. This leads directly to Theorems 4.3 and 4.4, which establish the probabilistic termination requirements.

LEMMA 4.13. *The probability that a nonfaulty process p selects a set S_p^l of candidate messages for its total order prefix T_p^l, contingent on its having constructed its total order prefix T_p^{l-1}, increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. Let S^l be the largest set of candidate messages in the partial order for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. By Proposition 4.6, p cannot decide against S^l unless p has decided for a proper subset of S^l. By Lemma 4.11, if all of the messages of S^l are followed by each of the messages of a deciding pattern, then p decides for S^l or for a proper subset of S^l. By Proposition 4.7, the probability that each message of S^l is followed by all of the messages of a deciding pattern tends to unity as the number of steps taken by p tends to infinity. □

PROPOSITION 4.8. *The probability that a nonfaulty process p selects a set S_p^l of candidate messages for the lth extension of the total order increases asymptotically to unity as the number of steps taken by p tends to infinity. Consequently, the probability that p constructs a total order prefix T_p^l increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The statement follows by induction on l from Lemma 4.13 and from elementary probability theory. □

THEOREM 4.3. *The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. By Proposition 4.8, for any given l, the probability that p constructs a total order prefix T_p^l increases asymptotically to unity as the number of steps taken by p tends to infinity. Choose an l such that $l \geq i$. Then the ith message in the total order is contained in T_p^l. □

THEOREM 4.4. *For each message m from a nonfaulty process q, the probability that a nonfaulty process p places message m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. Only messages that are followed by a message from a nonfaulty process can be included in p's total order prefix, and only such messages can be elements of a candidate set, each message of which is followed by a message from a nonfaulty process.

Since message m is broadcast by a nonfaulty process q, m is followed by a message from each nonfaulty process. Since each such message follows a finite number of messages in the partial order, there exists a finite number, say l, of messages that are followed by a message from a nonfaulty process and that do not follow m. By Proposition 4.8, the probability that p selects a set S_p^l of candidate messages for the lth extension of the total order increases asymptotically to unity as the number of steps taken by p tends to infinity. At the lth extension, either m is already in the total order prefix T_p^{l-1} or all messages that are followed by a message from a nonfaulty process and that do not follow m are in T_p^{l-1}. In the latter case, m is the only candidate message and is selected for the lth extension. □

5. The $n/2$ resilient algorithm.

For this algorithm, we assume that the resilience $k < n/2$. The algorithm is defined by the following voting, proposing and deciding criteria; these criteria determine which candidate set is chosen for inclusion in the total order.

- Each process, before determining the vote or proposal of a message m on a candidate set S for the lth extension of the total order, must obtain a prefix A of the partial order such that $m \in A$.
- A message can vote or propose on a candidate set S for the lth extension of the total order at stage i only if no previous message from its source has already done so at stage i.

- The number of proposals required for a decision, the number of votes required for a proposal, and the number of messages related to an indifferent proposal must be at least N_d, N_p, and N_v, respectively, where

$$N_d = k + 1, \quad N_p = (n + 1)/2, \quad \text{and} \quad N_v = n - k.$$

The Criteria for Voting on a Candidate Set *S* for the *l*th Extension

At stage 0,

- A message votes for S if it follows every message in S and it follows no other candidate message. (A candidate message votes for the set containing only itself.)
- A message votes against S if it follows a candidate message not in S. (A candidate message votes against all sets of which it is not a member.)

At stage i, where $i > 0$,

- A message votes for S if
  - it follows a message that proposes for S at stage $i - 1$.
- A message votes against S if
  - it follows a message that proposes against S at stage $i - 1$,

  or
  - it follows no message that proposes for or against S at stage $i - 1$, and
  - it follows at least N_v messages that propose indifferent to S at stage $i - 1$.

The Criteria for Proposing on a Candidate Set *S* for the *l*th Extension

At stage i, where $i \geq 0$,

- A message proposes for S if
  - it follows at least N_p messages that vote for S at stage i.
- A message proposes against S if
  - it follows at least N_p messages that vote against S at stage i.
- A message proposes indifferent to S if
  - it does not propose for or against S at stage i, and
  - it follows at least N_v messages that vote on S at stage i.

The Criteria for Deciding on a Candidate Set *S* for the *l*th Extension

At stage i, where $i \geq 0$,

- A process decides for S if
  - it determines that at least N_d messages propose for S at stage i, and
  - for each proper subset of S, it decides against that proper subset.
- A process decides against S if
  - it determines that at least N_d messages propose against S at stage i, or
  - it decides for a proper subset of S.

The numbers of votes, which determine the values of N_d and N_v, are based on the following properties: Decisions do not conflict ($N_d > k$), proposals do not conflict ($2 * N_p > n$), decisions are feasible ($N_d \leq n-k$), proposals for or against are feasible ($N_p \leq n-k$ and $N_p \leq N_v$), and stages of voting and proposing advance ($N_v \leq n - k$). These inequalities lead to the above requirements and also to the constraint $k < n/2$.

The proof for the $n/2$ Total algorithm is generally similar to that for the $n/3$ algorithm and can be found in the Appendix.

6. Complexity analysis. The primary complexity measures of a total ordering algorithm are the number of messages broadcast per message ordered and the latency or delay between broadcasting a message and placing it in the total order.

The Total algorithms are based on a partial order that can be constructed from acknowledgments of messages that most broadcast protocols provide. Using an appropriate broadcast protocol that piggybacks acknowledgments on messages, no additional messages are required beyond retransmissions that are needed because messages are not received immediately [11]. The cost is thus one message broadcast per message placed in the total order. The algorithm of Chang and Maxemchuk [4] typically requires about three broadcast messages, while other algorithms [3], [12] require a multiple of n messages in an n-process system.

If the partial order is broad and contains many unordered pairs of messages, the analysis of the latency of the Total algorithms is hard and is an area of continuing research. If, however, the partial order is derived from messages broadcast over a network for which there is a high probability that messages are received immediately by all processes, then it is likely that substantial portions of the partial order are totally ordered with only occasional pairs of unordered messages. Under these conditions, it is probable that there is only a single candidate message since all other unordered messages follow that message. Subsequent messages vote for the single candidate message at stage 0, and the latency is equal to the number of messages in the partial order prefix when enough messages have voted at stage 0 to satisfy the decision criteria.

In our demonstration of probabilistic termination we made the weakest assumptions about the probability distributions for messages from different processes, assumptions of independence and nonzero probability. A complexity analysis, however, requires stronger assumptions; we now assume that every process is equally likely to broadcast at each moment. This allows us to calculate a mean latency as an accurate measure of performance on a broadcast medium that delivers almost all messages immediately.

Consider first the $n/3$ resilient algorithm. To place a message m in the total order, $\lceil (n+k+1)/2 \rceil$ messages from distinct processes must vote for the singleton set containing m. The first of these is the message m itself and, thus, the latency is the expected number of messages to obtain $\lceil (n+k-1)/2 \rceil$ messages from the $n-1$ other processes that vote for m. If i messages from distinct processes have voted, then the probability that the next message is from one of the remaining $n-i$ processes is $(n-i)/n$. Consequently, the expected number of messages to obtain the $(i+1)$st vote is

$$1 * \frac{n-i}{n} + 2 * \frac{i}{n} * \frac{n-i}{n} + 3 * \left(\frac{i}{n}\right)^2 * \frac{n-i}{n} + \ldots = \frac{n}{n-i}$$

and the expected latency of the $n/3$ resilient algorithm is

$$\sum_{i=1}^{\lceil \frac{n+k-1}{2} \rceil} \frac{n}{n-i}$$

broadcast messages.

For the $n/2$ resilient algorithm, first $\lceil (n+1)/2 \rceil$ votes must be obtained to propose and then $k+1$ proposals must be obtained to decide. By an analysis similar to the above, the expected latency of the $n/2$ resilient algorithm is

$$\sum_{i=1}^{\lceil \frac{n-1}{2} \rceil} \frac{n}{n-i} + \sum_{i=1}^{k} \frac{n}{n-i}$$

broadcast messages.

LEMMA 6.1. *If none of the processes is faulty and n is odd and k is odd or n is even and k is even, the expected latency for the $n/2$ algorithm is less than that for the $n/3$ algorithm, whereas if n is odd and k is even, or n is even and k is odd, the expected latency for the $n/3$ algorithm is less than that for the $n/2$ algorithm.*

Proof. The proof consists of a case analysis and is omitted. □

The above analysis assumes that none of the processes is faulty. If k of the n processes are faulty and do not broadcast, the expected latency is

$$\sum_{i=1}^{\lceil \frac{n+k-1}{2} \rceil} \frac{n-k}{n-k-i}$$

broadcast messages for the $n/3$ resilient algorithm and

$$\sum_{i=1}^{\lceil \frac{n-1}{2} \rceil} \frac{n-k}{n-k-i} + \sum_{i=1}^{k} \frac{n-k}{n-k-i}$$

broadcast messages for the $n/2$ resilient algorithm.

LEMMA 6.2. *If k of the n processes are faulty and contribute no messages to the partial order and if n is odd and k is odd, n is odd and k is even, or n is even and k is even, then the expected latency for the $n/2$ algorithm is less than that for the $n/3$ algorithm, whereas if n is even and $k = 1$, then the expected latency for the $n/3$ algorithm is less than that for the $n/2$ algorithm.*

Proof. The proof consists of a case analysis and is omitted. □

The analysis in the case that n is even and k is odd is difficult in general because $\sum_{i=1}^{k} \frac{n-k}{n-k-i}$ has one more term than $\sum_{j=1}^{\frac{k-1}{2}} \frac{2(n-k)}{n-2k-2j}$. However, we have the following conjecture.

CONJECTURE. *If k of the n processes are faulty and contribute no messages to the partial order and if n is even and k is odd, then the expected latency for the $n/2$ algorithm is less than that for the $n/3$ algorithm if and only if $n \leq k(k+1) + 2\lfloor (k-1)/4 \rfloor$.*

We have written a Pascal program to compute the latencies for the $n/3$ and $n/2$ algorithms and have verified this conjecture for all n and k such that $n \leq 2^{15}$ and $k < n/3$, but we have not been able to find an analytical proof.

7. Impossibility of a terminating algorithm.

It is easy to demonstrate that it is impossible to construct a totally correct algorithm (i.e., an algorithm that terminates) for converting a partial order on messages into a total order in an asynchronous distributed system with n processes ($n \geq 2$) in which even a single process is faulty. The proof depends on the well-known result of Fischer, Lynch, and Paterson [8], who demonstrated that no algorithm using asynchronous processes and asynchronous FIFO communication channels can be guaranteed to reach a consensus decision in a finite number of steps if even one process can fail.

Given an asynchronous distributed system with FIFO communication channels it is easy to construct a partial order on messages. Simplistically, each process includes in its message all of the messages it has received and that, thus, precede that message in the partial order. More efficient strategies are, of course, available [2], [11]. Given a consistent total order on messages, processes can easily reach a consensus decision; the content of the first message in the total order is the consensus value on which they decide [6]. If there exists a totally correct algorithm for converting a partial order on messages into a total order in an asynchronous distributed system in which even a single process is faulty, then that algorithm can be combined with the two algorithms above to yield a totally correct algorithm for reaching consensus in an asynchronous distributed system with FIFO channels, in contradiction to [8].

8. Conclusion. The justification of the Total algorithms is not just that they provide an elegant solution to the total ordering problem, but that they have practical application to distributed systems that utilize broadcast communication over a local area network. Because local area networks are quite reliable, almost every decision is a best case decision in which there is only one candidate message, and the decision is reached at stage 0 with minimal delay and little computational cost. Only one broadcast message is required for each decision plus any additional retransmissions that may be required due to messages not being received immediately. We have implemented the Total algorithms on a network of Sun workstations and have built a fault-tolerant distributed database on top of them.

Appendix–The Proof for the $n/2$ Resilient Algorithm

Partial Correctness

The proof of partial correctness involves showing that, if processes p and q make lth extensions to their total order prefixes, then the candidate sets they choose for those extensions are identical and, thus, their lth total order prefixes are identical, i.e., $S_p^l = S_q^l$ and $T_p^l = T_q^l$. The proof is by induction on l. The base case is trivial since, if $l = 0$, these sets are all empty. The bulk of the proof constitutes establishing the inductive step.

As in the $n/3$ resilient case, we have the following lemma which states that the only information used by the algorithm is the partial order on messages. By "determine the vote (proposal) of a message," we mean determine if the message voted (proposed) for or against a candidate set or was unable to vote (propose).

LEMMA 5.1. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p and q each determine the vote of a message m on a candidate set S for the lth extension of the total order at stage i, then they determine the same vote of m on S at stage i.*

Proof. The proof is similar to that of Lemma 4.1. □

LEMMA 5.2. *Each process p broadcasts at most one message that votes on a candidate set S at stage i; that message does not vote both for and against S at stage i.*

Proof. Since process p broadcasts at most one message that votes on S at stage i, it suffices to show that if a message m votes for (against) a candidate set S at stage i, then m does not vote against (for) S at stage i. The proof is by induction on i. For $i = 0$, the voting criteria imply the statement.

We now assume the statement for $i - 1$ and argue by contradiction to establish the statement for i. Thus, we assume that m votes both for and against S at stage i. By the voting criteria, since m votes for S at stage i, it follows a message that proposes for S at stage $i - 1$. Consequently, since m votes against S at stage i, it follows a message that proposes against S at stage $i - 1$. By the proposing criteria, m follows $N \geq N_p$ messages that vote against S at stage $i - 1$. Moreover, the inductive assumption implies that those messages do not vote for S at stage $i - 1$. Thus, the number of messages that vote for S at stage $i - 1$ is at most $n - N \leq (n - 1)/2 < (n + 1)/2 = N_p$. Consequently, no message proposes for S at stage $i - 1$, which is a contradiction. □

LEMMA 5.2.1. *If message m votes against (for) a candidate set S at stage i, then m follows a message that votes against (for) S at stage $i - 1$, where $i > 0$.*

Proof. Assume that m votes against S at stage i and that m follows no message that votes against S at stage $i - 1$. Then, by the proposing criteria, m follows no message that proposes against S at stage $i - 1$. But m votes against S at stage i. Therefore, by the voting criteria, m follows at least N_v messages that propose indifferent to S at stage $i - 1$. Hence, by the proposing criteria, m follows $N \geq N_v$ messages that vote on S at stage $i - 1$. If they all vote for S then, since $N_v = n - k \geq (n + 1)/2 = N_p$ (since $k < n/2$), m proposes for S at stage $i - 1$ and, therefore, votes for S at stage i, which is a contradiction. Consequently, at least one of the N messages votes against S at stage $i - 1$. The proof in the case that m votes for S at stage i is obvious. □

LEMMA 5.2.2. *If message m proposes for (against) a candidate set S at stage i, then no message proposes against (for) S at stage i.*

Proof. The statement follows from Lemma 5.2 and the fact that $N_p + N_p > n$. □

LEMMA 5.2.3. *Each process p broadcasts at most one message that proposes on a candidate set S at stage i; that message proposes only once at stage i.*

Proof. The first statement follows from the requirement that a message proposes on a candidate set S at stage i only if no previous message from its source has proposed on S at stage i. The second statement is given by Lemma 5.2.2. □

LEMMA 5.3. *If a message follows at least N_d messages that propose for (against) a candidate set S at stage $i - 1$, then no message votes against (for) S at stage i, where $i > 0$.*

Proof. Assume that message m' votes against S at stage i. Then, either (1) m' follows a message that proposes against S at stage $i - 1$, or (2) m' follows at least N_v messages that propose indifferent to S at stage $i - 1$. By the hypothesis and Lemma 5.2.2, case (1) cannot occur. In case (2), by the hypothesis and Lemma 5.2.3, at most $n - N_d = n - k - 1 < n - k = N_v$ messages propose indifferent to S at stage $i - 1$, which is a contradiction. The proof for the alternative statement of the lemma is similar but simpler, because only case (1) arises. □

LEMMA 5.4. *If message m votes against (for) a candidate set S at stage j then, for each i such that $0 \leq i \leq j$, m follows a message that votes against (for) S at stage i.*

Proof. The proof is similar to that of Lemma 4.4 and uses Lemma 5.2.1 in place of the voting criteria. □

LEMMA 5.5. *If no message votes against (for) a candidate set S at stage i, then no message votes against (for) S at stage j, where $j \geq i$. Likewise, if process p decides for (against) a candidate set S at stage i, then no message proposes against (for) S at stage j, where $j > i$.*

Proof. The proof of the first statement is identical to that of the first statement of Lemma 4.5. To prove the second statement, we note that if process p decides for a candidate set S at stage i, then p determines that at least N_d messages propose for S at stage i. By Lemma 5.3, no message votes against S at stage $i + 1$. By the first statement, no message votes against S at stage j, where $j \geq i + 1$. Therefore, by the proposing criteria, no message proposes against S at stage j, where $j > i$. □

PROPOSITION 5.1. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p decides for a candidate set S for the lth extension of the total order, then q does not decide for a proper subset S' of S for the lth extension.*

Proof. The proof is similar to that of Proposition 4.1 but is based on Lemma 5.1, Lemma 5.2.2, and Lemma 5.5. □

PROPOSITION 5.2. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$. If p decides for (against) a candidate set S for the lth extension of the total order, then q does not decide against (for) S for the lth extension.*

Proof. The proof is similar to that of Proposition 4.2 but is based on Proposition 5.1, Lemma 5.1, Lemma 5.2.2, and Lemma 5.5. □

The statement of Lemma 5.6 is weaker than that of Lemma 4.6, but it suffices to prove Lemma 5.6.1, which is needed to prove Proposition 5.3 below.

LEMMA 5.6. *Let S and S' denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$, and $s' \in S'$, $s' \notin S$. If message m from process p votes for S at stage i, then no message from p votes for S' at stage i.*

Proof. The proof is by induction on i. If message m from process p votes for S at stage 0, then it follows the candidate message $s \in S$ and no previous message from p votes on S. Since m follows s and $s \notin S'$, m votes against S' at stage 0. By Lemma 5.2, no message from p votes for S' at stage 0.

We now assume the statement for $i-1$. If m votes for S at stage i then, by the voting and proposing criteria, m follows $N \geq N_p$ messages from distinct processes that vote for S at stage $i-1$. By the inductive assumption, for each of those messages, no message from its source votes for S' at stage $i-1$. Thus, at most $n - N \leq n - N_p = (n-1)/2 < (n+1)/2 = N_p$ messages from distinct processes vote for S' at stage $i-1$. Consequently, by the proposing criteria, no message proposes for S' at stage $i-1$ and, by the voting criteria, no message votes for S' at stage i. □

LEMMA 5.6.1. *Let S and S' denote candidate sets for the lth extension of the total order such that there exist $s \in S$, $s \notin S'$, and $s' \in S'$, $s' \notin S$. If message m proposes for S at stage i, then no message proposes for S' at stage i.*

Proof. If m proposes for S at stage i then, by the proposing criteria, at least N_p messages from distinct processes vote for S at stage i. By Lemma 5.6, for each of those messages, no message from its source votes for S' at stage i. Thus, at most $n - N_p = (n-1)/2 < (n+1)/2 = N_p$ messages from distinct processes vote for S' at stage i. By the proposing criteria, no message proposes for S' at stage i. □

PROPOSITION 5.3. *Let p and q be processes such that $T_p^{l-1} = T_q^{l-1}$, where $l > 0$, and let S and S' denote candidate sets for the lth extension of the total order such that there exist messages $s \in S$, $s \notin S'$, and $s' \in S'$, $s' \notin S$. If process p decides for S for the lth extension of the total order, then process q does not decide for S' for the lth extension.*

Proof. If process p decides for S at stage i then, by the decision criteria, p determines that a message (in fact, at least N_d messages) proposes for S at stage i. Thus, by Lemma 5.6.1, no message proposes for S' at stage i. Consequently, by the decision criteria, process q does not decide for S' at stage i. □

PROPOSITION 5.4. *If process p makes the lth extension of its total order prefix T_p^{l-1} by including the candidate set S_p^l, and if process q makes the lth extension of its total order prefix T_q^{l-1} by including the candidate set S_q^l, and if $T_p^{l-1} = T_q^{l-1}$, then $S_p^l = S_q^l$ and, consequently, $T_p^l = T_q^l$.*

Proof. The proof is identical to that of Proposition 4.4. □

PROPOSITION 5.5. *If process p makes the lth extension of the total order and process q makes the lth extension of the total order, then $S_p^l = S_q^l$ and $T_p^l = T_q^l$.*

Proof. The proof is identical to that of Proposition 4.5. □

The main theorem, Theorem 5.1, guarantees that the total orders determined by different processes, even faulty processes, are consistent.

THEOREM 5.1. *If process p determines that m is the ith message in the total order, then process q does not determine that m' is the ith message, where $m' \neq m$.*

Proof. The proof is identical to that of Theorem 4.1. □

Theorem 5.2 guarantees that no message that follows a message in the partial order precedes that message in the total order. Thus, the total order is not arbitrary but is consistent with the partial order.

THEOREM 5.2. *If m' follows m in process p's total order prefix, where $m' \neq m$, then m does not follow m' in the partial order.*

Proof. The proof is identical to that of Theorem 4.2. □

Probabilistic Termination

To prove probabilistic termination we use the following definition.

A *deciding pattern* consists of five *ranks* such that:

1. Each rank contains $n-k$ messages from distinct nonfaulty processes, and all ranks contain messages from the same $n-k$ processes.
2. Each message in the second rank directly follows every message in the first rank and no other message.

3. Each message in the third rank directly follows every message in the second rank and no other message.
4. Each message in the fourth rank directly follows every message in the third rank and no other message.
5. Each message in the fifth rank directly follows every message in the fourth rank and no other message.

In Proposition 5.6 we show that a process cannot decide against all available candidate sets.

PROPOSITION 5.6. *Let S^l be the largest set of candidate messages in the partial order for the lth extension of the total order, each of which is followed by a message from a nonfaulty process. Then $S^l \neq \phi$ and, if process p makes the lth extension of its total order prefix T_p^{l-1}, then p cannot decide against S^l unless p has decided for a proper subset of S^l.*

Proof. The proof that $S^l \neq \phi$ is identical to the corresponding proof of Proposition 4.6. Now, if process p decides against S^l at stage i and p has not decided for a proper subset of S^l, then p determines that at least N_d messages propose against S^l at stage i. Each of those messages follows at least N_p messages that vote against S^l at stage i. Since $N_p = (n+1)/2 > k$ (since $k < n/2$), at least one of those messages is from a nonfaulty process. Let m be such a message. By Lemma 5.4, m follows a message m' that votes against S^l at stage 0. Thus, m' follows a candidate message s such that $s \notin S^l$. But, m also follows s, which contradicts the definition of S^l. □

The next three lemmas are used to prove Lemma 5.10, which states that there exists a largest i at which a message m or a previous message from its source votes on a candidate set S.

LEMMA 5.7. *If message m votes on a candidate set S at stage i, where $i > 0$, then m follows a message that votes on S at stage $i - 1$ but does not vote on S at stage i.*

Proof. Consider the set of all messages that m follows and that vote on S at stage $i - 1$. This set is finite (of cardinality at most n) and the follows relation is acyclic (except for cycles of length one). Thus, this set contains a message m' that follows no message that votes on S at stage $i - 1$, aside from itself. Since $N_p = (n + 1)/2 > 1$ and $N_v = n - k > 1$, m' does not propose on S at stage $i - 1$ and thus does not vote on S at stage i. □

LEMMA 5.8. *If message m from process p votes on a candidate set S at stage j then, for each i such that $0 \leq i \leq j$, m or a previous message from p votes on S at stage i.*

Proof. The proof is by induction on j. If $j = 0$, then the statement holds since m follows itself. If $j = 1$, then three cases arise: (1) m follows a message m' that proposes for S at stage 0 and, thus, m' follows at least N_p messages that vote for S at stage 0, (2) m follows a message m' that proposes against S at stage 0 and, thus, m' follows at least N_p messages that vote against S at stage 0, or (3) m follows at least N_v messages that propose indifferent to S at stage 0, each of which follows at least N_v messages that vote on S at stage 0. In each case, m follows a message that votes on S at stage 0 and, thus, m follows every message in S or it follows a message that is not in S. Consequently, m or a previous message from p votes on S at stage 0.

Now assume that the statement holds for $j - 1$, where $j > 1$. If m votes on S at stage j, then three cases arise: (1) m follows a message m' that proposes for S at stage $j - 1$ and, thus, m' follows at least N_p messages that vote for S at stage $j - 1$, each of which follows a message that proposes for S at stage $j - 2$, (2) m follows a message m' that proposes against S at stage $j - 1$ and, thus, m' follows at least N_p messages that vote against S at stage $j - 1$, each of which follows a message that proposes against S at stage $j - 2$ or follows at least N_v messages that propose indifferent to S at stage $j - 2$, or (3) m follows at least N_v messages that propose indifferent to S at stage $j - 1$, each of which follows at least N_v messages that vote on S at stage $j-1$; each of these messages either follows a message that proposes for S at stage $j-2$, follows a message that proposes against S at stage $j-2$, or follows at least N_v messages that propose indifferent to S at stage $j-2$. In each case, m or a previous message from p votes on S at stage $j-1$. The inductive assumption now gives the result. □

LEMMA 5.9. *If message m votes on a candidate set S at stage i, then m follows at least i distinct messages that vote on S at stages* 0 *through $i-1$.*

Proof. The proof is identical to that of Lemma 4.9. □

LEMMA 5.10. *If message m from process q follows each message in a candidate set S, then there exists an i such that m or a previous message from q votes on S at stage i and neither m nor any previous message from q votes on S at stage $i+1$.*

Proof. The proof is identical to that of Lemma 4.10. □

The above lemma is used in the proof of Lemma 5.11, which demonstrates that a deciding pattern is indeed a deciding pattern.

LEMMA 5.11. *If the partial order prefix of process p contains a deciding pattern such that each message in its first rank follows all of the messages in a candidate set S, then process p decides on S.*

Proof. Every message in the second rank of the deciding pattern follows exactly the same set of messages, aside from itself. Thus, if any message in the second rank or a previous message from its source votes on S at stage i, then every message in the second rank or a previous message from its source votes on S at stage i. Consequently, by Lemma 5.10, there exists a largest stage i at which every message in the second rank or a previous message from its source votes on S.

If any message in the second rank proposes on S at stage i, then all messages in the second rank propose on S at stage i and, furthermore, they all propose the same way. Four cases arise: (1) All messages in the second rank propose for S or all propose against S at stage i. This results in a decision at stage i based on the proposals by messages in the second rank. (2) All messages in the second rank propose indifferent to S at stage i. In this case, all messages in the third rank vote against S at stage $i+1$, and all messages in the fourth rank propose against S at stage $i+1$, resulting in a decision against S at stage $i+1$. (3) No message in the second rank proposes on S at stage i, and all messages in the third rank propose for S or all propose against S at stage i, resulting in a decision at stage i. (4) No message in the second rank proposes on S at stage i and all messages in the third rank propose indifferent to S at stage i. In this case, all messages in the fourth rank vote against S at stage $i+1$, and all messages in the fifth rank propose against S at stage $i+1$, which results in a decision against S at stage $i+1$. □

Lemma 5.12 shows that the probability that a process decides on a candidate set increases with the size of its partial order prefix, while Proposition 5.7 shows that the probability that a process decides on a candidate set increases with the number of steps the process takes.

LEMMA 5.12. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process q. The probability that a partial order prefix of size x obtained by a nonfaulty process p contains a deciding pattern, such that each message in its first rank follows all of the messages in S, increases asymptotically to unity as x tends to infinity.*

Proof. The proof is identical to that of Lemma 4.12. □

PROPOSITION 5.7. *Let S be a candidate set, each message of which is followed by a message from a nonfaulty process q, and let p be a nonfaulty process. The probability that p's partial order prefix at step t contains a deciding pattern, such that each message in its first rank follows all of the messages in a candidate set S, increases asymptotically to unity as t tends to infinity.*

Proof. The proof is identical to that of Proposition 4.7. □

Lemma 5.13 provides the inductive step for Proposition 5.8, which shows that the probability that a process constructs the lth extension of the total order increases with the number of steps taken by that process. This leads directly to Theorems 5.3 and 5.4, which establish the probabilistic termination requirements.

LEMMA 5.13. *The probability that a nonfaulty process p selects a set S_p^l of candidate messages for its total order prefix T_p^l, contingent on its having constructed its total order prefix T_p^{l-1}, increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Lemma 4.13. □

PROPOSITION 5.8. *The probability that a nonfaulty process p selects a set S_p^l of candidate messages for the lth extension of the total order increases asymptotically to unity as the number of steps taken by p tends to infinity. Consequently, the probability that p constructs a total order prefix T_p^l increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Proposition 4.8. □

THEOREM 5.3. *The probability that a nonfaulty process p places an ith message in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Theorem 4.3. □

THEOREM 5.4. *For each message m from a nonfaulty process q, the probability that a nonfaulty process p places message m in the total order increases asymptotically to unity as the number of steps taken by p tends to infinity.*

Proof. The proof is identical to that of Theorem 4.4. □

Acknowledgment. We wish to thank the referees for their valuable comments and criticisms which have greatly improved this paper.

REFERENCES

[1] M. BEN-OR, *Another advantage of free choice: Completely asynchronous agreement protocols*, in Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, August 1986, pp. 27–30.

[2] K. P. BIRMAN AND T. A. JOSEPH, *Reliable communication in the presence of failures*, ACM Trans. Comput. Systems, 5 (1987), pp. 47–76.

[3] G. BRACHA AND S. TOUEG, *Asynchronous consensus and broadcast protocols*, J. ACM, 32 (1985), pp. 824–840.

[4] J. CHANG AND N. F. MAXEMCHUK, *Reliable broadcast protocols*, ACM Trans. Comput. Systems, 2 (1984), pp. 251–273.

[5] F. CRISTIAN, H. AGHILI, AND R. STRONG, *Atomic broadcast: From simple message diffusion to Byzantine agreement*, in Proceedings of the IEEE Symposium on Fault Tolerant Comput. Systems, June 1985, pp. 200–206.

[6] D. DOLEV, C. DWORK, AND L. STOCKMEYER, *On the minimal synchronism needed for distributed consensus*, J. ACM, 34 (1987), pp. 77–97.

[7] P. D. EZHILCHELVAN, *Early stopping algorithms for distributed agreement under fail-stop, omission and timing faults*, in Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems, March 1987, pp. 201–212.

[8] M. J. FISCHER, N. A. LYNCH, AND M. S. PATERSON, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[9] L. LAMPORT, *Time, clocks and the ordering of events in a distributed system*, Comm. ACM, 21 (1978), pp. 558–565.

[10] L. LAMPORT, R. SHOSTAK, AND M. PEASE, *The Byzantine generals problem*, ACM Trans. Programming Languages and Systems, 4 (1982), pp. 382–401.

[11] P. M. MELLIAR-SMITH, L. E. MOSER, AND V. AGRAWALA, *Broadcast protocols for distributed systems*, IEEE Trans. Parallel and Distributed Systems, 1 (1990), pp. 17–25.

[12] K. J. PERRY AND S. TOUEG, *Distributed agreement in the presence of processor and communication faults*, IEEE Trans. Software Engrg., SE-12 (1986), pp. 477–482.

[13] L. L. Peterson, N. C. Buchholz, and R. D. Schlichting, *Preserving and using context information in interprocess communication*, ACM Trans. Comput. Systems, 7 (1989), pp. 217–246.

[14] M. Rabin, *Randomized Byzantine generals*, in Proceedings of the 24th Symposium on Foundations of Computer Science, November 1983, pp. 403–409.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 751–777, August 1993

A FIBONACCI VERSION OF KRAFT'S INEQUALITY APPLIED TO DISCRETE UNIMODAL SEARCH*

ARTHUR S. GOLDSTEIN† AND EDWARD M. REINGOLD†

Abstract. A function is unimodal if it strictly increases to a unique maximum and then strictly decreases. The problem of determining the smallest possible interval containing the maximum of a unimodal function, by probing only at integer values is studied. In the finite case, the search takes place over the range 0 to N, while in the infinite case the search takes place over the nonnegative integers. The analyses are based on an unusual Fibonacci version of Kraft's inequality.

Key words. unimodal search, Fibonacci search, Kraft's inequality, Fibonacci numbers, unimodal functions, unbounded search, optimal algorithms, Ackermann's function, inverse Ackermann's function

AMS subject classifications. 68Q25, 68Q20, 11B39

1. Introduction.

A function is *unimodal* if it strictly increases to a unique maximum and then strictly decreases; that is, a function f is unimodal if there exists a unique value M such that $f(x) < f(y)$ if $x < y \leq M$ and $f(x) > f(y)$ if $M \leq x < y$. Kiefer [5] studied the problem of finding the maximum of a unimodal function over the unit interval and gave an algorithm that, in n function evaluations (probes), narrows the interval containing the maximum to at most $1/F_{n+1} + \epsilon$, for any fixed $\epsilon > 0$, where F_i is the ith Fibonacci number defined by $F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$. Kiefer proved that his algorithm is optimal in the sense that any algorithm using n probes that narrows the interval containing the maximum to at most $1/F_{n+1}$, for some functions, must fail to narrow the interval to at most $1/F_{n+1} + \epsilon$ for other functions. Note that finding the maximum of a differentiable unimodal function is equivalent to finding the zero of its first derivative.

Oliver and Wilde [11] considered a discrete search for the maximum of a unimodal function: All probes must be at least δ apart, but the probes need not be at integer values. They gave an algorithm that uses n function evaluations over the unit interval to narrow the interval containing the maximum to $(1 + \delta F_{n-1})/F_{n+1}$, where $\delta \leq 1/F_{n+2}$. Avriel and Wilde [3] proved Oliver and Wilde's algorithm optimal for $\delta = 1/F_{n+2}$, in the sense that any other algorithm must fail to narrow the interval to $(1 + \delta F_{n-1})/F_{n+1}$ for some function. Witzgall [15] considered the case when the first probe occurs at a prescribed value.

We study a form of the *discrete unimodal search problem*—determining the smallest possible interval containing the maximum of a unimodal function, by probing only at integer values. In the finite case, the search takes place over the range 0 to N and we assume that $f(0) = f(N) = -\infty$. Oliver and Wilde's algorithm [11] can solve this discrete unimodal search problem in n function evaluations, when scaled to an initial interval of length F_{n+2}; our results, based on a Fibonacci version of Kraft's inequality [9], give a new derivation of the results in [3] and [11], and extend them to intervals that are not of Fibonacci length. Furthermore, the new techniques we use to analyze the finite case are applied to the infinite case, the *unbounded discrete unimodal search problem*, introduced by Raoult and Vuillemin [12], in which the search takes place over the nonnegative integers and we assume that $f(0) = \lim_{N\to\infty} f(N) = -\infty$.

Note that a single function value gives no information about the location of the maximum. Two function values, however, do give information, as shown in Fig. 1: For $i < j$, if $f(i) < f(j)$ then the maximum occurs to the right of i, while if $f(i) > f(j)$ then the maximum occurs to the left of j; if $f(i) = f(j)$ then the maximum occurs between i and j. Thus

*Received by the editors October 9, 1990; accepted for publication April 24, 1992.

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

if $f(i) < f(i+1) > f(i+2)$, we know that the maximum occurs between i and $i+2$ and we can narrow the interval no further (by probes at integer values) since the maximum could occur on either side of $i+1$ as shown in Fig. 2. Only in the special cases when either $f(i) = f(i+1)$ or $f(i+1) = f(i+2)$ can we narrow the interval further to $i < M < i+1$ or $i+1 < M < i+2$, respectively.

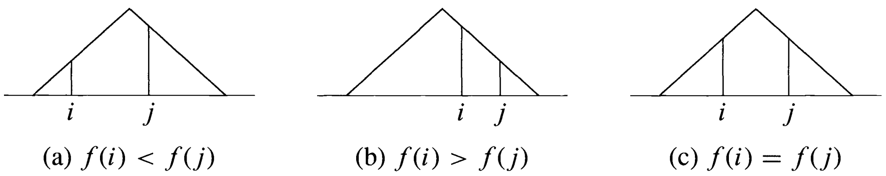


FIG. 1. *A unimodal function $f(x)$ and probes at i and j.*

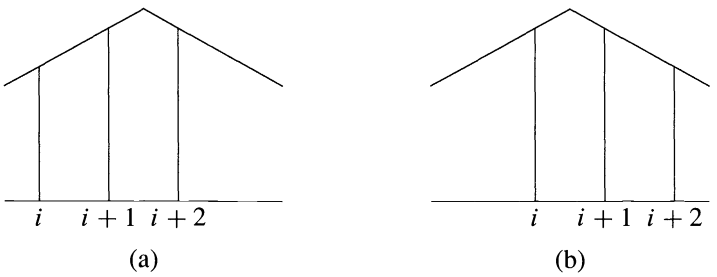


FIG. 2. *Unimodal functions f with $f(i) < f(i+1) > f(i+2)$. In* (a) *the maximum occurs between i and $i+1$; in* (b) *the maximum occurs between $i+1$ and $i+2$.*

We define the *cost* of i, $c(i)$, to be the number of probes needed by an algorithm, in the worst case, to find the smallest possible interval containing the maximum when $i \le M < i+1$. If the search range is finite, we want a search algorithm that minimizes $\max_{0 \le i < N} c(i)$. When the search range is infinite, we try to minimize the growth rate of $c(i)$, as $i \to \infty$. Although it is possible to compute a lower bound on $c(i)$ by induction on N in the finite case [3], [5], [11], we employ new techniques that apply to the unbounded search problem as well.

2. Finite unimodal search trees. An algorithm that solves the finite discrete unimodal search problem can be represented as a unimodal search tree as shown in Fig. 3. Each node has the form $[i, j, k]$ with $i < j < k$, meaning that the algorithm has found that $f(i) \le f(j) \ge f(k)$ and hence that the maximum occurs between i and k. The left and right children of a node $[i, j, k]$ are $[i, x, j]$ and $[x, j, k]$, respectively, if the next value probed, x, satisfies $i < x < j$; otherwise, if $j < x < k$, the left and right children of $[i, j, k]$ are $[i, j, x]$ and $[j, x, k]$, respectively. The root of the tree is $[0, i_0, N]$, where i_0 is the first value probed and the interval searched is from 0 to N. External nodes have the form $[i, i+1, i+2]$, meaning that the algorithm has determined that the maximum occurs in the interval $i < M < i+2$. Note that there are no redundant probes in the tree.

The cost of i, $c(i)$, is one more than the longest distance from the root to a node of the form $[i-1, i, i+1]$ or $[i, i+1, i+2]$, since these are the only two nodes corresponding to $i \le M < i+1$. To derive a lower bound on $c(i)$, $0 \le i < N$, we will compute a lower bound on the height of the search tree.

To motivate our approach to the determination of a lower bound on $c(i)$, consider the problem of determining the unit interval containing the zero of a strictly increasing function

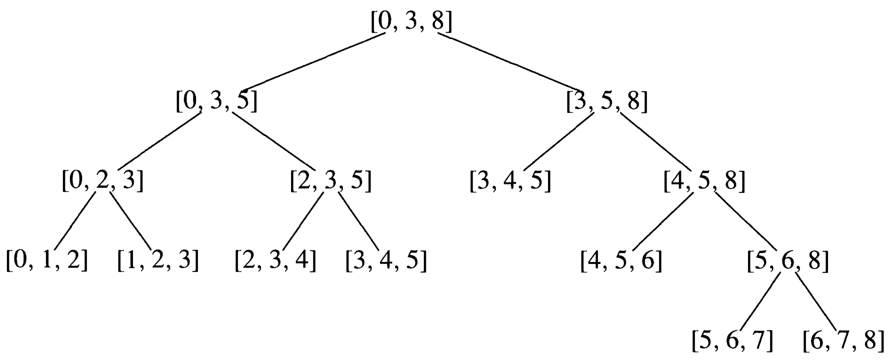


FIG. 3. *A unimodal search tree with $N = 8$. The first two probes of the algorithm are at* 3 *and* 5. *If $f(3) > f(5)$ the next probe is at* 2; *otherwise the next probe is at* 4. *We have $c(0) = c(1) = c(2) = c(3) = c(4) = 4$ and $c(5) = c(6) = c(7) = 5$.*

with probes only at integer values; that is, we have $f(x) < f(y)$ for $x < y$ and we want to find i such that $f(i) \leq 0$ and $f(i+1) > 0$. In this case, too, the cost of i, $c(i)$, is the number of probes, in the worst case, needed by an algorithm to find i. For the bounded version of this problem, in which we minimize $\max_{0 \leq i < N} c(i)$, we would use simple binary search (see [7, pp. 406–422], for example). The unbounded version of this problem, in which we try to minimize the growth rate of $c(i)$, as $i \to \infty$, has been studied extensively by Bentley and Yao [4], Knuth [8], Reingold and Shen [13], [14], and others. Lower bounds for both the bounded and unbounded search follow from Kraft's inequality [9] (see [1, pp. 53–59] or [8]):

$$\sum_{i=0}^{N-1} \frac{1}{2^{c(i)}} \leq 1. \tag{1}$$

We will prove a Fibonacci version of Kraft's inequality for unimodal search trees and hence obtain lower bounds on the number of probes. Specifically, we prove the following theorem in §3.

THEOREM 1. *In a unimodal search tree, let $[A, x, Z]$ be a node at distance l from the root, where x is any value such that $A < x < Z$, and let q be a value in the range $A \leq q < Z$ having the least cost in the tree; that is, for all i, $A \leq i < Z$, $c(q) \leq c(i)$. Then*

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}. \tag{2}$$

This theorem is similar to the following, easily proved, observation: In a binary tree, let u be a node at distance l from the root and let $E(u)$ be the set of external nodes that are descendents of u. If the cost of node i, $c(i)$, is its distance from the root and $v \in E(u)$ is an external node of least cost among $E(u)$, then

$$\sum_{i \in E(u)} \frac{1}{2^{c(i)}} \leq \frac{2^{c(v)-l}}{2^{c(v)}} = \frac{1}{2^l}.$$

Kraft's inequality (1) follows from this observation by taking u as the root ($l = 0$). Similarly, from Theorem 1 we obtain the following corollary.

COROLLARY 1. *A unimodal search tree for the range* 0 *to N satisfies*

$$\frac{N}{F_{N+1}} \leq \sum_{i=0}^{N-1} \frac{1}{F_{c(i)+2}} \leq 1. \tag{3}$$

Proof. The upper bound follows from Theorem 1 by taking $[A, x, Z]$ as the root $[0, i_0, N]$, that is, taking $l = 0$.

To prove the lower bound, we give a unimodal search tree for the range $[0, N]$ in which every path from the root to a leaf has length $N - 2$, that is, probes at every integer in the range $[1, N-1]$; the case $N = 5$ is shown in Fig. 4. The root of the search tree is $[0, 1, N]$. In general, every node at a distance k from the root has the form $[i, i+1, i+N-k]$ or $[i, i+N-k-1, i+N-k]$. In either case, we let the children be $[i, i+1, i+N-k-1]$ and $[i+1, i+N-k-1, i+N-k]$. Each external node corresponds to $k = N-2$ and is at a distance $N-2$ from the root. Thus each of the N terms in the sum in (3) is of the form $1/F_{N+1}$. Since there can be at most $N - 1$ nonredundant probes for any path in any search algorithm, no other tree can have deeper external nodes or a smaller sum. □

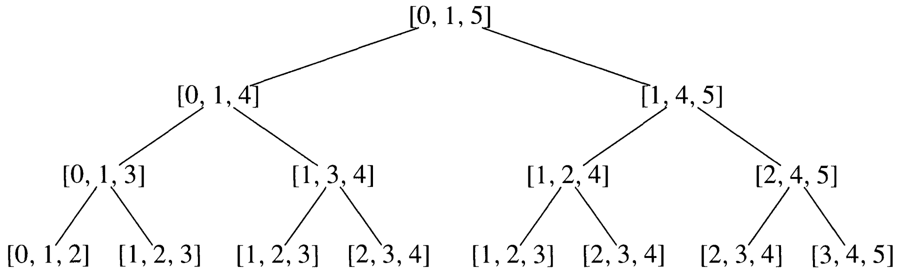


FIG. 4. *A worst-case unimodal search tree with* $N = 5$. *No matter where the maximum is, the algorithm corresponding to such a tree will use* $N - 1$ *probes.*

The upper bound in Corollary 1 is similar to the inequality

$$\int_0^N \varphi^{-c(x)} dx \le \frac{2}{\varphi}, \tag{4}$$

from [12], in which $c(x)$ is the number of probes, in the worst case, used by an algorithm to locate the maximum to within an interval of length two; $\varphi = (1+\sqrt{5})/2$ is the golden ratio. However, Corollary 1 is tighter than (4) in the sense that (3) is tight for infinitely many N (the Fibonacci numbers), but the integral in (4) cannot equal $2/\varphi$ for $N > 2$.

A lower bound on the number of probes in an algorithm for the finite unimodal search problem follows from Corollary 1. Let $F^{-1}(n)$ be the *inverse Fibonacci function* defined by

$$F_{F^{-1}(n)-1} < n \le F_{F^{-1}(n)};$$

note that

$$\frac{\varphi^{F^{-1}(n)-1} - (-\hat{\varphi})^{F^{-1}(n)-1}}{\varphi + \hat{\varphi}} < n \le \frac{\varphi^{F^{-1}(n)} - (-\hat{\varphi})^{F^{-1}(n)}}{\varphi + \hat{\varphi}},$$

where $\hat{\varphi} = 1/\varphi = \varphi - 1 = (-1+\sqrt{5})/2$. Taking logarithms and using Taylor series gives

$$\begin{aligned} &F^{-1}(n) - 1 - (-\hat{\varphi})^{F^{-1}(n)-1} \frac{1}{\varphi^{F^{-1}(n)-1} \ln \varphi} + O\left(\frac{1}{\varphi^{F^{-1}(n)-1+1}}\right) - \log_\varphi(\varphi + \hat{\varphi}) \\ &< \log_\varphi n \\ &\le F^{-1}(n) - (-\hat{\varphi})^{F^{-1}(n)} \frac{1}{\varphi^{F^{-1}(n)} \ln \varphi} + O\left(\frac{1}{\varphi^{F^{-1}(n)+1}}\right) - \log_\varphi(\varphi + \hat{\varphi}), \end{aligned}$$

and therefore

$$F^{-1}(n) = \log_\varphi n + O(1).$$

We have the following corollary.

COROLLARY 2. *For any discrete unimodal search algorithm over the range* 0 *to* N, *the number of probes needed in the worst case is at least* $F^{-1}(N) - 2$.

Proof. If the maximum cost of an algorithm is at most $F_{F^{-1}(N)-3}$, we have

$$\sum_{i=0}^{N-1} \frac{1}{F_{c(i)+2}} \geq \frac{N}{F_{F^{-1}(N)-1}} > 1,$$

contradicting Corollary 1. □

The lower bound in Corollary 2 is achievable, as we show in §4 .

3. Proof of Theorem 1.

3.1. Some Fibonacci identities.

We need the basic identities

$$F_{n+m} = F_m F_{n+1} + F_{m-1} F_n \tag{5}$$

and

$$F_{n+1} F_{n-1} - F_n^2 = (-1)^n; \tag{6}$$

see Knuth [6, pp. 78–86], for example. With these identities we prove

$$F_{n+m-1} F_n - F_{n-1} F_{n+m} = F_m (-1)^{n+1} \tag{7}$$

as follows:

$$\begin{aligned} &F_{n+m-1} F_n - F_{n-1} F_{n+m} \\ &\quad = F_n^2 F_m + F_{n-1} F_{m-1} F_n - F_{n-1} F_m F_{n+1} - F_{n-1} F_{m-1} F_n \end{aligned}$$

by (5), which simplifies to

$$= F_m (F_n^2 - F_{n-1} F_{n+1}),$$

and by (6) we get

$$= F_m (-1)^{n+1}.$$

PROPOSITION 1. *For* $i \geq 0$, $j > 0$, *and* $k > 0$, *if* i *is odd then*

$$\frac{F_i}{F_{i+k}} > \frac{F_{i+j}}{F_{i+j+k}},$$

and if i *is even then*

$$\frac{F_i}{F_{i+k}} < \frac{F_{i+j}}{F_{i+j+k}}.$$

Proof. The relations can be written jointly as

$$F_i F_{i+j+k} \overset{?}{<>} F_{i+k} F_{i+j},$$

which by (5) becomes

$$F_i F_k F_{i+j+1} + F_i F_{k-1} F_{i+j} \overset{?}{<>} F_k F_{i+1} F_{i+j} + F_{k-1} F_i F_{i+j}.$$

Simplifying, we get

$$F_i F_{i+j+1} \overset{?}{<>} F_{i+1} F_{i+j},$$

and rewriting,

$$F_i F_{i+j+1} - F_{i+1} F_{i+j} \overset{?}{<>} 0.$$

By (7), with $n = i + 1$ and $m = j$, this becomes

$$F_j(-1)^{i+1} \overset{?}{<>} 0.$$

Thus the term on the left is larger if i is odd and smaller if i is even. □

Remark. Proposition 1 can be restated as

$$\frac{2}{F_i} > \frac{5}{F_{i+2}} > \frac{13}{F_{i+4}} > \ldots > \varphi^{-i+3} > \ldots > \frac{21}{F_{i+5}} > \frac{8}{F_{i+3}} > \frac{3}{F_{i+1}},$$

for $i \geq 4$.

PROPOSITION 2. *For $i \geq 0$, $j \geq 0$, and $k > 0$,*

$$\frac{F_i}{F_{i+k}} - \frac{F_{i+j}}{F_{i+j+k}} = \frac{F_j F_k (-1)^{i+1}}{F_{i+k} F_{i+j+k}}.$$

Proof. We have

$$\frac{F_i}{F_{i+k}} - \frac{F_{i+j}}{F_{i+j+k}} = \frac{F_{i+j+k} F_i - F_{i+j} F_{i+k}}{F_{i+k} F_{i+j+k}},$$

which, by (5),

$$= \frac{F_{i+k} F_{j+1} F_i + F_{i+k-1} F_j F_i - F_i F_{j+1} F_{i+k} - F_{i-1} F_j F_{i+k}}{F_{i+k} F_{i+j+k}}.$$

Simplifying, we get

$$= \frac{F_j \left(F_{i+k-1} F_i - F_{i-1} F_{i+k}\right)}{F_{i+k} F_{i+j+k}},$$

which, by (7),

$$= \frac{F_j F_k (-1)^{i+1}}{F_{i+k} F_{i+j+k}},$$

as desired. □

3.2. Inductive proof of Theorem 1. For convenience, we restate the theorem here.

THEOREM 1. *In a unimodal search tree, let $[A, x, Z]$ be a node at distance l from the root, where x is any value such that $A < x < Z$, and let q be a value in the range $A \leq q < Z$ having the least cost in the tree; that is, for all i, $A \leq i < Z$, $c(q) \leq c(i)$. Then*

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

We prove Theorem 1 by induction on the height of the subtree rooted at a node. The basis of the induction is an external node for which this height is zero. For internal nodes, we will assume that Theorem 1 is true for all descendents of a node; there will be two different cases depending on the parity of $c(q) - l$.

3.2.1. Basis of the induction. Consider an external node $[A, A + 1, A + 2]$, that is, $Z = A + 2$, at distance l from the root. If $c(A + 1) \geq c(A) \geq l + 1$, then $q = A$ and

$$\frac{1}{F_{c(A)+2}} + \frac{1}{F_{c(A+1)+2}} \leq \frac{2}{F_{c(A)+2}} \leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}};$$

otherwise $c(A) \geq c(A + 1) \geq l + 1$, implying $q = A + 1$ and

$$\frac{1}{F_{c(A)+2}} + \frac{1}{F_{c(A+1)+2}} \leq \frac{2}{F_{c(A+1)+2}} \leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

This establishes the basis of the induction for Theorem 1.

3.2.2. Induction when $c(q)-l$ is odd. Let $c(q)-l > 0$ be odd for internal node $[A, x, Z]$ at distance l from the root. We consider the case when we have a subtree of the form in Fig. 5 (or its mirror image, in which case the arguments that follow are easily adjusted); in the degenerate case, $[A, C, E]$ is an external node. First note that by applying Theorem 1 inductively to node $[C, E, Z]$ we get

$$\sum_{i=C}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(q)+1-l}}{F_{c(q)+2}}. \tag{8}$$

If $[A, C, E]$ is an external node then, since $c(A) \geq c(q) > l$, we have

$$\begin{aligned}\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} &= \frac{1}{F_{c(A)+2}} + \sum_{i=C}^{Z-1} \frac{1}{F_{c(i)+2}} \\ &\leq \frac{1}{F_{c(A)+2}} + \frac{F_{c(q)+1-l}}{F_{c(q)+2}} \\ &\leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}},\end{aligned}$$

proving Theorem 1.

If $[A, C, E]$ is an internal node, by applying Theorem 1 to node $[A, B, D]$ we get

$$\sum_{i=A}^{D-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(r)-l}}{F_{c(r)+2}}, \tag{9}$$

for some r, $A \leq r < D$, with $c(r) \geq c(q)$ by the definition of q. Since $c(q) - l$ is odd, we have, by Proposition 1,

$$\frac{F_{c(r)-l}}{F_{c(r)+2}} \leq \frac{F_{c(q)-l}}{F_{c(q)+2}}. \tag{10}$$

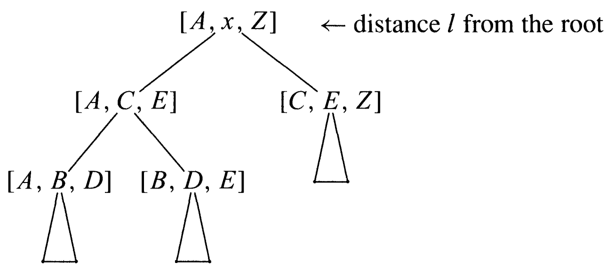


FIG. 5. *Subtree for the case $c(q) - l$ odd. The value of C is either B or D; the value of x is either C or E. The value with least cost among A through $Z - 1$, q, is contained in the subtree rooted at [C, E, Z]. In the degenerate case, [A, C, E] may be an external node.*

Since $C \le D$,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \le \sum_{i=A}^{D-1} \frac{1}{F_{c(i)+2}} + \sum_{i=C}^{Z-1} \frac{1}{F_{c(i)+2}},$$

and from (8) and (9)

$$\le \frac{F_{c(r)-l}}{F_{c(r)+2}} + \frac{F_{c(q)+1-l}}{F_{c(q)+2}},$$

so by (10)

$$\le \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)+1-l}}{F_{c(q)+2}}$$
$$= \frac{F_{c(q)+2-l}}{F_{c(q)+2}},$$

proving Theorem 1 for $c(q) - l$ odd.

Note that inequality (10) does not hold if $c(q) - l$ is even and therefore a separate case is needed.

3.2.3. Induction when $c(q) - l$ is even. Let $c(q) - l > 0$ be even for internal node $[A, x, Z]$ at distance l from the root, so $c(q) > l + 1$. We consider the case when we have a subtree of the form in Fig. 6 (or its mirror image, in which case the arguments given below are easily adjusted); in various degenerate cases, one or more of the internal nodes shown in Fig. 6 are external nodes. First note the following identities: applying Theorem 1 inductively to node $[D, H, Z]$ we get

$$\sum_{i=D}^{Z-1} \frac{1}{F_{c(i)+2}} \le \frac{F_{c(q)+1-l}}{F_{c(q)+2}}; \tag{11}$$

applying Theorem 1 to $[A, D, H]$,

$$\sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} \le \frac{F_{c(r)+1-l}}{F_{c(r)+2}},$$

for some r, $A \le r < H$ with $c(r) \ge c(q)$. Since $c(q) - l$ is even, $c(q) - l + 1$ is odd and we have, by Proposition 1,

$$\sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} \le \frac{F_{c(q)+1-l}}{F_{c(q)+2}}; \tag{12}$$

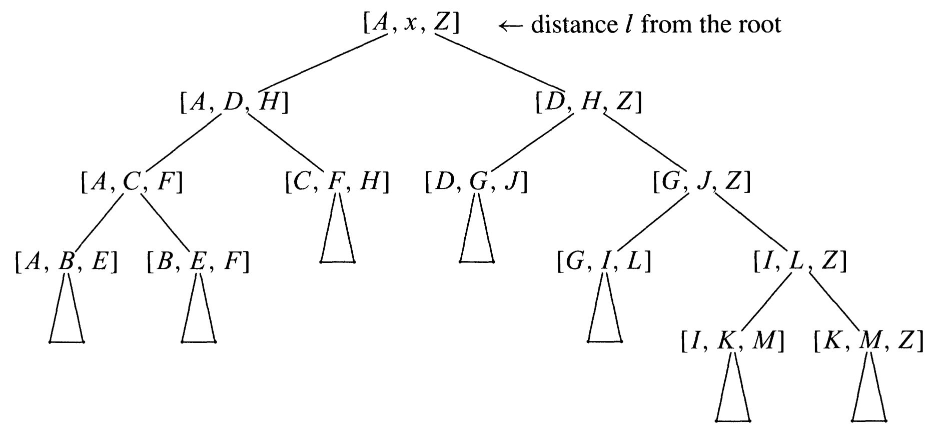


FIG. 6. *Subtree for the case $c(q) - l$ even. The value of C is either B or E; D is C or F; L is K or M; J is I or L; H is G or J; x is D or H. The value of least cost among A through $Z-1$, q, is contained in the subtree rooted at $[D, H, Z]$. In various degenerate cases, some of the nodes shown as internal nodes may actually be external nodes.*

applying Theorem 1 to $[G, I, L]$,

$$\sum_{i=G}^{L-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(s)-1-l}}{F_{c(s)+2}},$$

for some s, $G \leq s < L$, with $c(s) \geq c(q)$. Since $c(q) - 1 - l$ is odd, we have, by Proposition 1,

$$\sum_{i=G}^{L-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(q)-1-l}}{F_{c(q)+2}}. \tag{13}$$

If $[A, D, H]$ is an external node, then

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} = \frac{1}{F_{c(A)+2}} + \sum_{i=D}^{Z-1} \frac{1}{F_{c(i)+2}},$$

which, by (11) and noting that $c(A) \geq c(q) > l + 1$,

$$\begin{aligned} &\leq \frac{1}{F_{c(A)+2}} + \frac{F_{c(q)+1-l}}{F_{c(q)+2}} \\ &\leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}. \end{aligned}$$

Otherwise, if $[A, D, H]$ is an internal node but $[D, H, Z]$ is an external node,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} = \sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} + \frac{1}{F_{c(H)+2}},$$

which, by (12) and noting that $c(H) \geq c(q) > l + 1$,

$$\begin{aligned} &\leq \frac{F_{c(q)+1-l}}{F_{c(q)+2}} + \frac{1}{F_{c(H)+2}} \\ &\leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}. \end{aligned}$$

Otherwise, if $[A, D, H]$ and $[D, H, Z]$ are both internal nodes but $[A, C, F]$ is an external node,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \le \frac{1}{F_{c(A)+2}} + \frac{1}{F_{c(C)+2}} + \sum_{i=D}^{Z-1} \frac{1}{F_{c(i)+2}},$$

which, by (11) and noting that $c(A) \ge c(q)$, $c(C) \ge c(q)$, and $c(q) > l+1$,

$$\le \frac{1}{F_{c(A)+2}} + \frac{1}{F_{c(C)+2}} + \frac{F_{c(q)+1-l}}{F_{c(q)+2}}$$

$$\le \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

Otherwise, if $[A, D, H]$, $[D, H, Z]$, and $[A, C, F]$ are all internal nodes but $[G, J, Z]$ is an external node,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \le \sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} + \frac{1}{F_{c(G)+2}} + \frac{1}{F_{c(J)+2}}$$

which, by (12) and noting that $c(G) \ge c(q)$, $c(J) \ge c(q)$, and $c(q) > l+1$,

$$\le \frac{F_{c(q)+1-l}}{F_{c(q)+2}} + \frac{1}{F_{c(G)+2}} + \frac{1}{F_{c(J)+2}}$$

$$\le \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

Otherwise, if $[A, D, H]$, $[D, H, Z]$, $[A, C, F]$, and $[G, J, Z]$ are all internal nodes but $[I, L, Z]$ is an external node, then since $G \le H$,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \le \sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} + \sum_{i=G}^{L-1} \frac{1}{F_{c(i)+2}} + \frac{1}{F_{c(L)+2}},$$

which, by (12), (13), and noting that $c(L) \ge c(q) > l+1$,

$$\le \frac{F_{c(q)+1-l}}{F_{c(q)+2}} + \frac{F_{c(q)-1-l}}{F_{c(q)+2}} + \frac{1}{F_{c(L)+2}}$$

$$\le \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

Having handled all possible degenerate cases, we may now assume that all the nodes in Fig. 6 are present. We outline the remainder of the proof to give a clear picture of what is being done and to show that the proof is complete. We know by hypothesis that $D \le q < Z$. We consider the case $G \le q < Z$ and show that Theorem 1 holds. Next we assume $D \le q < J$ and consider the least cost value s such that $G \le s < Z$ with $c(s) > c(q)$; otherwise we can set $q = s$ and revert to the previous case $G \le q < Z$. We show Theorem 1 holds if $c(s) \ge c(q) + 2$, leaving only the case $c(s) = c(q) + 1$. We divide this case into two subcases. In the first subcase we show Theorem 1 holds if $I \le s < Z$; in the second subcase we show Theorem 1 holds if $G \le s < L$ and the least cost value t such that $I \le t < Z$ satisfies $c(t) > c(s)$; if not, we can set $s = t$ and revert to the previous case $I \le s < Z$.

If $G \leq q < Z$ then since $G \leq H$,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \sum_{i=A}^{H-1} \frac{1}{F_{c(i)+2}} + \sum_{i=G}^{Z-1} \frac{1}{F_{c(i)+2}},$$

and by (12) and applying Theorem 1 to $[G, J, Z]$ we get

$$\leq \frac{F_{c(q)+1-l}}{F_{c(q)+2}} + \frac{F_{c(q)-1-l}}{F_{c(q)+2}}$$

$$\leq \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

For the remainder of the proof, assume $D \leq q < J$ and that the least cost value s such that $G \leq s < Z$, satisfies $c(s) > c(q)$. Note the important observation that if we apply Theorem 1 and Proposition 1 to $[A, B, E]$ we get

$$\sum_{i=A}^{E-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(r)-1-l}}{F_{c(r)+2}} \leq \frac{F_{c(q)+1-l}}{F_{c(q)+4}}, \tag{14}$$

for some r, $A \leq r < E$, with $c(r) > c(q)$, for if $c(r) = c(q)$ we would have the mirror image of the case $G \leq q < Z$.

If $c(s) \geq c(q) + 2$, then applying Theorem 1 and Proposition 1 to $[G, J, Z]$ we get

$$\sum_{i=E}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \frac{F_{c(s)-l}}{F_{c(s)+2}} \leq \frac{F_{c(q)+3-l}}{F_{c(q)+5}}. \tag{15}$$

Because $C \leq E$ and $G \leq H$,

$$\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} \leq \sum_{i=A}^{E-1} \frac{1}{F_{c(i)+2}} + \sum_{i=C}^{H-1} \frac{1}{F_{c(i)+2}} + \sum_{i=G}^{Z-1} \frac{1}{F_{c(i)+2}};$$

hence by (14) and (15) and applying Theorem 1 to $[C, F, H]$ we get

$$\leq \frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)+3-l}}{F_{c(q)+5}}.$$

Therefore, we must prove

$$\frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)+3-l}}{F_{c(q)+5}} \overset{?}{\leq} \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

Subtracting $F_{c(q)-l}/F_{c(q)+2}$ from both sides this becomes

$$\frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)+3-l}}{F_{c(q)+5}} \overset{?}{\leq} \frac{F_{c(q)+1-l}}{F_{c(q)+2}}.$$

Writing $F_{c(q)+1-l}$ as $F_{c(q)-l} + F_{c(q)-1-l}$ we get

$$\frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)+3-l}}{F_{c(q)+5}} \overset{?}{\leq} \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)-1-l}}{F_{c(q)+2}},$$

and rewriting,

$$\frac{F_{c(q)+3-l}}{F_{c(q)+5}} - \frac{F_{c(q)-l}}{F_{c(q)+2}} \overset{?}{\leq} \frac{F_{c(q)-1-l}}{F_{c(q)+2}} - \frac{F_{c(q)+1-l}}{F_{c(q)+4}}.$$

Applying Proposition 2 to both sides gives

$$-\frac{2F_{2+l}(-1)^{c(q)+1-l}}{F_{c(q)+2}F_{c(q)+5}} \overset{?}{\leq} \frac{F_{l+3}(-1)^{c(q)-l}}{F_{c(q)+4}F_{c(q)+2}}.$$

Canceling $1/F_{c(q)+2}$ from both sides and recalling that $c(q) - l$ is even we get

$$\frac{2F_{l+2}}{F_{c(q)+5}} \overset{?}{\leq} \frac{F_{l+3}}{F_{c(q)+4}},$$

which we can rewrite as

$$\frac{F_{c(q)+4}}{F_{c(q)+5}} \overset{?}{\leq} \frac{F_{l+3}}{2F_{l+2}},$$

or, by writing F_{l+3} as $F_{l+2} + F_{l+1}$,

$$\frac{F_{c(q)+4}}{F_{c(q)+5}} \overset{?}{\leq} \frac{1}{2} + \frac{F_{l+1}}{2F_{l+2}}.$$

By Proposition 1 the term on the left is at most 5/8, whereas the term on the right is at least 3/4 and the relation is proven.

We can now assume that $c(s) = c(q) + 1$. If $F \leq s < Z$, then by applying Theorem 1 and Proposition 1 to $[A, C, F]$ in a manner similar to the derivation of (14), applying Theorem 1 to $[D, G, J]$, and applying Theorem 1 to $[I, L, Z]$, we get

$$\begin{aligned}
\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} &\leq \sum_{i=A}^{F-1} \frac{1}{F_{c(i)+2}} + \sum_{i=D}^{J-1} \frac{1}{F_{c(i)+2}} + \sum_{i=I}^{Z-1} \frac{1}{F_{c(i)+2}} \\
&\leq \frac{F_{c(q)+1-l}}{F_{c(q)+3}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)-l}}{F_{c(q)+3}} \\
&\leq \frac{F_{c(q)+2-l}}{F_{c(q)+3}} + \frac{F_{c(q)-l}}{F_{c(q)+2}}.
\end{aligned}$$

We must prove

$$\frac{F_{c(q)+2-l}}{F_{c(q)+3}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} \overset{?}{\leq} \frac{F_{c(q)+2-l}}{F_{c(q)+2}},$$

which simplifies to

$$\frac{F_{c(q)+2-l}}{F_{c(q)+3}} \overset{?}{\leq} \frac{F_{c(q)+1-l}}{F_{c(q)+2}}.$$

This inequality follows from Proposition 1 because $c(q) - l$ is even.

The final case is $c(s) = c(q) + 1$, $G \leq s < J$, and the least cost value t such that $I \leq t < Z$, satisfies $c(t) > c(s)$. Then by (14), applying Theorem 1 to $[C, H, E]$, applying

Theorem 1 to $[G, I, L]$, and applying Theorem 1 and Proposition 1 to $[K, M, Z]$ in a manner similar to the derivation of (14), we get

$$\begin{aligned}\sum_{i=A}^{Z-1} \frac{1}{F_{c(i)+2}} &\le \sum_{i=A}^{E-1} \frac{1}{F_{c(i)+2}} + \sum_{i=C}^{H-1} \frac{1}{F_{c(i)+2}} + \sum_{i=G}^{L-1} \frac{1}{F_{c(i)+2}} + \sum_{i=K}^{Z-1} \frac{1}{F_{c(i)+2}} \\ &\le \frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)-l}}{F_{c(q)+3}} + \frac{F_{c(q)+1-l}}{F_{c(q)+5}}.\end{aligned}$$

We must prove

$$\frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)-l}}{F_{c(q)+2}} + \frac{F_{c(q)-l}}{F_{c(q)+3}} + \frac{F_{c(q)+1-l}}{F_{c(q)+5}} \overset{?}{\le} \frac{F_{c(q)+2-l}}{F_{c(q)+2}}.$$

Subtracting $F_{c(q)-l}/F_{c(q)+2}$ from both sides we get

$$\frac{F_{c(q)+1-l}}{F_{c(q)+4}} + \frac{F_{c(q)-l}}{F_{c(q)+3}} + \frac{F_{c(q)+1-l}}{F_{c(q)+5}} \overset{?}{\le} \frac{F_{c(q)+1-l}}{F_{c(q)+2}}.$$

The term on the right is equal to

$$\frac{2F_{c(q)-1-l}}{F_{c(q)+2}} + \frac{F_{c(q)-2-l}}{F_{c(q)+2}},$$

so rearranging terms gives

$$\frac{F_{c(q)+1-l}}{F_{c(q)+5}} - \frac{F_{c(q)-2-l}}{F_{c(q)+2}} \overset{?}{\le} \frac{F_{c(q)-1-l}}{F_{c(q)+2}} - \frac{F_{c(q)+2-l}}{F_{c(q)+4}} + \frac{F_{c(q)-1-l}}{F_{c(q)+2}} - \frac{F_{c(q)-l}}{F_{c(q)+3}}.$$

Applying Proposition 2 gives

$$-\frac{2F_{l+4}(-1)^{c(q)-1-l}}{F_{c(q)+5}F_{c(q)+2}} \overset{?}{\le} \frac{F_{l+3}(-1)^{c(q)-l}}{F_{c(q)+2}F_{c(q)+4}} + \frac{F_{l+3}(-1)^{c(q)-l}}{F_{c(q)+2}F_{c(q)+3}}.$$

Noting that $c(q) - l$ is even and canceling $1/F_{c(q)+2}$ gives

$$\frac{2F_{l+4}}{F_{c(q)+5}} \overset{?}{\le} F_{l+3}\left(\frac{1}{F_{c(q)+4}} + \frac{1}{F_{c(q)+3}}\right).$$

Dividing by F_{l+3} and multiplying by $F_{c(q)+5}$, we get

$$\frac{2F_{l+4}}{F_{l+3}} \overset{?}{\le} \frac{F_{c(q)+5}}{F_{c(q)+4}} + \frac{F_{c(q)+5}}{F_{c(q)+3}},$$

and expanding the numerators,

$$2 + \frac{2F_{l+2}}{F_{l+3}} \overset{?}{\le} 3 + \frac{F_{c(q)+3}}{F_{c(q)+4}} + \frac{F_{c(q)+2}}{F_{c(q)+3}}.$$

The term on the left is at most $2 + 2(1/1) = 4$ and the term on the right is at least $3 + (2/3) + (1/2) = 25/6 > 4$, so the inequality holds.

The proof of Theorem 1 is now complete.

4. An optimal finite unimodal search algorithm. Procedure *UnimodalSearch*, given in Algorithm 1, solves the finite discrete unimodal search problem by finding the unique maximum of a unimodal function f over the interval (i, k). *UnimodalSearch* can be described inductively as follows: Given i and k, let $j > 0$ be a value such that $F_j < k - i \leq F_{j+2}$ and assume we know that $f(i) \leq f(i + F_j) \geq f(k)$. If $k - i > F_{j+1}$, the next probe of f is made at $i + F_{j+1}$; otherwise the next probe is at $i + F_{j-1}$. The appropriate subinterval is then searched recursively until an interval of length 2 is obtained.

ALGORITHM 1

Optimal discrete unimodal search algorithm.

```
**procedure** $UnimodalSearch($
    **var** $i$: *NonnegativeInteger*; { Upon entry, search has reached... }
    $j, k$: *NonnegativeInteger*;  { ... node $[i, i + F_j, k]$ }
    **var** $CurrentMax$: *real*; { Upon entry, value of $f(i + F_j)$ }
    $f$: **function**(*NonnegativeInteger*): *real*); { Unimodal function to search }
{ Upon return, the maximum of $f$ is in the range $[i, i + 1, i + 2]$
  and $CurrentMax = f(i + 1)$ }
**var**
  $NewValue$: *real*;
**begin** { $UnimodalSearch$ }
  { **invariant**: $CurrentMax = f(i + F_j)$ is maximum of all probes
                   and $F_j < k - i \leq F_{j+2}$ }
  **if** $k - i = 2$ **then**
    { $UnimodalSearch$ ends at interval $[i, i + 1, i + 2]$ }
  **else if** $k - i \leq F_{j+1}$ **then begin**
    $NewValue := f(i + F_{j-1})$;
    **if** $NewValue > CurrentMax$ **then begin**
      $CurrentMax := NewValue$;
      $UnimodalSearch(i, j - 1, i + F_j, CurrentMax, f)$
      **end**
    **else begin**
      $i := i + F_{j-1}$;
      $UnimodalSearch(i, j - 2, k, CurrentMax, f)$
      **end**
    **end**
  **else** { $k - i > F_{j+1}$ } **begin**
    $NewValue := f(i + F_{j+1})$;
    **if** $NewValue > CurrentMax$ **then begin**
      $CurrentMax := NewValue$;
      $i := i + F_j$;
      $UnimodalSearch(i, j - 1, k, CurrentMax, f)$
      **end**
    **else**
      $UnimodalSearch(i, j, i + F_{j+1}, CurrentMax, f)$
    **end**
**end**; { $UnimodalSearch$ }
```

ALGORITHM 2
Optimal discrete unimodal search over the interval (0, N).

```
function FiniteUnimodalSearch(
    N: PositiveInteger; { Search over interval (0, N), $N \geq 2$ }
    f: function(NonnegativeInteger): real { Unimodal function to be searched }
    ): NonnegativeInteger; { Left endpoint of a length-two interval
                             containing the unique maximum of f }
procedure UnimodalSearch
    ⋮ { See Algorithm 1 }
end;
var
    i: NonnegativeInteger; { Left endpoint of interval containing the maximum }
    Maximum: real; { Largest value of f found in the probes }
begin { FiniteUnimodalSearch }
    i := 0;
    Maximum := $f(F_{F^{-1}(N)-1})$;
    UnimodalSearch (i, $F^{-1}(N) - 1$, N, Maximum, f);
    return(i)
end; { FiniteUnimodalSearch }
```

LEMMA 1. *The invariant always holds as stated in $UnimodalSearch$, when proper initial values are given.*

Proof. The invariant says

$$CurrentMax = f(i + F_j) \text{ is maximum of all probes and } F_j < k - i \leq F_{j+2}.$$

We first remark that $CurrentMax$ is only changed if the new value probed is larger and so therefore it remains the maximum of all probes.

We prove the other part of the invariant by induction. Let i_s, j_s, and k_s be the values of the parameters i, j, and k, respectively, for the sth call to $UnimodalSearch$. By hypothesis, i_0, j_0, and k_0 are such that the invariant holds. Assume the invariant is true at the sth call to $UnimodalSearch$; we show that the invariant is true on the $(s+1)$st call to $UnimodalSearch$.

If $j_s \leq 2$, then $i_s - k_s = 2$ and there is no $(s+1)$st recursive call. Suppose $j_s > 2$; there are four possible recursive calls in $UnimodalSearch$. We consider them in turn. For each case we must show

$$F_{j_{s+1}} < k_{s+1} - i_{s+1} \leq F_{j_{s+1}+2},$$

given that the invariant holds for i_s, j_s, and k_s.

The first recursive call is the easiest case: We have $i_{s+1} = i_s$, $j_{s+1} = j_s - 1$, and $k_{s+1} = F_{j_s} + i_s$ and so

$$F_{j_{s+1}} = F_{j_s-1} < F_{j_s} = k_{s+1} - i_{s+1} < F_{j_s+1} = F_{j_{s+1}+2},$$

as desired.

In the second recursive call, we have $i_{s+1} = i_s + F_{j_s-1}$, $j_{s+1} = j_s - 2$, and $k_{s+1} = k_s$, and hence $F_{j_{s+1}} = F_{j_s-2}$. By the invariant at step s, $k_s - i_s > F_{j_s}$ and hence

$$F_{j_s-2} < k_s - i_s - F_{j_s-1} = k_{s+1} - i_{s+1}.$$

By the conditions leading to the call, $k_s - i_s \leq F_{j_s+1}$ and hence

$$k_s - i_s - F_{j_s-1} \leq F_{j_s} = F_{j_{s+1}+2},$$

as desired.

In the third recursive call, we have $i_{s+1} = i_s + F_{j_s}$, $j_{s+1} = j_s - 1$, and $k_{s+1} = k_s$. Thus,

$$F_{j_{s+1}} = F_{j_s-1} = F_{j_s+1} - F_{j_s}.$$

By the conditions leading to the call, $k_s - i_s > F_{j_s+1}$ so

$$F_{j_{s+1}} < k_s - i_s - F_{j_s} = k_{s+1} - i_{s+1}.$$

By the invariant at step s, $k_s - i_s \leq F_{j_s+2}$ and hence

$$k_{s+1} - i_{s+1} \leq F_{j_s+1} = F_{j_{s+1}+2},$$

as desired.

Finally, in the fourth recursive call, we have $i_{s+1} = i_s$, $j_{s+1} = j_s$, and $k_{s+1} = i_s + F_{j_s+1}$. Thus,

$$F_{j_{s+1}} = F_{j_s} < F_{j_s+1} = k_{s+1} - i_{s+1} < F_{j_s+2} = F_{j_{s+1}+2},$$

as desired. □

LEMMA 2. *Let i_s and k_s be the values of i and k, respectively, for the sth recursive call to UnimodalSearch. We have*

$$k_s - i_s \leq F_{F^{-1}(k_0-i_0)-s}.$$

Proof. We prove the lemma by induction. When $s = 0$, by definition of F^{-1}, we have $k_0 - i_0 \leq F_{F^{-1}(k_0-i_0)}$, as required. Therefore, assume the lemma is true in the sth call to *UnimodalSearch*, and we show that the lemma is true on the $(s + 1)$st call to *UnimodalSearch*.

We consider each recursive call to *UnimodalSearch* as done in the proof of Lemma 1 and examine what happens to the size of the interval in each case. In the first call $k_s - i_s > F_{j_s}$, by the invariant, $i_{s+1} = i_s$, and $k_{s+1} = i_s + F_{j_s}$ so $k_{s+1} - i_{s+1} = F_{j_s}$. In the second call $k_s - i_s > F_{j_s}$ by the invariant, $k_s - i_s \leq F_{j_s+1}$ by the conditions leading to the call, $i_{s+1} = i_s + F_{j_s-1}$, and $k_{s+1} = k_s$ so $k_{s+1} - i_{s+1} = k_s - i_s - F_{j_s-1} \leq F_{j_s+1} - F_{j_s-1} = F_{j_s}$. In the third call $k_s - i_s \leq F_{j_s+2}$ by the invariant, $k_s - i_s > F_{j_s+1}$ by the conditions leading to the call, $i_{s+1} = i_s + F_{j_s}$, and $k_{s+1} = k_s$ so $k_{s+1} - i_{s+1} = k_s - i_s - F_{j_s} \leq F_{j_s+2} - F_{j_s} = F_{j_s+1}$. In the fourth call $k_s - i_s > F_{j_s+1}$, by the conditions leading to the call, $i_{s+1} = i_s$, and $k_{s+1} = i_s + F_{j_s+1}$ so $k_{s+1} - i_{s+1} = F_{j_s+1}$. In each of the four cases the interval has been narrowed by at least one Fibonacci index, proving the lemma. □

THEOREM 2. *Let i_0 and k_0 be the values of i and k, respectively, for the initial call to UnimodalSearch. When procedure UnimodalSearch is given proper initial values, it returns an interval of length two containing the maximum of unimodal function f after at most $F^{-1}(k_0 - i_0) - 3$ steps.*

Proof. Let i_s and k_s be the values of k and i, respectively, for the sth recursive call to *UnimodalSearch*. *UnimodalSearch* stops when $k_s - i_s = 2$, so that by Lemma 2 we have

$$2 \leq F_{F^{-1}(k_0-i_0)-s}.$$

Therefore

$$3 \leq F^{-1}(k_0 - i_0) - s,$$

so that

$$s \leq F^{-1}(k_0 - i_0) - 3.$$

By Lemma 1 it will return the proper interval. □

The function *FiniteUnimodalSearch*, given in Algorithm 2, uses *UnimodalSearch* to solve the finite discrete unimodal search problem over the interval (0, N). *FiniteUnimodal-Search* is a generalization of Oliver and Wilde's algorithm, behaving identically to it when N is a Fibonacci number.

COROLLARY 3. *The function FiniteUnimodalSearch uses at most $F^{-1}(N) - 2$ calls to locate the interval containing the maximum; it is therefore optimal.*

Proof. *FiniteUnimodalSearch* passes $i = 0$ and $k = N$ to *UnimodalSearch* so that by Theorem 2 there are at most $F^{-1}(N) - 3$ probes by *UnimodalSearch*. Since there is one initial probe before the call to *UnimodalSearch* in *FiniteUnimodalSearch* the total number of probes is at most $F^{-1}(N) - 2$. By Corollary 2 this is optimal. □

5. The unbounded unimodal search problem. In the unbounded discrete unimodal search problem, the search takes place over the nonnegative integers and we assume that $f(0) = \lim_{N \to \infty} f(N) = -\infty$. In this section we establish lower bounds on the growth rate of $c(i)$ as $i \to \infty$, where $c(i)$ is the number of probes used in the worst case to determine the maximum when the maximum occurs in the range $[i, i + 1)$. Then, we give a series of algorithms that approach these lower bounds.

We have the following Corollary to Theorem 1.

COROLLARY 4. *In an unbounded discrete unimodal search, if $c(i)$ is the number of probes used in the worst case to determine the maximum when the maximum occurs in the range $[i, i + 1)$, then*

$$\sum_{i=0}^{\infty} \frac{1}{F_{c(i)+2}} \leq 1. \tag{16}$$

Proof. If not, then there must be some N for which

$$\sum_{i=0}^{N-1} \frac{1}{F_{c(i)+2}} > 1,$$

contradicting Corollary 1. □

Similarly, (4) can be extended to

$$\int_0^{\infty} \varphi^{-c(x)} dx \leq \frac{2}{\varphi}. \tag{17}$$

As a direct consequence of Corollary 4, we obtain a lower bound on the cost function for the unbounded discrete unimodal search problem.

COROLLARY 5. *Let $c(i)$ be the number of probes used in the worst case by an unbounded discrete unimodal search algorithm when the maximum occurs in the range $[i, i + 1)$. If, for some nondecreasing integer-valued function d, the sum $\sum_{i=0}^{\infty} \frac{1}{F_{d(i)+2}}$ diverges, then $c(i) > d(i)$ for infinitely many i.*

The following presentation of an infinite hierarchy of algorithms for the unbounded discrete unimodal search problem is based on the methods of Reingold and Shen [13]. We need several definitions. The *Fibonacci Ackermann's function*, a slight modification of Ackermann's function [2], is defined as follows for $n \geq 0$:

$$A_i(n) = \begin{cases} F_{n+2} & i = 1, \\ A_{i-1}^{(n)}(1) & i \geq 2, \end{cases} \tag{18}$$

where $A_{i-1}^{(j)}(n) = A_{i-1}(A_{i-1}^{(j-1)}(n))$ and $A_{i-1}^{(0)}(n) = n$. Thus, $A_2(n) = fibtower(n)$, an n-high tower of Fibonacci numbers defined as

$$fibtower(n) = \begin{cases} 1 & n = 0, \\ F_{fibtower(n-1)+2} & n \geq 1. \end{cases} \tag{19}$$

Some values of $A_i(n)$ are shown in Table 1.

TABLE 1
Fibonacci Ackermann's function as defined by (18). $fibtower(n)$ *is an n-high tower of Fibonacci numbers as defined by* (19).

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$A_1(n) = F_{n+2}$	1	2	3	5	8	13	21	34	55	89	144	233	377	610
$A_2(n) = fibtower(n)$	1	2	3	5	13	610	F_{612}	$F_{F_{612}+2}$						
$A_3(n)$	1	2	3	5	610	$A_2(610)$								

The *inverse Fibonacci Ackermann's function* is defined as follows for $n \geq 1$:

$$\alpha_i(n) = \begin{cases} F^{-1}(n+1) - 3 & i = 1, \\ \text{least } j \text{ such that } \alpha_{i-1}^{(j)}(n) \leq 1 & i \geq 2, \end{cases} \tag{20}$$

where $\alpha_{i-1}^{(j)}(n) = \alpha_{i-1}(\alpha_{i-1}^{(j-1)}(n))$ and $\alpha_{i-1}^{(0)}(n) = n$. For convenience, we let $\alpha_i(0) = 0$. For $i \geq 1$, $\alpha_i(n)$ is the functional inverse of $A_i(n)$, that is,

$$\alpha_i(n) = A_i^{-1}(n) = \text{greatest } x \geq 1 \text{ such that } A_i(x) \leq n,$$

and therefore $\alpha_i(A_i(n)) = n$. See Table 2 for some values of $\alpha_i(n)$.

TABLE 2
Inverse Fibonacci Ackermann's function as defined by (20).

n	0	1	2	3	4	$5 \cdots 7$	$8 \cdots 12$	$13 \cdots 20$	609	610
$\alpha_1(n) = F^{-1}(n+1) - 3$	0	0	1	2	2	$3 \cdots 3$	$4 \cdots 4$	$5 \cdots 5$	12	13
$\alpha_2(n)$	0	0	1	2	2	$3 \cdots 3$	$3 \cdots 3$	$4 \cdots 4$	4	5
$\alpha_3(n)$	0	0	1	2	2	$3 \cdots 3$	$3 \cdots 3$	$3 \cdots 3$	3	4

The *length function* $L_i(n)$ is defined as follows for $n \geq 1$:

$$L_i(n) = \begin{cases} \alpha_1(n) = F^{-1}(n+1) - 3 & i = 1, \\ L_{i-1}(n) + L_i(\alpha_{i-1}(n)) & i \geq 2. \end{cases} \tag{21}$$

For convenience, we let $L_i(0) = 0$. Table 3 has some values of $L_i(n)$.

TABLE 3
The length function $L_i(n)$ as defined by (21).

n	0	1	2	3	4	$5\cdots 7$	$8\cdots 12$	$13\cdots 20$	609	610
$L_1(n) = F^{-1}(n+1) - 3$	0	0	1	2	2	$3\cdots 3$	$4\cdots 4$	$5\cdots 5$	12	13
$L_2(n)$	0	0	1	3	3	$6\cdots 6$	$7\cdots 7$	$11\cdots 11$	19	24
$L_3(n)$	0	0	1	4	4	$10\cdots 10$	$11\cdots 11$	$15\cdots 15$	23	34

LEMMA 3. $\sum_{n=0}^{\infty} \frac{1}{F_{L_1(n)+2}}$ *diverges.*

Proof. We can rewrite the sum as

$$\sum_{n=0}^{\infty} \frac{1}{F_{L_1(n)+2}} = 1 + 1 + \sum_{m=1}^{\infty} \sum_{n\geq 2,\, \alpha_1(n)=m} \frac{1}{F_{L_1(n)+2}}.$$

Noting that $\alpha_1(n) = L_1(n) = F^{-1}(n+1) - 3$ for $n \geq 1$, we have

$$\sum_{n\geq 2,\, \alpha_1(n)=m} \frac{1}{F_{L_1(n)+2}} = \sum_{n\geq 2,\, F^{-1}(n+1)=m+3} \frac{1}{F_{m+2}}.$$

Since there are F_{m+1} integers whose inverse Fibonacci number is $m+3$,

$$= \frac{F_{m+1}}{F_{m+2}}$$

$$\geq \frac{1}{2}, \tag{22}$$

for $m \geq 1$. Thus

$$\sum_{n=0}^{\infty} \frac{1}{F_{L_1(n)+2}} \geq 2 + \sum_{m=1}^{\infty} \frac{1}{2},$$

which diverges. □

LEMMA 4. $\sum_{n=0}^{\infty} \frac{1}{F_{L_2(n)-\alpha_2(n)+2}}$ *diverges.*

Proof. We can rewrite the sum as

$$\sum_{n=0}^{\infty} \frac{1}{F_{L_2(n)-\alpha_2(n)+2}} = 1 + 1 + \sum_{m=1}^{\infty} \sum_{n\geq 2,\, \alpha_2(n)=m} \frac{1}{F_{L_2(n)+2-m}}.$$

Noting that $\alpha_2(n) = \alpha_2(\alpha_1(n)) + 1$ and using the definition of $L_2(n)$,

$$\sum_{n\geq 2,\, \alpha_2(n)=m} \frac{1}{F_{L_2(n)+2-m}} = \sum_{n\geq 2,\, \alpha_2(\alpha_1(n))=m-1} \frac{1}{F_{L_1(n)+1+L_2(\alpha_1(n))-(m-1)}}.$$

We will show by induction that the above sum is at least 1. As a basis, if $m = 1$ then $n = 2$ and the sum is 1. By (5)

$$= \sum_{n\geq 2,\, \alpha_2(\alpha_1(n))=m-1} \frac{1}{F_{L_1(n)+1}F_{L_2(\alpha_1(n))+1-(m-1)} + F_{L_1(n)}F_{L_2(\alpha_1(n))-(m-1)}},$$

and can be rewritten as

$$= \sum_{n\geq 2,\, \alpha_2(\alpha_1(n))=m-1} \frac{1}{F_{L_1(n)+1}} \frac{1}{F_{L_2(\alpha_1(n))+2-(m-1)}} \frac{1}{\frac{F_q}{F_{q+1}} + \frac{F_r}{F_{r+1}}\frac{F_{q-1}}{F_{q+1}}},$$

where $q = F_{L_2(\alpha_1(n))+1-(m-1)}$ and $r = F_{L_1(n)}$. We have

$$\frac{F_q}{F_{q+1}} + \frac{F_r}{F_{r+1}}\frac{F_{q-1}}{F_{q+1}} = \frac{F_{r+1}F_q + F_rF_{q-1}}{F_{q+1}F_{r+1}} = \frac{F_{r+q}}{F_{q+1}F_{r+1}},$$

by (5). By a relation similar to Proposition 1 we get

$$\frac{F_{r+q}}{F_{q+1}F_{r+1}} \leq 1.$$

Thus

$$\begin{aligned}
&\sum_{n\geq 2,\, \alpha_2(n)=m} \frac{1}{F_{L_2(n)+2-m}} \\
&\geq \sum_{n\geq 2,\, \alpha_2(\alpha_1(n))=m-1} \frac{1}{F_{L_1(n)+1}}\frac{1}{F_{L_2(\alpha_1(n))+2-(m-1)}} \\
&\geq \sum_{k\geq 2,\, \alpha_2(k)=m-1} \frac{1}{F_{L_2(\alpha_1(k))+2-(m-1)}} \sum_{n\geq 2,\, \alpha_1(n)=k} \frac{1}{F_{L_1(n)+1}}.
\end{aligned}$$

By an argument similar to the one leading to (22), the inner sum is 1, so

$$\sum_{n\geq 2,\, \alpha_2(n)=m} \frac{1}{F_{L_2(n)+2-m}} \geq \sum_{k\geq 2,\, \alpha_2(k)=m-1} \frac{1}{F_{L_2(\alpha_1(k))+2-(m-1)}},$$

which is greater than 1 by the induction hypothesis.

Therefore

$$\sum_{n=0}^{\infty} \frac{1}{F_{L_2(n)-\alpha_2(n)+2}} \geq 1+1+\sum_{m=1}^{\infty} 1,$$

which diverges. □

THEOREM 3. *$L_1(n)$ and $L_2(n) - \alpha_2(n)$ are lower bounds on the unbounded discrete unimodal search problem, in the sense that if an unbounded discrete unimodal search algorithm uses $c(n)$ probes to narrow the interval to $[n, n+1, n+2]$, then $c(n) > L_1(n)$ and $c(n) > L_2(n) - \alpha_2(n)$ for infinitely many n.*

Proof. The proof follows directly from Corollary 5 and Lemmas 3 and 4. □

We would have liked to establish that for $i \geq 3$, $L_i(n) - k\alpha_i(n)$ is a lower bound on the unbounded discrete unimodal search for some constant k. Unfortunately, it can be shown that

$$\sum_{n=0}^{\infty} \frac{1}{F_{L_i(n)+2-k\alpha_i(n)}}$$

converges for $i \geq 3$ and any constant k. Therefore our methods are insufficient to show that for levels $i \geq 3$, *InfiniteUnimodalSearch* is within $k\alpha_i(n)$ of being optimal for constant k.

We now have lower bounds on the growth rate of $c(n)$ for unbounded discrete unimodal search. Upper bounds follow from the performance of the function *InfiniteUnimodal-Search* and the procedure *LevelSearch*, Algorithms 3 and 4, which solves the unbounded discrete unimodal search problem as follows. For a given unimodal function f and a given value of *level*, *InfiniteUnimodalSearch* probes successively at $A_{level}(0), A_{level}(1), \ldots$ until $f(A_{level}(p)) \geq f(A_{level}(p+1))$ and then passes the information to procedure *LevelSearch*.

ALGORITHM 3
Unbounded discrete unimodal search.

```
function $InfiniteUnimodalSearch$(
    $level$: PositiveInteger; { Level of the search }
    $f$: function(NonnegativeInteger): real { Unimodal function to be searched }
    ): NonnegativeInteger; { Left endpoint of a length-two interval containing the
                              unique maximum of $f$ }
procedure $LevelSearch$
    ⋮ { See Algorithm 4 }
end;
var
    $p$: integer;
    $CurrentValue$: real; { Value of $f(A_{level}(p+1))$ }
    $NextValue$: real; { Value of $f(A_{level}(p+2))$ }
begin { $InfiniteUnimodalSearch$ }
    $p := -1$;
    $CurrentValue := f(1)$;
    $NextValue := f(2)$;
    while $NextValue > CurrentValue$ do begin
        { invariant: $NextValue = f(A_{level}(p+2)) > CurrentValue =$
                $f(A_{level}(p+1)) > f(A_{level}(p)) > \cdots > f(A_{level}(0))$ }
        $p := p + 1$;
        $CurrentValue := NextValue$;
        $NextValue := f(A_{level}(p+2))$
        end;
    if $p = -1$ then
        $p := 0$ { Interval containing the maximum is [0, 1, 2] }
    else
        $LevelSearch(level, p, CurrentValue, f)$;
    return($p$)
end; { $InfiniteUnimodalSearch$ }
```

ALGORITHM 4

Level-by-level unbounded discrete unimodal search.

```
procedure LevelSearch(
    λ: PositiveInteger; { Level of the search }
    var m: NonnegativeInteger; { Upon entry, the left endpoint of the interval
                                 containing the maximum of $f$, relative to $f(A_\lambda)$;
                                 that is, the maximum is in the range
                                 $(A_\lambda(m), A_\lambda(m+2))$ }
    var CurrentMax: real; { Upon entry, the value of $f(A_\lambda(m+1))$, the largest
                            known value of $f$ }
    f: function(NonnegativeInteger): real); { Unimodal function to be searched at
                                              current level }
{ Upon return, m and CurrentMax are such that the maximum of $f$ is in the range
  $[m, m+1, m+2]$ and $CurrentMax = f(m+1)$ }
procedure UnimodalSearch
    ⋮ { See Algorithm 1 }
end;
var
    temp: NonnegativeInteger;
begin { LevelSearch }
    { precondition: $m \geq 0$ and
                $f(A_\lambda(m)) \leq f(A_\lambda(m+1)) = CurrentMax \geq f(A_\lambda(m+2))$ }
    if $m = 0$ then
        m := 1 { Return interval $[A_\lambda(0), A_\lambda(1), A_\lambda(2)] = [1, 2, 3]$ }
    else if $\lambda = 1$ then begin
        temp := $A_1(m)$;
        { Since $A_1(m) = F_{m+2}$, $A_1(m+2) = F_{m+4}$, and $CurrentMax = f(F_{m+3})$,
          the following call to UnimodalSearch is correct }
        UnimodalSearch(temp, $m+1$, $A_1(m+2)$, CurrentMax, f)
        { Location of the maximum has been determined by UnimodalSearch }
        m := temp
        end
    else { $m > 0$ and $\lambda > 1$ } begin
        m := m - 1;
        LevelSearch(λ, m, CurrentMax, $f(A_{\lambda-1})$);
        { As a result of the call to LevelSearch, we have
          $f(A_{\lambda-1}(m)) \leq f(A_{\lambda-1}(m+1)) = CurrentMax \geq f(A_{\lambda-1}(m+2))$ }
        LevelSearch(λ - 1, m, CurrentMax, f)
        end
    { postcondition: $m \geq 0$ and $f(m) \leq f(m+1) = CurrentMax \geq f(m+2)$ }
end; { LevelSearch }
```

In *LevelSearch*, the parameters are λ, the level of the search, f, the unimodal function being searched, *CurrentMax*, the largest known value of f, and m, the left endpoint of the interval containing the maximum of f relative to $f(A_\lambda)$, that is,

$$f(A_\lambda(m)) \le f(A_\lambda(m+1)) = \mathit{CurrentMax} \ge f(A_\lambda(m+2)).$$

Since $A_\lambda(0) = 1$, $A_\lambda(1) = 2$, and $A_\lambda(2) = 3$, if $m = 0$ then the maximum is in the range $(1, 3)$. If $m \ge 1$ and $\lambda = 1$, *LevelSearch* performs the *finite* optimal discrete unimodal search (Algorithm 1) over the interval $(A_1(m), A_1(m+2)) = (F_{m+2}, F_{m+4})$ with the maximum of all probes occurring at $A_1(m+1) = F_{m+3}$. Otherwise, if $m \ge 1$ and $\lambda > 1$, *LevelSearch* recursively performs a level λ search on the composite function $f(A_{\lambda-1})$ over the interval $(A_\lambda(m), A_\lambda(m+2))$ with the maximum of all probes occurring at $A_\lambda(m+1)$. This narrows the interval containing the maximum to $(A_{\lambda-1}(\hat{m}), A_{\lambda-1}(\hat{m}+2))$ for some $\hat{m}$ with the maximum of all probes occurring at $A_{\lambda-1}(\hat{m}+1)$. A level $\lambda - 1$ search is recursively performed over the interval $(A_{\lambda-1}(\hat{m}), A_{\lambda-1}(\hat{m}+2))$ on the function f which narrows the interval to a length-two interval containing the unique maximum of f. The recursion works because the composite function $f(A_{\lambda-1})$ is unimodal, since $A_{\lambda-1}$ is a strictly increasing function.

LEMMA 5. *In procedure LevelSearch, if the precondition is true for a given call, then the precondition and postcondition will be true for all following recursive calls and the postcondition will be true for the given call.*

Proof. We prove the lemma by induction on m and λ. If $m = 0$, there are no recursive calls and *LevelSearch* simply returns the interval $[1, 2, 3]$ with $\mathit{CurrentMax} = f(2)$, as desired. If $\lambda = 1$, there are no recursive calls and *LevelSearch* calls procedure *UnimodalSearch* which, by Theorem 2, returns interval $[\mathit{temp}, \mathit{temp}+1, \mathit{temp}+2]$ containing the maximum. The value of m is set to *temp* and $\mathit{CurrentMax} = f(m+1)$, as desired.

Let $\hat{m} > 0$ and $\hat{\lambda} > 1$ and assume the lemma is true for all m when $\lambda < \hat{\lambda}$ and for $m < \hat{m}$ when $\lambda = \hat{\lambda}$. There are two recursive calls to consider. Before the first recursive call we have

$$f(A_{\hat{\lambda}}(\hat{m})) \le f(A_{\hat{\lambda}}(\hat{m}+1)) \ge f(A_{\hat{\lambda}}(\hat{m}+2)). \tag{23}$$

Let $\tilde{m} = \hat{m} - 1$ and $\tilde{f} = f(A_{\hat{\lambda}-1})$; to show that the precondition holds for the first recursive call we need

$$\tilde{f}(A_{\hat{\lambda}}(\tilde{m})) \le \tilde{f}(A_{\hat{\lambda}}(\tilde{m}+1)) \ge \tilde{f}(A_{\hat{\lambda}}(\tilde{m}+2)). \tag{24}$$

Note the Ackermann function property that, for $i > 1$,

$$\begin{aligned} A_i(n) &= A_{i-1}^{(n)}(1) \\ &= A_{i-1}(A_{i-1}^{(n-1)}(1)) \\ &= A_{i-1}(A_i(n-1)). \end{aligned}$$

Therefore

$$\begin{aligned} f(A_{\hat{\lambda}}(\hat{m})) &= f(A_{\hat{\lambda}-1}(A_{\hat{\lambda}}(\hat{m}-1))) \\ &= \tilde{f}(A_{\hat{\lambda}}(\tilde{m})). \end{aligned}$$

Similarly,

$$f(A_{\hat{\lambda}}(\hat{m}+1)) = \tilde{f}(A_{\hat{\lambda}}(\tilde{m}+1))$$

and

$$f(A_{\hat{\lambda}}(\hat{m}+2)) = \tilde{f}(A_{\hat{\lambda}}(\tilde{m}+2)).$$

Thus (24) follows by substitution into (23). By the induction hypothesis, after the first recursive call has been completed and before the second recursive call,

$$f(A_{\hat{\lambda}-1}(\hat{m})) \le f(A_{\hat{\lambda}-1}(\hat{m}+1)) = CurrentMax \ge f(A_{\hat{\lambda}-1}(\hat{m}+2)). \tag{25}$$

Let $\tilde{\lambda} = \hat{\lambda} - 1$; to show that the precondition holds for the second recursive call we need

$$f(A_{\tilde{\lambda}}(\hat{m})) \le f(A_{\tilde{\lambda}}(\hat{m}+1)) \ge f(A_{\tilde{\lambda}}(\hat{m}+2)). \tag{26}$$

As above, we have

$$\begin{aligned} f(A_{\hat{\lambda}-1}(\hat{m})) &= f(A_{\tilde{\lambda}}(\hat{m})), \\ f(A_{\hat{\lambda}-1}(\hat{m}+1)) &= f(A_{\tilde{\lambda}}(\hat{m}+1)), \end{aligned}$$

and

$$f(A_{\hat{\lambda}-1}(\hat{m}+2)) = f(A_{\tilde{\lambda}}(\hat{m}+2)).$$

Thus (26) follows by substitution into (25). By the induction hypothesis and the postcondition of the second recursive call, the second recursive call returns the interval $[\hat{m}, \hat{m}+1, \hat{m}+2]$ with the maximum probe $CurrentMax = f(\hat{m}+1)$; this is the postcondition for the given call to $LevelSearch$. □

LEMMA 6. *For all $\lambda \ge 1$ and $n > 0$, $L_\lambda(n)$ is the number of probes (function evaluations) used in the worst case in finding the length-two interval $[n, n+1, n+2]$ containing the maximum of a unimodal function f with procedure LevelSearch, after the interval has been narrowed to $[A_\lambda(\alpha_\lambda(n)), A_\lambda(\alpha_\lambda(n)+1), A_\lambda(\alpha_\lambda(n)+2)]$ by function InfiniteUnimodalSearch.*

Proof. We prove the lemma by induction on n and λ. Procedure $LevelSearch$ is not called if $n = 0$ and therefore does no probes, satisfying the lemma for this case since $L_\lambda(0) = 0$. If $n = 1$, then $\alpha_\lambda(n) = 0$ and $LevelSearch$ does no probes satisfying the lemma for this case since $L_\lambda(1) = 0$. If $n > 1$, then $\alpha_\lambda(n) > 0$ and if $\lambda = 1$, $LevelSearch$ performs a discrete unimodal search over the interval $(A_1(\alpha_1(n)), A_1(\alpha_1(n)+2)) = (F_{\alpha_1(n)+2}, F_{\alpha_1(n)+4})$. Since this interval has length $F_{\alpha_1(n)+3}$ and the value of f at $F_{\alpha_1(n)+3}$ has already been probed, the number of probes done by $UnimodalSearch$ is, from Lemma 2, at most $\alpha_1(n) = L_1(n)$.

Assume the lemma is true for $\hat{n} < n$ and $\hat{\lambda} < l$. If $n > 1$ then $\alpha_\lambda(n) > 0$, and if $\lambda > 1$, $LevelSearch$ performs two recursive calls. Let $P_\lambda(n)$ be the number of probes by $LevelSearch$ at level λ if the maximum occurs at n. In the first recursive call, the procedure searches the function $f(A_{\lambda-1})$ over $\alpha_{\lambda-1}(n)$ values and making $P_\lambda(\alpha_{\lambda-1}(n))$ probes. In the second recursive call, the procedure searches the function f on the interval

$$[A_{\lambda-1}(\alpha(n)), A_{\lambda-1}(\alpha(n)+1), A_{\lambda-1}(\alpha(n)+2)]$$

as returned by the first call. This is the interval searched by a level $\lambda - 1$ algorithm in the worst case by induction resulting in $P_{\lambda-1}(n)$ probes for the second call. Thus, $P_\lambda(n) = P_\lambda(\alpha_{\lambda-1}(n)) + P_{\lambda-1}(n)$, precisely the definition of $L_i(n)$. □

THEOREM 4. *Using the function InfiniteUnimodalSearch, the cost of finding the interval $[n, n+1, n+2]$ containing the maximum of a unimodal function f is, for level ≥ 1,*

$$L_{level}(n) + \alpha_{level}(n) + \begin{cases} 2 & n = 0, \\ 3 & n > 0. \end{cases} \tag{27}$$

Proof. In the worst case, $InfiniteUnimodalSearch$ probes until reaching interval

$$[A_{level}(\alpha_{level}(n)), A_{level}(\alpha_{level}(n)+1), A_{level}(\alpha_{level}(n)+2)],$$

which procedure *LevelSearch* searches in $L_{level}(n)$ probes (Lemma 6). We count the number of probes leading to the call to *LevelSearch*. If $n = 0$ then the maximum is in the interval $[0, 1, 2]$, and only two probes are made satisfying the lemma since $\alpha_{level}(0) = 0$. Otherwise, if $n > 0$, function *InfiniteUnimodalSearch* increases p until

$$f(A_{level}(p+1)) \geq f(A_{level}(p+2)),$$

and then calls *LevelSearch*. In the worst case, the function probes until $n < A_{level}(p+1)$, once at each $A_{level}(i)$ for $i \leq p+2$. Thus, by the definition of $\alpha_{level}(n)$, the number of probes for $n \geq 1$ is $\alpha_{level}(n) + 3$. $\square$

By Theorems 4 and 3 the function *InfiniteUnimodalSearch* with $level = 1$ is within $\alpha_1(n)$ of being optimal and with $level = 2$ is within $2\alpha_2(n)$ of being optimal (this is similar to [12]).

6. Conclusions and open problems. We can get asymptotically better algorithms for the unbounded discrete unimodal search by repeated diagonalization. Define, for example,

$$\hat{A}_i(n) = \begin{cases} A_n(n) & i = 1, \\ \hat{A}^{(n)}_{i-1}(1) & i \geq 2, \end{cases} \tag{28}$$

for $n > 0$, where $\hat{A}^{(j)}_{i-1}(n) = \hat{A}_{i-1}(\hat{A}^{(j-1)}_{i-1}(n))$ and let $\hat{A}_i(0) = 0$ for all i. For a given level λ, make the initial probes at $\hat{A}_\lambda(0)$, $\hat{A}_\lambda(1)$, $\hat{A}_\lambda(2)$, ... and then perform a recursive search in a manner similar to that done in [14]. The diagonalization can be repeated transfinitely often. Unfortunately, although each algorithm thus constructed is better than its predecessors, we cannot show that these algorithms get significantly closer to being optimal.

We can improve *InfiniteUnimodalSearch* so that for $level \geq 3$, $c(n)$ is lower by at least $\alpha_3(n)$ than the cost given in Theorem 4. Consider the subtree in Fig. 7. After *InfiniteUnimodalSearch* has narrowed the interval containing the maximum to $[A, B, D]$, the procedure *LevelSearch* searches that interval. Notice that there is overlap between that interval and the interval $[B, D, E]$; values B to $D - 1$ occur in each. *LevelSearch* gains nothing by keeping the external nodes with values B to $D - 1$ close to the root in the subtree rooted at $[A, B, D]$ because they occur deep in the subtree rooted at $[B, D, E]$. This problem occurs throughout the tree.

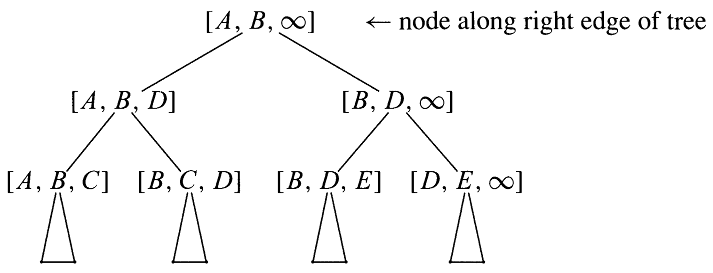


FIG. 7. *Subtree of the search tree of function InfiniteUnimodalSearch.* $A = \alpha_{level}(p)$, $B = \alpha_{level}(p+1)$, $D = \alpha_{level}(p+2)$, *and* $E = \alpha_{level}(p+3)$, *for some level and p. The value of C is determined from procedure LevelSearch.*

Suppose the next probe, C, is at $B + 1$, followed by a probe where *LevelSearch* would probe, and incorporate such changes throughout the recursive calls of the algorithm. This modification decreases the cost for values A to $B - 1$ without increasing the cost of values from B to $D - 1$, since they occur deeper in the tree elsewhere. The resulting algorithm is much more complicated than *LevelSearch*; its cost can be shown to be

$$\hat{L}_{level}(n) + \alpha_{level}(n) + \begin{cases} 2 & n = 0, \\ 3 & n > 0, \end{cases} \tag{29}$$

where $\hat{L}_i(n)$ is a new length function defined by

$$\hat{L}_i(n) = \begin{cases} \alpha_1(n) = F^{-1}(n+1) - 3 & i = 1, \\ \hat{L}_{i-1}(n) & i \geq 2 \text{ and } n \leq 4, \\ \hat{L}_{i-1}(n) + \hat{L}_i(\alpha_{i-1}(n)) & i \geq 2 \text{ and } n \geq 5. \end{cases}$$

Subtracting (29) from (27) we get

$$L_i(n) - \hat{L}_i(n) \begin{cases} = 0 & n \leq 2, \\ = 0 & i = 1 \text{ and } n \geq 3, \\ = 1 & i = 2 \text{ and } n \geq 3, \\ = \alpha_3(n) & i = 3 \text{ and } n \geq 3, \\ > \alpha_3(n) & i \geq 4 \text{ and } n \geq 3, \end{cases}$$

so that *InfiniteUnimodalSearch* is improved by $\alpha_3(n)$ for $level \geq 3$. Thus no level of *InfiniteUnimodalSearch* is within $\alpha_4(n)$ of being optimal, but it is open whether or not *InfiniteUnimodalSearch* is within $\alpha_3(n)$ of being optimal for $level \geq 3$. We do not know how close to optimal the modified *InfiniteUnimodalSearch* is, though we believe it is not within $\alpha_4(n)$ of being optimal for any level.

We can also ask about a converse to Corollary 1. Knuth [8] gives a converse to Kraft's inequality: If $c(1)$, $c(2)$, $c(3)$, ... is a nondecreasing sequence of positive integers such that $\sum_{n=1}^{\infty} 2^{-c(n)} = 1$, then there is an unbounded search algorithm in which the number of probes made is $c(n)$ if the zero occurs in the interval $[n, n+1)$. Similarly, we can ask if $c(1)$, $c(2)$, $c(3)$, ... is a nondecreasing sequence of positive integers such that $\sum_{n=1}^{\infty} \frac{1}{F_{c(n)+2}} \leq 1$, under what conditions can one construct a unbounded discrete unimodal search algorithm in which the number of probes made is $c(n)$ if the maximum occurs in the interval $[n, n+1)$?

Finally, can one analyze $\bar{c}(i)$, the average cost in a unimodal search tree, where $\bar{c}(i)$ is the average path length from the root to a node of the form $[i-1, i, i+1]$ or $[i, i+1, i+2]$?

Note added in proof. Anmol Mathur and the second author [10] have generalized the results of this paper to k-modal searching.

Acknowledgments. We thank Xiaojun Shen for extensive preliminary discussions of ideas relating to this material.

REFERENCES

[1] N. ABRAMSON, *Information Theory and Coding*, McGraw-Hill, New York, 1963.
[2] W. ACKERMANN, *Zum Hilbertschen Aufbau der reellen Zahlen*, Math. Ann., 99 (1928), pp. 118–133.
[3] M. AVRIEL AND D. J. WILDE, *Optimality proof for the symmetric Fibonacci search technique*, Fib. Quart., 4 (1966), pp. 265–269.

[4] J. L. Bentley and A. C.-C. Yao, *An almost optimal algorithm for unbounded searching*, Inform. Process. Lett., 5 (1976), pp. 82–87.

[5] J. Kiefer, *Sequential minimax search for a maximum*, Proc. Amer. Math. Soc., 4 (1955), pp. 502–505.

[6] D. E. Knuth, *The Art of Computer Programming, Volume* I: *Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[7] ———, *The Art of Computer Programming, Volume* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[8] ———, *Supernatural numbers*, in The Mathematical Gardner, D. A. Klarner, ed., Wadsworth International, Belmont, CA, 1981, pp. 310–325.

[9] L. G. Kraft, *A device for quantizing, grouping and coding amplitude modified pulses*, Master's thesis, Electrical Engineering Department, Massachusetts Institute of Technology, Cambridge, MA, 1949.

[10] A. Mathur and E. M. Reingold, *Generalized Kraft's inequality and discrete k-modal search*, Report Number UIUCDCS-R-93-1798, Department of Computer Science, University of Illinois, Urbana, IL, March, 1993.

[11] L. T. Oliver and D. J. Wilde, *Symmetrical sequential minimax search for a maximum*, Fib. Quart., 2 (1964), pp. 169–175.

[12] J.-C. Raoult and J. Vuillemin, *Optimal unbounded search strategies*, Rapport de Recherche 33, Laboratoire de Recherche en Informatique, Université de Paris-Sud, Orsay, France, 1979.

[13] E. M. Reingold and X. Shen, *More nearly optimal algorithms for unbounded searching, Part* I: *The finite case*, SIAM J. Comput., 20 (1991), pp. 156–183.

[14] ———, *More nearly optimal algorithms for unbounded searching, Part* II: *The transfinite case*, SIAM J. Comput., 20 (1991), pp. 184–208.

[15] C. Witzgall, *Fibonacci search with arbitrary first evaluation*, Fib. Quart. 10, (1972), pp. 113–134; 146.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 778–793, August 1993

COUNTING CIRCULAR ARC INTERSECTIONS*

PANKAJ K. AGARWAL,† MARCO PELLEGRINI,‡ AND MICHA SHARIR§

Abstract. In this paper efficient algorithms for counting intersections in a collection of circles or circular arcs are presented. An algorithm for counting intersections in a collection of n circles is presented whose running time is $O(n^{3/2+\epsilon})$, for any $\epsilon > 0$ is presented. Using this algorithm as a subroutine, it is shown that the intersections in a set of n circular arcs can also be counted in time $O(n^{3/2+\epsilon})$. If all arcs have the same radius, the running time can be improved to $O(n^{4/3+\epsilon})$, for any $\epsilon > 0$.

Key words. arrangements, point location, random sampling, partition tree

AMS subject classifications. 68P05, 68Q25, 68Q40, 68U05

1. Introduction. Intersection problems are among the fundamental topics in computational geometry. In 1979, Bentley and Ottmann, in their famous paper on the line sweep technique, showed that all K intersections in a collection of n Jordan arcs in $\mathbb{R}^2$ can be reported in time $O((n + K) \log n)$ [BO] (under reasonable assumptions concerning the shape of these arcs and on the model of computation). Their algorithm is significantly faster than the naive quadratic algorithm for small values of K, but is worse than the naive approach if $K = \Theta(n^2)$. Since then much effort has been invested to remove the $\log n$ factor from K. Recently Chazelle and Edelsbrunner [CE] presented an $O(n \log n + K)$ time algorithm to report all intersections in a collection of line segments. However, for general arcs, the Bentley–Ottmann algorithm is still the best known deterministic algorithm. If we allow randomization, algorithms with improved running time of $O(n \log n + K)$ can be obtained (see Clarkson and Shor [CS2] and Mulmuley [Mu]).

All of the above algorithms are quite efficient if we want to *report* the intersections explicitly. In some applications, however, we are only interested in *counting* the total number of intersections (not in finding the actual intersection points). In that case we prefer an algorithm whose running time does not depend on K, because K can be as large as $\Theta(n^2)$. Ideally we would like to have an algorithm that counts the number of intersections in time $O(n \log n)$, but developing such an algorithm seems to be quite hard. For line segments, Chazelle [Cha] proposed an $O(n^{3/2} \log n)$ time algorithm. Later Guibas, Overmars, and Sharir [GOS] gave a randomized algorithm whose expected running time was $O(n^{4/3+\epsilon})$, for any $\epsilon > 0$. Their algorithm can be made deterministic and somewhat improved using recent partitioning algorithms of Matoušek [Mat1] and Agarwal [Ag1]. The best known running time, at present, for counting segment intersections is $O(n^{4/3} \log^{1/3} n)$ (see [Ag2], [Chb]), and the corresponding algorithms are deterministic.

The "bichromatic" version of the segment intersection counting problem has also been studied. Here we are given a set of "red" segments and another set of "blue" segments, and we want to count the number of red-blue intersections. This problem can also be solved in time

*Received by the editors January 28, 1991; accepted for publication (in revised form) May 20, 1992. Part of this work was done while the first author was visiting the Center for Discrete Machematics and Theoretical Computer Science (DIMACS), Rutgers University. Work by the first author has been supported by DIMACS, a National Science Foundation Science and Technology Center, under grant NSF-STC88-09648. Work by the third author has been supported by Office of Naval Research grant N00014–90–J–1284, National Science Foundation grant CCR-89-01484, by grants from the U.S.–Israeli Binational Science Foundation, the Fund for Basic Research administered by the Israeli Academy of Sciences, and the German–Israeli Foundation for Scientific Research and Development.

†Computer Science Department, Duke University, Durham, North Carolina 27706.

‡Computer Science Department, King's College, London, United Kingdom.

§School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel and Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012.

$O(n^{4/3} \log^{1/3} n)$ using a variant of the Guibas, Overmars, and Sharir algorithm (see [Ag2]). For the special case, where no two red segments and no two blue segments intersect, an optimal $O(n \log n)$ algorithm has been developed by Chazelle et al. [CEGS2].

The intersection counting problem seems to be much harder for general arcs, because unlike segments, two arcs may intersect at more than one point, which makes their intersection patterns more involved than those of segments. Using Chazelle and Sharir's generalized point location technique in semi-algebraic varieties [CS1], the arc intersection problem can be solved in slightly subquadratic time (roughly $O(n^{1.98})$). However, we are not aware of any substantially subquadratic algorithm for the general case. The recent algorithm of Agarwal et al. [AASS] can be applied to obtain an $O(n^{4/3} \log^{2/3} n)$ time algorithm for counting intersections in a set of circles with the same radius, but it does not extend to circles of arbitrary radii.

In this paper we consider the intersection-counting problem for a collection of arbitrary circles and of circular arcs. Our main results are:

(i) An algorithm for counting intersections in a collection of n arbitrary circles whose running time is $O(n^{3/2+\epsilon})$, for any $\epsilon > 0$.[1]

(ii) An algorithm for counting intersections in a collection of n circular arcs, whose running time is $O(n^{3/2+\epsilon})$.

(iii) If all arcs are of circles of the same radius, the running time of both algorithms can be improved to $O(n^{4/3+\epsilon})$.

The remainder of the paper is organized as follows. In §2 we present our circle intersection algorithm. Section 3 deals with counting intersections between a set of circles and a set of circular arcs. In §4 we describe our main algorithm, that is, counting intersections in a set of circular arcs. We conclude with some open problems in §5.

2. Counting circle intersections.

In this section we present a divide-and-conquer algorithm for the following problem: "Given a collection $\mathcal{C} = \{C_1, \ldots, C_n\}$ of n circles, count the number of pairs of intersecting circles in $\mathcal{C}$." Let $\mathcal{C}_1 = \{C_1, \ldots, C_{\lceil n/2 \rceil}\}$ and $\mathcal{C}_2 = \{C_{\lceil n/2 \rceil+1}, \ldots, C_n\}$. We first count the number of pairs of intersecting circles in $\mathcal{C}_1$ and in $\mathcal{C}_2$ recursively, and then we count the number of intersecting pairs $(C_i, C_j) \in \mathcal{C}_1 \times \mathcal{C}_2$. Thus, it is sufficient to solve the "bichromatic version" of the above problem, that is,

Given a collection $\mathcal{C}$ of m "red" circles and another collection $\mathcal{C}'$ of n "blue" circles in the plane, count $\mathcal{I}(\mathcal{C}, \mathcal{C}')$, the number of intersecting "red-blue" pairs of circles.

Note that, assuming nondegenerate configurations, the number of red-blue intersection points is twice that count.

LEMMA 2.1. *Given a pair of circles C, C' with centers p, p' and radii ρ, ρ', respectively, C intersects C' if and only if*

$$(1) \qquad |\rho - \rho'| \le d(p, p') \le \rho + \rho'.$$

Proof. The proof follows from elementary geometry. □

To compute $\mathcal{I}(\mathcal{C}, \mathcal{C}')$, we define two geometric transforms φ and Ψ. Let C be a circle with center (a, b) and radius r. The first transform, φ, maps C to the point

$$(2) \qquad \varphi(C) = (a, b, r)$$

[1]Throughout this paper, ϵ denotes an arbitrarily small positive constant. The meaning of such a bound is that for any $\epsilon > 0$ the algorithm can be fine-tuned so that its running time is within that bound, where the constant of proportionality depends on ϵ and usually tends to ∞ as $\epsilon \downarrow 0$.

in $\mathbb{R}^3$, and the second transform, Ψ, maps C to the region

$$\Psi(C) = \{ (x, y, z) \mid z \geq 0 \text{ and } (z + r)^2 \geq (x - a)^2 + (y - b)^2 \geq (z - r)^2 \} \tag{3}$$

in $\mathbb{R}^3$. The boundary of $\Psi(C)$ consists of two surfaces:

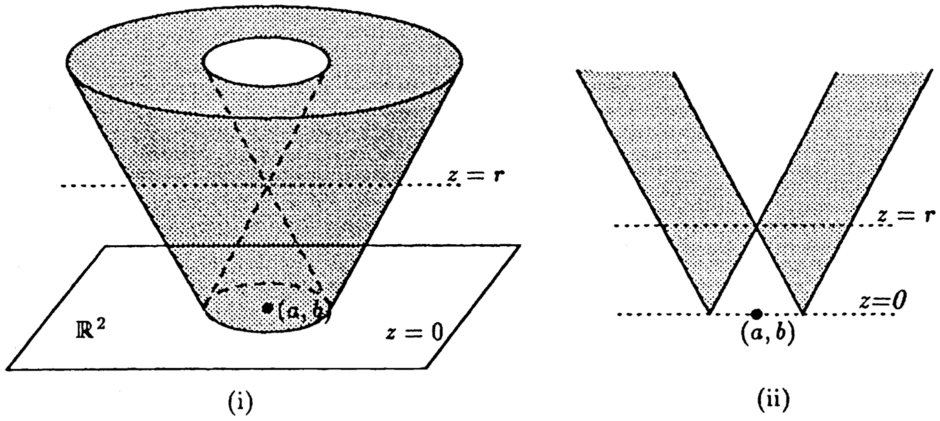


FIG. 1. (i) *Shaded region denotes* $\Psi(C)$; (ii) *cross-section of* $\Psi(C)$ *at a vertical plane through* (a, b).

(i) *The outer surface*, denoted $\psi_O(C)$, is the truncated cone

$$(z + r)^2 = (x - a)^2 + (y - b)^2, \qquad z \geq 0.$$

(ii) *The inner surface*, denoted $\psi_I(C)$, is the truncated double cone

$$(z - r)^2 = (x - a)^2 + (y - b)^2, \qquad z \geq 0.$$

A point $p = (a, b)$ can be considered a circle of radius zero, so $\psi_I(p) = \psi_O(p)$ is the cone $z^2 = (x - a)^2 + (y - b)^2$. An immediate consequence of the previous lemma is the following lemma.

LEMMA 2.2. *A circle C intersects another circle C' if and only if $\varphi(C) \in \Psi(C')$.*

It therefore suffices to count for each circle $C \in \mathcal{C}$ the number of circles $C' \in \mathcal{C}'$ for which $\varphi(C) \in \Psi(C')$. We describe two algorithms for counting these quantities. The first algorithm is quite simple and works well when $m \geq n^3$, while the second algorithm, though more complicated, is efficient for all ranges of m and n.

2.1. A simple algorithm. Our first algorithm for computing $\mathcal{I}(\mathcal{C}, \mathcal{C}')$ works as follows. Let

$$\mathcal{H} = \{ \psi_O(C') \mid C' \in \mathcal{C}' \} \cup \{ \psi_I(C') \mid C' \in \mathcal{C}' \}. \tag{4}$$

Fix a surface $H \in \mathcal{H}$. Intersect it with all other surfaces of $\mathcal{H}$. It is easily checked that each intersection curve is the intersection of H with a plane, and is thus a conic planar curve. Moreover, each pair of such curves intersect each other in at most two points (these points are the points of intersection of H with the line which is the intersection of the two corresponding planes; since H is a quadric, there are at most two such points).

Let Γ denote the resulting collection of curves on H. We next form the two-dimensional arrangement $\mathcal{A}(\Gamma)$ of these curves on H. This can be done deterministically, e.g., in time $O(n^2 \log n)$ using a sweeping technique, similar to that of Bentley and Ottmann [BO]. For each circle $C' \in \mathcal{C}'$ and each face $f \in \mathcal{A}(\Gamma)$, f lies either fully inside or fully outside $\Psi(C')$. Consequently, after $\mathcal{A}(\Gamma)$ is computed, we can calculate, for each face f of this arrangement, the number τ_f of regions $\Psi(C')$ that contain it. Since this number changes only by one as we cross from one face to an adjacent one, a simple traversal of the arrangement will produce all the quantities τ_f in time $O(n^2)$. We repeat this step for each surface $H \in \mathcal{H}$ in overall time $O(n^3 \log n)$.

Next, preprocess $\mathcal{A}(\mathcal{H})$ for spatial point location in time $O(n^{3+\epsilon})$ using the algorithm of Chazelle et al. [CEGS1].[2] The processing is done so that the output to a query is the surface that lies directly below it. The time to answer a query is $O(\log n)$. Note that the method of [CEGS1] does not require an explicit construction of the arrangement of $\mathcal{H}$, especially if the output to queries is to have this restricted form.

We now take each circle $C \in \mathcal{C}$ and locate its image $\varphi(C)$ in $\mathcal{A}(\mathcal{H})$, obtaining the surface H lying directly below $\varphi(C)$ and the vertical projection $\omega(C)$ of $\varphi(C)$ on that surface. We now locate the face f of the corresponding arrangement on H that contains $\omega(C)$, and add either τ_f or $\tau_f + 1$ to a global count, depending on whether the region $\Psi(C')$ bounded by H lies below H or above it near the face f. The sum of these quantities, over all $C \in \mathcal{C}$, is the desired $\mathcal{I}(\mathcal{C}, \mathcal{C}')$. The time required by this step is $O(m \log n)$, so the total running time of the algorithm is $O(n^{3+\epsilon} + m \log n)$.

The running time can be improved using a standard "batching" technique; that is, partition $\mathcal{C}'$ into $t = \lceil \frac{n}{m^{1/3}} \rceil$ sets $\mathcal{C}'_1, \ldots, \mathcal{C}'_t$, each containing at most $m^{1/3}$ circles. For each i, we compute $\mathcal{I}(\mathcal{C}, \mathcal{C}'_i)$ separately, and then add these quantities to determine $\mathcal{I}(\mathcal{C}, \mathcal{C}')$. The overall running time is therefore

$$\left\lceil \frac{n}{m^{1/3}} \right\rceil \times O(m^{1+\epsilon}) = O(m^{2/3+\epsilon} n + m^{1+\epsilon}).$$

Hence, we obtain the following result.

THEOREM 2.3. *Given a collection $\mathcal{C}$ of m "red" circles and a collection $\mathcal{C}'$ of n "blue" circles, the number of intersecting red-blue pairs can be counted in time $O(m^{2/3+\epsilon} n + m^{1+\epsilon})$, for any $\epsilon > 0$.*

Remark 2.4.

1. By flipping the role of red and blue circles, we can obtain another algorithm whose running time is $O(mn^{2/3+\epsilon} + n^{1+\epsilon})$.

2. Note that if $n < m^{1/3}$ (or $m < n^{1/3}$), then the above algorithm runs in time $O(m^{1+\epsilon})$ (respectively, $O(n^{1+\epsilon})$), which is almost optimal.

COROLLARY 2.5. *Given a collection of n circles, the number of intersecting pairs of circles can be counted in time $O(n^{5/3+\epsilon})$, for any $\epsilon > 0$.*

2.2. An improved algorithm. We now present an algorithm for computing $\mathcal{I}(\mathcal{C}, \mathcal{C}')$, which is significantly faster than the previous one, especially when m and n are of the same order of magnitude.

If $n \geq m^3$, $\mathcal{I}(\mathcal{C}, \mathcal{C}')$ is computed using the previous algorithm, in time $O(n^{1+\epsilon})$, so assume that $n < m^3$. Let $\mathcal{H}$ be the set of surfaces defined in (4). Let r be some sufficiently large constant. We compute a $\frac{1}{2r}$-net $\mathcal{R} \subseteq \Gamma$, of size $O(r \log r)$, in time $O(n)$ [Mat2].[3] We decompose $\mathcal{A}(\mathcal{R})$ into $M = O(r^3 \beta(r) \log^3 r)$ simple cells (of constant complexity), using the algorithm of Chazelle et al. [CEGS1] (see also [CEGSW, §6]); $\beta(r)$ is an extremely slow

[2]Although the algorithm described in the original paper is randomized, it can be made deterministic without affecting its asymptotic performance, using a recent result of Matousek for deterministic construction of ϵ-nets [Mat2].

[3]Specializing from the general concept, we call a subset $\mathcal{R} \subseteq \Gamma$ of a set of n (algebraic) surfaces a $\frac{1}{r}$-net, $r < n$, if every (open) cell of constant complexity, of the form obtained in the stratification algorithm of [CEGS1], which does not intersect any surface of $\mathcal{R}$, intersects at most n/r surfaces of Γ; see [HW] for a more formal definition. Haussler and Welzl [HW] showed that a random subset of Γ of size $O(r \log r)$ is a $\frac{1}{r}$-net with high probability. Later Matousek [Mat2] gave an $O(nr^{O(1)})$-time deterministic algorithm for computing a $\frac{1}{r}$-net of size $O(r \log r)$.

growing function, which depends on the inverse Ackermann function $\alpha(r)$. Let $\mathcal{A}^*(\mathcal{R})$ denote the resulting subdivision. Since $\mathcal{A}^*(\mathcal{R})$ is a $\frac{1}{2r}$-net of $\mathcal{H}$, each cell $\tau \in \mathcal{A}^*(\mathcal{R})$ of dimension ≥ 1 intersects at most $2n/2r = n/r$ surfaces of $\mathcal{H}$.

Our approach is to partition the points $\varphi(C)$ among the cells $\tau \in \mathcal{A}^*(\mathcal{R})$ and to distribute the surfaces ψ_I, ψ_O among those cells that they intersect. This gives us a collection of subproblems, one for each cell τ, whose combined solutions give us the desired count.

In more detail, we regard $\mathcal{A}^*(\mathcal{R})$ as a collection of pairwise disjoint relatively open cells of dimensions 3, 2, 1, or 0. For each cell $\tau \in \mathcal{A}^*(\mathcal{R})$, we define $\mathcal{C}'_\tau$ as the set of circles C' for which $\psi_I(C')$ or $\psi_O(C')$ intersects τ. Similarly, we define $\mathcal{C}_\tau$ as the set of circles C for which $\varphi(C)$ lies in τ. Let $m_\tau = |\mathcal{C}_\tau|$, $n_\tau = |\mathcal{C}'_\tau|$, and let λ_τ be the number of circles C' for which $\tau \subseteq \Psi(C')$. Every τ of dimension ≥ 1 intersects at most n/r surfaces of $\mathcal{H}$, thus $n_\tau \leq \frac{n}{r}$. It is easy to see that

$$\mathcal{I}(\mathcal{C}, \mathcal{C}') = \sum_{\tau \in \mathcal{A}^*(\mathcal{R})} \mathcal{I}(\mathcal{C}_\tau, \mathcal{C}'_\tau) + m_\tau \lambda_\tau .$$

Since each cell τ has constant complexity, $\mathcal{C}_\tau$, $\mathcal{C}'_\tau$ and λ_τ can be calculated in $O(m+n)$ time per cell, so the total time required to compute these quantities is $O((m+n)r^3\beta(r)\log^3 r) = O(m+n)$, as r is constant. Next, the quantities $\mathcal{I}(\mathcal{C}_\tau, \mathcal{C}'_\tau)$ are computed as follows. If τ is a three-dimensional cell, we compute $\mathcal{I}(\mathcal{C}_\tau, \mathcal{C}'_\tau)$ recursively. If τ is a two-dimensional cell, we also proceed recursively, except that we have a two-dimensional problem at hand. This can still be solved using an analogous two-dimensional partitioning scheme. If τ is an edge of $\mathcal{A}^*(\mathcal{R})$, we can compute $\mathcal{I}(\mathcal{C}_\tau, \mathcal{C}'_\tau)$, in time $O((m_\tau + n_\tau)\log n_\tau)$, by computing the intersection points of τ and $\partial\Psi(C')$, for $C' \in \mathcal{C}'_\tau$, sorting them along τ, and then locating the points $\varphi(C)$ along τ, for $C \in \mathcal{C}_\tau$. Finally, if τ is a vertex, then we have $\mathcal{I}(\mathcal{C}_\tau, \mathcal{C}'_\tau) = m_\tau \cdot n_\tau$ (note that in this case $m_\tau = 0$ or 1). The maximum total running time $T(m,n)$ to compute $\mathcal{I}(\mathcal{C}, \mathcal{C}')$ therefore satisfies:

$$(5) \qquad T(m,n) \leq \begin{cases} a_1(m+n)\log n + \displaystyle\sum_{\substack{\tau \in \mathcal{A}^*(\mathcal{R}) \\ \dim \tau \geq 2}} T(m_\tau, n_\tau) & \text{if} \quad n < m^3 \\ a_2 n^{1+\epsilon'} & \text{if} \quad n \geq m^3 , \end{cases}$$

where a_1, a_2 are some constants (a_2 depending on ϵ'), $\sum_\tau m_\tau \leq m$, and $n_\tau \leq \frac{n}{r}$ for every τ. It is easily checked that these equations also hold when we recurse on a two-dimensional problem. In this case the second equation holds already when $n \geq m^2$.

LEMMA 2.6. *For every $\epsilon > 0$, there exist constants A, B, $D > 0$ depending on ϵ such that*

$$(6) \qquad T(m,n) \leq Am^{3/4+\epsilon}n^{3/4} + Bn^{1+\epsilon/3} + Dm\log^2 n.$$

Proof. For $n \geq m^3$, (6) is obviously true provided we choose $B \geq a_2$ and $\epsilon \geq 3\epsilon'$. For $n < m^3$, we prove the lemma by induction on n. If A and B are chosen appropriately, (6) holds trivially for small values of n. Assume that the claim is true for all $n' < n$. If $r > 2$, then by inductive hypothesis, (5) can be written as follows (here we bound the number of cells in $\mathcal{A}^*(\mathcal{R})$ by $c_1 r^3\beta(r)\log^3 r$ for an appropriate constant $c_1 > 0$):

$$T(m,n) \leq \sum_{\substack{\tau \in \mathcal{A}^*(\mathcal{R}) \\ \dim \tau \geq 2}} \left(Am_\tau^{3/4+\epsilon}n_\tau^{3/4} + Bn_\tau^{1+\epsilon/3} + Dm_\tau \log^2 n_\tau \right) + a_1(m+n)\log n$$

$$\begin{aligned}
&\leq A\left(\frac{n}{r}\right)^{3/4}\cdot\left(\sum_{\substack{\tau\in\mathcal{A}^*(\mathcal{R})\\ \dim\tau\geq 2}} m_\tau^{3/4+\epsilon}\right)+Bc_1r^3\beta(r)\log^3 r\left(\frac{n}{r}\right)^{1+\epsilon/3}\\
&\quad+a_1(m+n)\log n+D\log n\log\frac{n}{r}\cdot\sum_{\substack{\tau\in\mathcal{A}^*(\mathcal{R})\\ \dim\tau\geq 2}} m_\tau\\
&\leq Am^{3/4+\epsilon}n^{3/4}\left(\frac{1}{r}\right)^{3/4}\left(c_1r^3\beta(r)\log^3 r\right)^{1/4-\epsilon}+Bn^{1+\epsilon/3}+Dm\log^2 n\\
&\quad+Bn^{1+\epsilon/3}\left(c_1r^{2-\epsilon/3}\beta(r)\log^3 r-1+\frac{a_1}{B}\frac{\log n}{n^{\epsilon/3}}\right)+(a_1-D\log r)m\log n
\end{aligned}$$

where the first term of the last inequality follows from Hölder's inequality. If $D > a_1$, then $a_1 - D\log r < 0$ (as $r > 2$). Since $n < m^3$, we have $n^{1+\epsilon/3} \leq m^{3/4+\epsilon}n^{3/4}$. We therefore obtain

$$\begin{aligned}
T(m,n) &\leq Am^{3/4+\epsilon}n^{3/4}\left(\frac{(c_1\beta(r)\log^3 r)^{1/4-\epsilon}}{r^{3\epsilon}}\right)+Bn^{1+\epsilon/3}+Dm\log^2 n\\
&\quad+Em^{3/4+\epsilon}n^{3/4},
\end{aligned}$$

where

$$E = B\left(c_1r^{2-\epsilon/3}\beta(r)\log^3 r-1+\frac{a_1}{B}\frac{\log n}{n^{\epsilon}}\right).$$

If we choose r and A sufficiently large so that

$$\frac{(c_1\beta(r)\log^3 r)^{1/4-\epsilon}}{r^{3\epsilon}}+\frac{E}{A}\leq 1,$$

the running time becomes

$$T(m,n)\leq Am^{3/4+\epsilon}n^{3/4}+Bn^{1+\epsilon/3}+Dm\log^2 n. \qquad \square$$

THEOREM 2.7. *Given a collection C of m "red" circles and a collection C' of n "blue" circles, the number of intersecting red-blue pairs of circles can be counted in time $O(m^{3/4+\epsilon}n^{3/4}+n^{1+\epsilon}+m\log^2 n)$, for any $\epsilon > 0$.*

Returning to the original problem of counting circle intersections, the above theorem implies that the merge step of our divide-and-conquer algorithm can be performed in time $O(n^{3/2+\epsilon})$. Hence, we obtain the main result of this section.

THEOREM 2.8. *Given a collection of n circles in the plane, the number of intersecting pairs can be counted in time $O(n^{3/2+\epsilon})$, for any $\epsilon > 0$.*

3. Counting intersections between circles and arcs. In this section, we study the problem of counting the number of intersection points between a collection C of n circles and another collection Γ of m circular arcs. For the sake of simplicity we assume that all intersections are transversal and do not lie at the endpoints of the arcs.

Let α, β denote the endpoints of an arc γ, and let c denote the center of the circle containing γ. The lines supporting the segments αc and βc partition the plane into four wedges (quadrants), and γ clearly lies completely in one of the wedges; we will denote this wedge by $\omega(\gamma)$, and denote by $\bar{\omega}(\gamma)$ the wedge lying opposite to $\omega(\gamma)$; see Fig. 3 for an

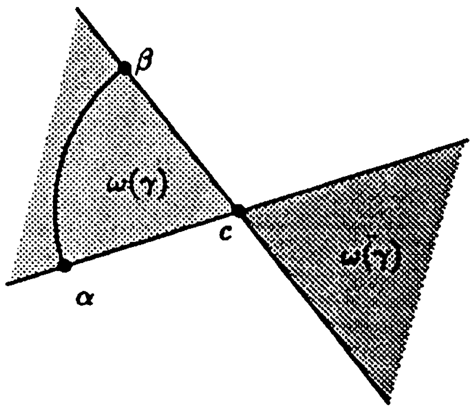


FIG. 2. *Wedges $\omega(\gamma)$ and $\bar{\omega}(\gamma)$.*

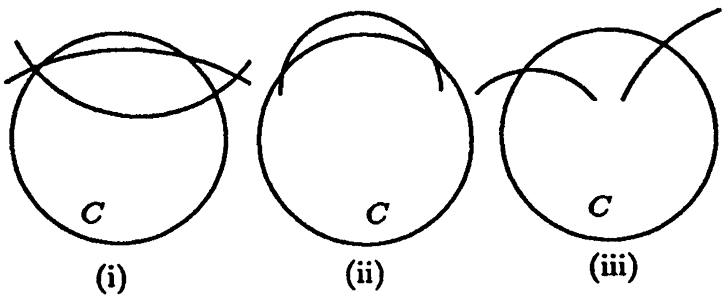


FIG. 3. *Various circle-arc intersection patterns.*

example. For a set $\mathcal{C}$ of circles, let $E(\mathcal{C})$ (respectively, $I(\mathcal{C})$) denote the common exterior (respectively, interior) of circles in $\mathcal{C}$.

LEMMA 3.1. *A circular arc γ intersects a circle C at one point if and only if one of the endpoints of γ lies in the interior of C and the other endpoint lies in the exterior of C.*

Proof. The proof follows immediately from the Jordan curve theorem and the fact that two circles intersect in at most two points. □

LEMMA 3.2. *A circular arc γ, which does not span more than a semicircle, intersects C at two points if and only if C intersects the circle containing γ, and one of the following two conditions is satisfied:*

(i) *the endpoints of γ lie outside C and the center of C lies in $\omega(\gamma)$ (see Fig.* 3(i)) *or*

(ii) *the endpoints of γ lie inside C and the center of C lies in $\bar{\omega}(\gamma)$ (see Fig.* 3(ii)).

Proof. Assume that γ and C intersect at two points, say z_1, z_2. Let ℓ be the perpendicular bisector of z_1z_2. Let C^* be the circle containing γ, and let c, r (respectively, c^*, r^*) denote the center and radius of C (respectively, C^*). Clearly, both c and c^* lie on ℓ. Since $z_1, z_2 \in \omega(\gamma)$, we can easily show that ℓ is fully contained in $\omega(\gamma) \cup \bar{\omega}(\gamma)$, so either $c \in \omega(\gamma)$ or $c \in \bar{\omega}(\gamma)$. See Fig. 4 an illustration.

If we fix z_1, z_2, and let c slide along ℓ, we obtain a one-parameter family of circles C all passing through z_1 and z_2. The line through z_1 and z_2 divides the plane into two halfplanes; we denote by H_1 the halfplane containing c^* and by H_2 the complementary halfplane. It is

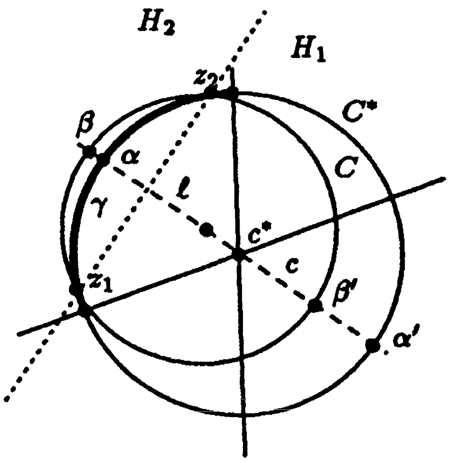


FIG. 4. *Illustration of Lemma* 3.2.

easily checked that, as c moves along ℓ towards $\omega(\gamma)$, $C \cap H_1$ keeps shrinking while $C \cap H_2$ keeps expanding. Since the endpoints of γ lie in H_1, and are incident to C when $c = c^*$, they clearly lie outside C if and only if $c \in \omega(\gamma)$ and inside C if and only if $c \in \bar{\omega}(\gamma)$. This proves the "only if" part of the lemma.

The "if" part is proved in a similar manner. We let z_1, z_2 denote the two points of intersection of C and C^*, and define ℓ, H_1, and H_2 as above. Any of the conditions (i), (ii) implies that ℓ is contained in $\omega(\gamma) \cup \bar{\omega}(\gamma)$. It then suffices to show that both endpoints of γ lie in H_1. If condition (i) occurs then, by the observation made above, we have $C \cap H_2 \supseteq C^* \cap H_2$, so both endpoints of γ must lie in H_1. A similar argument applies for condition (ii). This completes the proof of the lemma. □

In view of Lemmas 3.1 and 3.2, we can divide the pairs of intersecting arcs and circles, $(\gamma, C) \in \Gamma \times \mathcal{C}$, into the following three categories:

I. Both endpoints of γ lie in the exterior of C, the center of C lies in $\omega(\gamma)$, and C intersects the circle containing γ; let $\mathcal{I}_1(\Gamma, \mathcal{C})$ denote the number of such pairs.
II. Both endpoints of γ lie in the interior of C, the center of C lies in $\bar{\omega}(\gamma)$, and C intersects the circle containing γ; let $\mathcal{I}_2(\Gamma, \mathcal{C})$ denote the number of such pairs.
III. Exactly one of the endpoints of γ lies in the exterior of C; let $\mathcal{I}_3(\Gamma, \mathcal{C})$ denote the number of such pairs.

Although all three types of intersecting pairs can be counted by a single algorithm, we prefer to count each of them by a separate procedure, for the sake of clarity.

3.1. Counting $\mathcal{I}_1(\Gamma, \mathcal{C})$. We will view condition I as a conjunction of five constraints, four of which constrain the locations of endpoints of arcs and the locations of centers of circles, and the fifth one requires the two circles to intersect. We will construct a four-level structure, based on the decomposition scheme of Chazelle, Sharir, and Welzl [CSW], which decomposes Γ and $\mathcal{C}$ into a family $\mathcal{F}$ of canonical pairs

$$\mathcal{F} = \{(\Gamma_1, \mathcal{C}_1), \ldots, (\Gamma_k, \mathcal{C}_k)\},$$

such that

(i) $\Gamma_i \subseteq \Gamma$ and $\mathcal{C}_i \subseteq \mathcal{C}$,
(ii) for every pair $(\gamma, C) \in (\Gamma_i, \mathcal{C}_i)$, the endpoints of γ lie in the exterior of C and the center of C lies in $\omega(\gamma)$, and

(iii) for every pair $(\gamma, C) \in \Gamma \times \mathcal{C}$ such that the endpoints of γ lie in the exterior of C and the center of C lies in $\omega(\gamma)$, there is a unique i such that $\gamma \in \Gamma_i$ and $C \in \mathcal{C}_i$.

Each of the four constraints will be satisfied at a different level of the structure. The first-level structure decomposes Γ and $\mathcal{C}$ into a family of canonical pairs $(\Gamma_\triangle, \mathcal{C}_\triangle)$, $\Gamma_\triangle \subseteq \Gamma$, $\mathcal{C}_\triangle \subseteq \mathcal{C}$, so that the counterclockwise endpoints of all arcs in $\Gamma_\triangle$ lie in $E(\mathcal{C}_\triangle)$. The second-level structure then takes each of these canonical pairs and further decomposes them into a collection of canonical pairs $(\Gamma_\tau, \mathcal{C}_\tau)$, so that the clockwise endpoints of all arcs in Γ_τ also lie in $E(\mathcal{C}_\tau)$. Thus the first two levels together ensure that the endpoints of all arcs in Γ_τ lie in $E(\mathcal{C}_\tau)$.

Next, the third and fourth levels decompose each $(\Gamma_\tau, \mathcal{C}_\tau)$ into a family of canonical subsets $(\Gamma_\xi, \mathcal{C}_\xi)$, so that the centers of circles in $\mathcal{C}_\xi$ lie in $\bigcap_{\gamma \in \Gamma_\xi} \omega(\gamma)$. It is easily checked that the set of all fourth-level canonical pairs gives the decomposition $\mathcal{F}$. So, for each fourth-level canonical pair, we compute the number of intersecting pairs of circles using the algorithm described in §2.1, and add up the resulting counts to obtain $\mathcal{I}_1(\Gamma, \mathcal{C})$.

In order to describe the algorithm in detail, we have to define some geometric transforms.

For a circle C of radius r, centered at (a, b), let $\pi(C)$ denote the plane in $\mathbb{R}^3$

$$\pi(C) : z = 2ax + 2by + (r^2 - a^2 - b^2)\,. \tag{7}$$

For a point $p = (\alpha, \beta)$, let $\varpi(p)$ denote the point in $\mathbb{R}^3$

$$\varpi(p) = (\alpha, \beta, \alpha^2 + \beta^2)\,. \tag{8}$$

We will use $\pi^*(C)$ to denote the point dual to the plane $\pi(C)$,

$$\pi^*(C) = (2a, 2b, r^2 - a^2 - b^2)\,, \tag{9}$$

and $\varpi^*(p)$ to denote the plane dual to the point $\varpi(p)$,

$$\varpi^*(C) : z = -\alpha x - \beta y + \alpha^2 + \beta^2\,. \tag{10}$$

It is easily seen that p lies in the exterior of C if and only if $\varpi(p)$ lies above the plane $\pi(C)$, which is the same as saying that the point $\pi^*(C)$ lies below the plane $\varpi^*(p)$.

As in §2, we describe two algorithms. The first algorithm works efficiently when $m^3 \le n$, and the second algorithm, which uses the first algorithm as a subroutine, works well for all ranges of m and n.

3.2. First algorithm. Let r be some sufficiently large fixed constant. We map the counterclockwise endpoints of arcs in Γ to a collection of planes in $\mathbb{R}^3$

$$\{\varpi^*(\alpha) \mid \alpha \text{ is a counterclockwise endpoint of an arc in } \Gamma\}\,,$$

and decompose the space into a set Ξ of $O(r^3)$ simplices, each of which intersects at most m/r planes. Ξ can be computed in $O(m)$ time using the algorithm of Matoušek [Mat2]. We associate with each simplex $\triangle$ a subset $\Gamma_\triangle \subseteq \Gamma$ of arcs and a subset $\mathcal{C}_\triangle \subseteq \mathcal{C}$ of circles. An arc γ, whose counterclockwise endpoint is α, is in $\Gamma_\triangle$ if $\varpi^*(\alpha)$ intersects the interior of $\triangle$, and a circle $C \in \mathcal{C}_\triangle$ if $\pi^*(C) \in \triangle$. Let $A_\triangle \subseteq \Gamma$ denote the set of arcs corresponding to the planes that lie above $\triangle$. It follows from the above discussion that the counterclockwise endpoints of all arcs in $A_\triangle$ lie in $E(\mathcal{C}_\triangle)$, so we output $(A_\triangle, \mathcal{C}_\triangle)$ as one of the first-level canonical pairs.

We recursively decompose each $(\Gamma_\triangle, \mathcal{C}_\triangle)$. The recursion stops when the number of arcs or circles fall below some fixed constant. In this case, we decompose them by a brute-force method.

Next, we decompose each first-level canonical pair further using the same partitioning techniques, except that we now map the clockwise endpoints (instead of the counterclockwise endpoints) of arcs to planes. Let $(\Gamma_\tau, \mathcal{C}_\tau)$ be a second-level canonical pair. The endpoints of all arcs in Γ_τ lie in $E(\mathcal{C}_\tau)$.

Let $\ell_{CCW}(\gamma)$ (respectively, $\ell_{CW}(\gamma)$) denote the line passing through the center of γ and its counterclockwise (respectively, clockwise) endpoint. We map the arcs of Γ_τ to a set of lines $\{\ell_{CCW}(\gamma) \mid \gamma \in \Gamma_\tau\}$ and decompose the plane in $O(|\Gamma_\tau|)$ time into $O(r^2)$ triangle, each of which intersects at most $|\Gamma_\tau|/r$ lines, again using the technique of [Mat2]. We associate with each triangle ζ a subset of arcs $\Gamma_\zeta \subseteq \Gamma_\tau$ and a subset of circles $\mathcal{C}_\zeta \subseteq \mathcal{C}_\tau$. An arc $\gamma \in \Gamma_\zeta$ if $\ell_{CCW}(\gamma)$ intersects ζ, and a circle $C \in \mathcal{C}_\zeta$ if the center of C lies in ζ. We also associate two other subsets A_ζ and B_ζ of Γ_τ with ζ. An arc γ is in A_ζ (respectively, B_ζ) if both γ and ζ lie below (respectively, above) the line $\ell_{CCW}(\gamma)$. We output $(A_\zeta, \mathcal{C}_\zeta)$ and $(B_\zeta, \mathcal{C}_\zeta)$ as third-level canonical pairs and continue decomposing $(\Gamma_\zeta, \mathcal{C}_\zeta)$ recursively.

Next, for each third-level canonical pair $(\Gamma_\zeta, \mathcal{C}_\zeta)$, we map each arc $\gamma \in \Gamma_\zeta$ to the line $\ell_{CW}(\gamma)$ and apply the same partitioning scheme. For each triangle ξ, Γ_ξ, and $\mathcal{C}_\xi$ are defined in the same way as in the third-level structure. If $\ell_{CCW}(\gamma)$, for each arc $\gamma \in \Gamma_\zeta$, lies below (respectively, above) ξ, we define $A_\xi \subseteq \Gamma_\zeta$ to be the set of arcs γ such that $\ell_{CW}(\gamma)$ lies below (respectively, above) both ξ and γ. We output $(A_\xi, \mathcal{C}_\xi)$ as a fourth-level canonical pair, and continue decomposing $(\Gamma_\xi, \mathcal{C}_\xi)$ recursively.

Each fourth-level canonical pair $(\Gamma_\xi, \mathcal{C}_\xi)$ has the properties that the endpoints of all arcs in Γ_ξ lie in $E(\mathcal{C}_\xi)$ and that centers of all circles in $\mathcal{C}_\xi$ lie in $\bigcap_{\gamma \in \Gamma_\xi} \omega(\gamma)$. By Lemma 3.2, an arc $\gamma \in \Gamma_\xi$ intersects a circle $C \in \mathcal{C}_\xi$ if and only if C intersects the circle containing γ. Therefore, the number of intersecting pairs of arcs in Γ_ξ and of circles in $\mathcal{C}_\xi$ can be counted in time $O(|\mathcal{C}_\xi|^{3+\epsilon} + |\Gamma_\xi| \log |\mathcal{C}_\xi|)$ by the algorithm described in §2.1. For the sake of convenience, we will consider the circle intersection counting procedure as a fifth-level step of the algorithm.

Let $T^{(i)}(a, b)$ denote the maximum time spent in processing a set of a arcs and another set of b circles at level i (i.e., the time spent constructing and processing the structures at levels $\geq i$). It follows from §2.1 that $T^{(5)}(a, b) = O(a^{3+\epsilon} + b \log a)$. For $i \leq 4$ we decompose the problem into κ level-i subproblems, each involving at most a/r arcs and b_j circles, and into κ level-$(i + 1)$ problems, each involving at most a arcs and b circles. By construction, $\sum_j b_j = b$, $\kappa = O(r^3)$ for $i = 1, 2$, and $\kappa = O(r^2)$ for $i = 3, 4$. Since we spend $O(a + b)$ time in computing the set of simplices and various subsets of arcs and circles, we obtain the following recurrence:

$$T^{(i)}(a,b) = \begin{cases} O(b \log a + a^{3+\epsilon}) & \text{if } i = 5, \\ \kappa \cdot T^{(i+1)}(a,b) + \sum_{j=1}^{\kappa} T^{(i)}\left(\frac{a}{r}, b_j\right) + & \text{if } i < 5, \\ O(a+b) & \end{cases} \tag{11}$$

where $\kappa = O(r^3)$ and $\sum_{j=1}^{\kappa} b_j = b$. The solution of the above recurrence is easily seen to be

$$T^{(i)}(a, b) = O(a^{3+\epsilon} + b \log^{6-i} a) \quad \text{for } i \leq 5 .$$

Hence we can conclude that, given a set of m circular arcs and another set of n circles, we can count the number of intersection points between them in time $O(n \log^5 m + m^{3+\epsilon})$, for any $\epsilon > 0$.

Remark 3.3. (i) The running time can be improved to $O(n \log m + m^{3+\epsilon})$ by an appropriate modification of the algorithm, but for our purposes the time bound derived above is sufficient.

(ii) If the endpoints of arcs in Γ are already known to lie in $E(\mathcal{C})$, we do not have to construct the first two levels.

3.3. Second algorithm. We now describe another algorithm that works well for all ranges of m and n. In the above algorithm we mapped the endpoints of arcs to planes/lines and circles to points, and constructed a multilevel structure. We follow the same approach, but there are two key differences. The first difference is that we flip the roles of arcs and circles, i.e., we now map circles to planes/lines and endpoints of arcs to points. The second difference is that we stop the recursion when the ratio of the number of circles to the number of points becomes large and solve the problem directly using the first algorithm.

In more detail, we map the circles of $\mathcal{C}$ to a set of planes $\{\pi(C) \mid C \in \mathcal{C}\}$, and partition $\mathbb{R}^3$ into a set Ξ of $O(r^3)$ simplices each of which intersects at most n/r planes [Mat2]. We associate with each simplex $\Delta \in \Xi$ a subset $\Gamma_\Delta \subseteq \Gamma$ and another subset $\mathcal{C}_\Delta \subseteq \mathcal{C}$ of circles. An arc γ, whose counterclockwise endpoint is α, is in Γ_Δ if $\varpi(\alpha) \in \Delta$, and a circle C is in $\mathcal{C}_\Delta$ if $\pi(C)$ intersects the interior of Δ. Let $L_\Delta \subseteq \mathcal{C}$ denote the set of circles C such that $\pi(C)$ lies below Δ. By the above discussion, the counterclockwise endpoints of all arcs in Γ_Δ lie in $E(L_\Delta)$. We output $(\Gamma_\Delta, L_\Delta)$ as one of the first-level canonical pairs.

We recursively construct the first-level structure for $(\Gamma_\Delta, \mathcal{C}_\Delta)$. The recursion stops when $|\Gamma_\Delta|^3 \le |\mathcal{C}_\Delta|$. In this case, we process $(\Gamma_\Delta, \mathcal{C}_\Delta)$ using the first algorithm.

Next, for each first-level canonical pair we apply the same procedure for the clockwise endpoints of arcs. Let $(\Gamma_\tau, \mathcal{C}_\tau)$ be a second-level canonical pair. As earlier, the endpoints of arcs in Γ_τ lie in $E(\mathcal{C}_\tau)$.

In the third level, we dualize the centers of circles of $\mathcal{C}_\tau$ to a set of lines, and partition the plane into $O(r^2)$ triangles so that each triangle intersects at most $|\mathcal{C}_\tau|/r$ lines. We associate with each triangle ζ three subsets $\mathcal{C}_\zeta$, L_ζ, and U_ζ of $\mathcal{C}_\tau$ and a subset Γ_ζ of Γ_τ. An arc $\gamma \in \Gamma_\zeta$ if the point dual to $\ell_{CCW}(\gamma)$ lies in ζ. A circle C is in $\mathcal{C}_\zeta$ (respectively, L_ζ, U_ζ) if the line dual to its center intersects (respectively, lies above, lies below) ζ. Let Γ'_ζ (respectively, Γ''_ζ) be the subset of arcs in Γ_ζ that lie above (respectively, below) the line $\ell_{CCW}(\gamma)$. We output $(L_\zeta, \Gamma''_\zeta)$ and (U_ζ, Γ'_ζ) as third-level canonical pairs. We recursively continue constructing the third-level structure on $(\Gamma_\zeta, \mathcal{C}_\zeta)$. The recursion stops when $|\Gamma_\zeta|^3 \le |\mathcal{C}_\zeta|$. In this case, we invoke the first algorithm.

Next, for each third-level canonical pair $(\Gamma_\zeta, \mathcal{C}_\zeta)$, we apply a similar two-dimensional decomposition to the lines dual to the centers of circles in $\mathcal{C}_\zeta$ and to the points dual to the lines $\ell_{CW}(\gamma)$ for $\gamma \in \Gamma_\zeta$. Each fourth-level canonical pair $(\Gamma_\xi, \mathcal{C}_\xi)$ has the property that the endpoints of arcs in Γ_ξ lie in $E(\mathcal{C}_\xi)$, and that the centers of circles in $\mathcal{C}_\xi$ lie in $\bigcap_{\gamma\in\Gamma_\xi} \omega(\gamma)$. The number of intersection points between $\mathcal{C}_\xi$ and the circles containing the arcs of Γ_ξ can now be computed in time $O(a_\xi^{3/4+\epsilon} b_\xi^{3/4} + a_\xi^{1+\epsilon} + b_\xi^{1+\epsilon})$ using Theorem 2.7.

Let $T^{(i)}(a, b)$ denote the maximum running time of the algorithm at level i, defined as above. Notice that we invoke the first algorithm only when $a^3 \le b$, so the time required by the first algorithm is only $O(b^{1+\epsilon})$. Following the same argument as in the previous subsection, we obtain the following recurrence:

(12)

$$T^{(i)}(a,b) = \begin{cases} O(a^{3/4+\epsilon}b^{3/4} + a^{1+\epsilon} + b^{1+\epsilon}) & \text{for } i = 5\,, \\ O(b^{1+\epsilon}) & \text{for } i < 5, \quad a^3 \le b\,, \\ \kappa \cdot T^{(i+1)}(a,b) + \sum_{j=1}^{\kappa} T^{(i)}\left(a_j, \frac{b}{r}\right) + O(a+b) & \text{for } i < 5, \quad a^3 > b\,, \end{cases}$$

where $\kappa = O(r^3)$ and $\sum_j a_j = a$. The solution of the above recurrence is

$$T^{(i)}(a,b) = O(a^{3/4+\epsilon}b^{3/4} + a^{1+\epsilon} + b^{1+\epsilon}) \quad \text{for } i \le 5\,.$$

Hence, $\mathcal{I}_1(\Gamma, \mathcal{C})$ can be counted in time $O(m^{3/4+\epsilon}n^{3/4} + m^{1+\epsilon} + n^{1+\epsilon})$.

3.4. Counting $\mathcal{I}_2(\Gamma, \mathcal{C})$. $\mathcal{I}_2(\Gamma, \mathcal{C})$ is counted using an algorithm very similar to the one we just described. The only difference is that conditions (ii) and (iii) for $\mathcal{F}$ now become:

(ii′) For every pair $(\gamma, C) \in (\Gamma_i, \mathcal{C}_i)$, the endpoints of γ lie in the interior of C and the center of C lies in $\bar{\omega}(\gamma)$, and

(iii′) For every pair $(\gamma, C) \in \Gamma \times \mathcal{C}$, which satisfies the first four constraints of condition II, there is a unique i such that $\gamma \in \Gamma_i$ and $C \in \mathcal{C}_i$.

To this end, we construct a four-level structure similar to that for counting $\mathcal{I}_1(\Gamma, \mathcal{C})$, but we define the subsets of arcs at each level in a somewhat different way, to meet the new kind of constraints. For example, at the first level of the first algorithm, we define $A_\triangle \subseteq \Gamma$ to be the set of arcs corresponding to planes that lie below $\triangle$, and at the third level we define A_ξ (respectively, B_ξ) to be the set of arcs γ such that γ lies above (respectively, below) $\ell_{CCW}(\gamma)$ and $\triangle$ lies below (respectively, above) $\ell_{CCW}(\gamma)$. Similarly, we redefine $L_\triangle, L_\xi, U_\xi$ in the second algorithm. Following the same analysis as above, we can show that $\mathcal{I}_2(\Gamma, \mathcal{C})$ can be counted in time $O(m^{3/4+\epsilon}n^{3/4} + m^{1+\epsilon} + n^{1+\epsilon})$.

3.5. Counting $\mathcal{I}_3(\Gamma, \mathcal{C})$. Counting $\mathcal{I}_3(\Gamma, \mathcal{C})$ is relatively simpler, because the conditions on (γ, C') are now defined as a conjunction of only two constraints. Suppose we want to count the number of pairs (γ, C) such that the counterclockwise endpoint of γ lies in the exterior of C and the clockwise endpoint of γ lies in the interior of C; the other case can be handled symmetrically.

The first-level structure is exactly the same as that for counting $\mathcal{I}_1(\Gamma, \mathcal{C})$. Next, for each first-level canonical pair $(\Gamma_\triangle, \mathcal{C}_\triangle)$, we map the clockwise endpoints of arcs in $\Gamma_\triangle$ to a set of planes using (10) and apply the same decomposition scheme, except that for each simplex τ we output a canonical pair $(U_\tau, \mathcal{C}_\tau)$, if the planes corresponding to arcs in U_τ lie below τ. By Lemma 3.1, each $\gamma \in U_\tau$ intersects every circle $C \in \mathcal{C}_\tau$. The total running time is again $O(m^{3/4+\epsilon}n^{3/4} + m^{1+\epsilon} + n^{1+\epsilon})$.

Hence, we can conclude the following theorem.

THEOREM 3.4. *Given a set of m circular arcs and a set of n circles, we can count the number of intersection points between them in time* $O(m^{3/4+\epsilon}n^{3/4} + m^{1+\epsilon} + n^{1+\epsilon})$.

3.6. The case of unit circles and arcs. If all the circles in $\mathcal{C}$ and the circles supporting the arcs of Γ have the same radius, say 1, we can count the number of intersections between $\mathcal{C}$ and Γ more efficiently by modifying the algorithms given above. First, instead of using the algorithm described in §2, we use the algorithm of Agarwal et al. [AASS] to count the number of intersection points between two families of unit circles. Second, it turns out that we only need to apply the decomposition schemes in two dimensions at all levels of the structure (even in the previous algorithms the third and fourth levels apply two-dimensional decomposition schemes; only the first two levels of the structure required three-dimensional schemes). We will describe how to construct the first-level structure for both algorithms; the second-level structure can be handled analogously.

For a point p in the plane, let $D(p)$ denote the unit disk centered at p. We construct the first-level structure of the first algorithm as follows. We map the counterclockwise endpoints of arcs in Γ to a set of unit circles

$$D(\Gamma) = \{D(\alpha) \mid \alpha \text{ is the counterclockwise endpoint of an arc } \gamma \in \Gamma\}.$$

We compute a $\frac{1}{r}$-net $R \subseteq D(\Gamma)$ of size $O(r \log r)$ in time $O(m)$ (see §2 for a definition of a $\frac{1}{r}$-net). We compute the vertical decomposition of the arrangement of R, i.e., draw a vertical line from every vertex of the arrangement and all the points of vertical tangency of every circle in both directions until such a line hits an edge of the arrangement. If there is no such edge, the line is extended to infinity. The vertical decomposition of R partitions the plane into $O(r^2 \log^2 r)$ trapezoidal cells. Since R is a $\frac{1}{r}$-net, each cell of the vertical decomposition intersects at most m/r circles of $D(\Gamma)$. We associate with each cell Δ a subset $\Gamma_\Delta \subseteq \Gamma$ of arcs and another subset $\mathcal{C}_\Delta \subseteq \mathcal{C}$ of circles; an arc γ, whose counterclockwise endpoint is α, is in Γ_Δ if $D(\alpha)$ intersects Δ, and a circle C is in $\mathcal{C}_\Delta$ if the center of C lies in Δ. Let $A_\Delta \subseteq \Gamma$ be the set of arcs corresponding to discs that contain Δ in their exterior. It is easily seen that the counterclockwise endpoints of arcs in A_Δ lie in $E(\mathcal{C}_\Delta)$, so we output $(A_\Delta, \mathcal{C}_\Delta)$ as one of the first-level canonical pairs. We recursively decompose $(\Gamma_\Delta, \mathcal{C}_\Delta)$.

The second-level structure can also be modified similarly. The third and fourth levels remain the same. Finally, for each fourth-level canonical pair, we use the algorithm of [AASS] to count the number of intersection points between two families of circles, so $T^{(5)}(a, b) = O(a^2 + b \log a)$. Furthermore, $\kappa = O(r^2 \log r)$ for all $i \leq 4$, so we have the following recurrence for $i \leq 4$:

$$T^{(i)}(a,b) = O(r^2 \log^2 r) \cdot T^{(i+1)}(a,b) + \sum_{j=1}^{O(r^2 \log^2 r)} T^{(i)}\left(\frac{a}{r}, b_j\right) + O(a+b),$$

where $\sum_j b_j = b$. The solution of the above recurrence is $O(a^{2+\epsilon} + b \log^{6-i} a)$.

At the first-level structure of the second algorithm, we choose a $\frac{1}{r}$-net R' of $\mathcal{C}$ and compute the vertical decomposition of its arrangement. For each cell Δ of the vertical decomposition, we associate the subset $\mathcal{C}_\Delta \subseteq \mathcal{C}$ of circles that intersect Δ and a subset of arcs $\Gamma_\Delta \subseteq \Gamma$ whose counterclockwise endpoints lie in Δ. The set L_Δ is now defined to be the subset of circles that contain Δ in their exterior. We output the pair $(L_\Delta, \mathcal{C}_\Delta)$ and recursively decompose the pair $(\Gamma_\Delta, \mathcal{C}_\Delta)$.

By [AASS], $T^{(5)}(a, b)$ is now $O(a^{2/3+\epsilon} b^{2/3} + a^{1+\epsilon} + b^{1+\epsilon})$. Since $\kappa = O(r^2 \log r)$, the solution of (12) now becomes $O(a^{2/3+\epsilon} b^{2/3} + a^{1+\epsilon} + b^{1+\epsilon})$, which yields the following result.

THEOREM 3.5. *Given a set of m circular arcs, each of radius* 1, *and another set of n unit circles, we can count the number of intersection points between them in time $O(m^{2/3+\epsilon} n^{2/3} + m^{1+\epsilon} + n^{1+\epsilon})$.*

4. Counting arc intersections. In this section we obtain the main result of the paper: We present an algorithm to count the number of intersection points in a collection Γ of n arbitrary circular arcs. As in the previous section, we assume that all arcs in Γ are in general position. The algorithm is based on the following two lemmas.

LEMMA 4.1. *An arc γ of a circle C intersects an arc γ' of another circle C' at two points if and only if $|\gamma \cap C'| = 2$ and $|\gamma' \cap C| = 2$ (see Fig.* 5(i)).

Proof. The "only if" part is trivial. The "if" part follows from the observation that C and C' must intersect at two points, both of which lie in both γ and γ'. □

The case where two given arcs intersect each other in exactly one point is more involved and depends on the pattern of intersections between each arc and the circle containing the other.

LEMMA 4.2. *An arc γ of a circle C intersects an arc γ' of another circle C' at exactly one point if and only if one of the following conditions holds:*

(i) $|\gamma \cap C'| = 1$ *and* $|\gamma' \cap C| = 2$.

(ii) $|\gamma \cap C'| = 2$ *and* $|\gamma' \cap C| = 1$ (*see Fig.* 5(ii)).

(iii) $|\gamma \cap C'| = 1$ *and* $|\gamma' \cap C| = 1$, *and the following additional condition holds. Divide γ into two equal subarcs, and let $\gamma_{1/2}$ be the subarc for which $|\gamma_{1/2} \cap C'| = 1$. Define $\gamma'_{1/2}$ in an analogous manner. Then both arcs $\gamma_{1/2}$ and $\gamma'_{1/2}$ fully lie on the same side of the line connecting the centers of the circles C, C' (see Fig.* 5(iii)).

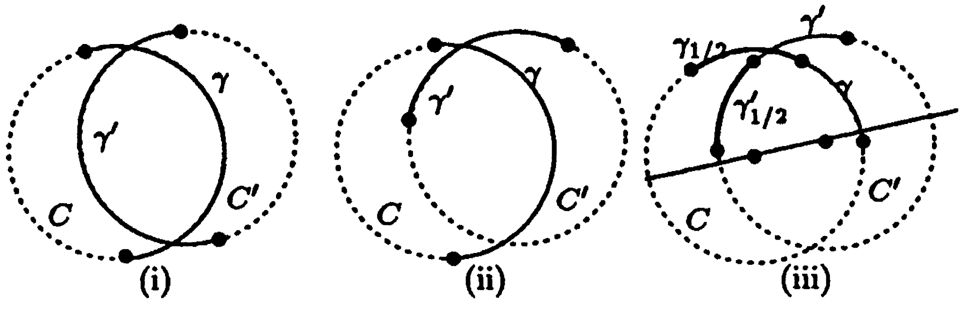


FIG. 5. *Different cases of arcs intersections.*

Proof. To prove the "only if" part of the lemma, suppose $|\gamma \cap \gamma'| = 1$. Then clearly $|\gamma \cap C'| \geq 1$ and $|\gamma' \cap C| \geq 1$. Suppose neither condition (i) nor (ii) holds. Then necessarily $|\gamma \cap C'| = 1$ and $|\gamma' \cap C| = 1$. (If both these numbers were 2, Lemma 4.1 would imply that $|\gamma \cap \gamma'| = 2$, contrary to assumption.) Let γ be an arc that intersects a circle C' in exactly one point. Then it is easily verified that its corresponding halfarc $\gamma_{1/2}$ lies completely on one side of the line connecting the centers of C' and one one side of the circle containing γ, which is the same side containing the point of intersection between γ and C'. This observation easily implies that the second part of condition (iii) is also satisfied.

Consider now the "if" part of the lemma. If γ and γ' satisfy condition (i), then γ' contains the two points of intersection between C and C', and γ contains just one of these points, so clearly these arcs intersect at exactly one point. A symmetric argument applies if condition (ii) holds. Suppose condition (iii) holds, so that $|\gamma \cap C'| = 1$ and $|\gamma' \cap C| = 1$. If γ and γ' do not intersect, then $p = \gamma \cap C'$ and $q = \gamma' \cap C$ are the two distinct points of intersection between C and C', each lying on a different side of the line connecting the circle centers. But then the above observation implies that each of the corresponding halfarcs $\gamma_{1/2}$ and $\gamma'_{1/2}$ fully lie on a different side of this line, contradicting the second part of (iii). This completes the proof of the lemma. □

Lemmas 4.1 and 4.2 suggest the following multilevel structure to count the number of pairs of intersecting arcs in $\Gamma \times \Gamma'$. The preceding lemmas imply that we must count pairs of arcs, (γ, γ'), that satisfy one of the following four conditions (where C is the circle containing γ and C' is the circle containing γ'):

(a) $|\gamma \cap C'| = 2$ and $|\gamma' \cap C| = 2$.

(b) $|\gamma \cap C'| = 1$ and $|\gamma' \cap C| = 2$.

(c) $|\gamma \cap C'| = 2$ and $|\gamma' \cap C| = 1$.

(d) $|\gamma \cap C'| = 1$ and $|\gamma' \cap C| = 1$, and the two halfarcs of condition (iii) of Lemma 4.2 lie on the same side of the line connecting the centers of the circles C, C'.

Finding those pairs of arcs that satisfy one of the conditions (a), (b), or (c) is relatively simple, applying appropriate variants of the machinery presented in the preceding section. Consider for example condition (a). By Lemma 3.2, we can find all these pairs by constructing a nine-level data structure; the first four levels are the same as the first four levels in the data structures of the preceding section, the next four levels are symmetric variants of the first four levels (obtained by interchanging the roles of Γ and Γ'), and the last level tests for intersections between the corresponding circles C, C'. In a similar manner (but using fewer levels) we can find all pairs satisfying condition (b) or (c).

Condition (d) is somewhat more involved. Again we use a multilevel structure. The first two levels enforce, as in the preceding section, the conditions that one endpoint of γ lies inside

C' and one endpoint lies outside C'; the next two levels enforce the symmetric condition for γ' and C. Each resulting canonical pair of subsets (Γ_i, Γ'_i) now has the property that $|\gamma \cap C'| = 1$ and $|\gamma' \cap C| = 1$, for each $\gamma \in \Gamma_i$, $\gamma' \in \Gamma'_i$, with C and C' defined as above.

We next enforce the second part of condition (d). For an arc γ, let $\bar{\omega}(\gamma)$ denote the double wedge formed by $\omega(\gamma) \cup \omega'(\gamma)$. It is easily verified that the second part of (d) is equivalent to requiring that the center of C' lies outside the double-wedge $\bar{\omega}(\gamma_{1/2})$ and the center of C lies outside the double-wedge $\bar{\omega}(\gamma'_{1/2})$. These two subconditions are easy to test for, using standard range counting techniques. That is, we take the collection of centers c' of the circles C' containing the arcs of Γ'_i, and the collection of double wedges that are complements of $\bar{\omega}(\gamma_{1/2})$, $\gamma \in \Gamma_i$, and process them, as in the preceding section, to obtain a canonical collection of pairs of subsets, $(\Gamma_{ij}, \Gamma'_{ij})$, so that, for each such pair, the centers of a circle containing the arc of Γ'_{ij} lie outside every double-wedge $\bar{\omega}(\gamma_{1/2})$, $\gamma \in \Gamma_{ij}$. Finally we apply a symmetric variant of this step to each of these canonical pairs, with the roles of Γ and Γ' being interchanged. The resulting new canonical pairs now fully satisfy condition (d), and the final counting is thus straightforward.

We omit the details of the analysis of the running time of the algorithm, since it is nearly identical to the analysis given in the preceding section. We summarize our results in the following theorem.

THEOREM 4.3. *Given a set Γ of n circular arcs, we can count the number of intersection points in Γ in time $O(n^{3/2+\epsilon})$, for any $\epsilon > 0$.*

If all arcs in Γ have the same radius then, as in §3.6, all levels of the data structure use only two-dimensional decomposition schemes. Thus, following the analysis of the previous section, we can easily conclude the following result.

THEOREM 4.4. *Given a collection Γ of n arcs of the same radius, we can count the number of intersection points in Γ in time $O(n^{4/3+\epsilon})$, for any $\epsilon > 0$.*

5. Conclusion. In this paper we presented efficient algorithms for counting intersections in collections of circles, of circular arcs, and of circles or circular arcs of some fixed radius. Although our algorithms are significantly faster than the best previously known algorithms, we believe that their running time can be further improved, because the best known lower bound for these problems is only $\Omega(n \log n)$. As a first goal, can circular arc intersections be counted in time close to $O(n^{4/3})$, as is the case for collections of segments? We showed that this is the case for circular arcs of the same radius.

Finally, the techniques presented here seem to be quite general. An open problem is to extend them to counting intersections for other types of arcs.

REFERENCES

[Ag1] P. K. AGARWAL, *Partitioning arrangements of lines: I. An efficient deterministic algorithm*, Discrete Comput. Geom., 5 (1990), pp. 449–483.

[Ag2] ———, *Partitioning arrangements of lines: II. Applications*, Discrete Comput. Geom., 5 (1990), pp. 533–573.

[AASS] P. K. AGARWAL, B. ARONOV, M. SHARIR, AND S. SURI, *Selecting distances in the plane*, Proc. 6th ACM Symposium on Computational Geometry, 1990, pp. 321–331; Algorithmica, to appear.

[BO] J. L. BENTLEY AND and T. OTTMANN, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., C–28 (1979), pp. 643–647.

[Cha] B. CHAZELLE, *Reporting and counting segment intersections*, J. Comput. Systems Sci., 32 (1986), pp. 156–182.

[Chb] ———, *An optimal convex hull algorithm and new results on cuttings*, Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 29–38.

[CE] B. CHAZELLE AND H. EDELSBRUNNER, *An optimal algorithm for intersecting line segments in the plane*, J. Assoc. Comput. Mach., 39 (1992), pp. 1–54.

[CEGS1] B. CHAZELLE, H. EDELSBRUNNER, L. GUIBAS, AND M. SHARIR, *A singly-exponential stratification scheme for real semi-algebraic varieties and its applications*, Proc. 16th International Colloquium on Automata, Languages and Programming, 1989, pp. 179–193; Theoret. Comput. Sci, 84 (1991), pp. 77–105.

[CEGS2] ———, *Algorithms for bichromatic line segment problems and polyhedral terrains*, Algorithmica, to appear.

[CS1] B. CHAZELLE AND M. SHARIR, *An algorithm for generalized point location and its application*, J. Symb. Comput., 10 (1990), pp. 281–309.

[CSW] B. CHAZELLE, M. SHARIR, AND E. WELZL, *Quasi-optimal upper bounds for simplex range searching and new zone theorems*, Algorithmica, 8 (1992), pp. 407–430.

[Cl] K. CLARKSON, *New applications of random sampling in computational geometry*, Discrete Comput. Geom., 2 (1987), pp. 195–222.

[CEGSW] K. CLARKSON, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND E. WELZL, *Combinatorial complexity bounds for arrangements of curves and spheres*, Discrete Comput. Geom., 5 (1990), pp. 99–160.

[CS2] K. CLARKSON AND P. SHOR, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[Ed] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[EGS] H. EDELSBRUNNER, L. J. GUIBAS, AND J. STOLFI, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.

[GOS] L. GUIBAS, M. OVERMARS, AND M. SHARIR, *Counting and reporting intersections in arrangements of line segments*, Tech. Report 434, Dept. Computer Science, New York University, New York, NY, March 1989.

[HW] D. HAUSSLER AND E. WELZL, *ϵ-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.

[Mat1] J. MATOUŠEK, *Construction of ϵ-nets*, Discrete Comput. Geom., 5 (1990), pp. 427–448.

[Mat2] ———, *Approximations and optimal geometric divide-and-conquer*, Proc. 23rd ACM Symposium on Theory of Computing, 1991.

[Mu] K. MULMULEY, *A fast planar partition algorithm*, I, J. Symb. Comput., 10 (1990), pp. 253–280.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 794–806, August 1993

RAY SHOOTING AND PARAMETRIC SEARCH*

PANKAJ K. AGARWAL† AND JIŘÍ MATOUŠEK‡

Abstract. Efficient algorithms for the *ray shooting* problem are presented: Given a collection Γ of objects in $\mathbb{R}^d$, build a data structure so that, for a query ray, the first object of Γ hit by the ray can be quickly determined. Using the parametric search technique, this problem is reduced to the *segment emptiness* problem. For various ray shooting problems, space/query-time trade-offs of the following type are achieved: For some integer b and a parameter m ($n \le m \le n^b$) the queries are answered in time $O((n/m^{1/b})\log^{O(1)} n)$, with $O(m^{1+\varepsilon})$ space and preprocessing time ($\varepsilon > 0$ is arbitrarily small but fixed constant). $b = \lfloor d/2 \rfloor$ is obtained for ray shooting in a convex d-polytope defined as an intersection of n half spaces, $b = d$ for an arrangement of n hyperplanes in $\mathbb{R}^d$, and $b = 3$ for an arrangement of n half planes in $\mathbb{R}^3$. This approach also yields fast procedures for finding the first k objects hit by a query ray, for searching nearest and farthest neighbors, and for the hidden surface removal. All the data structures can be maintained dynamically in amortized time $O(m^{1+\varepsilon}/n)$ per insert/delete operation.

Key words. ray shooting, arrangements, convex polytope, range searching, parametric search

AMS subject classifications. 52B11, 68P05, 68Q20, 68Q25, 68U05

1. Introduction.

Consider the following *ray shooting* problem: *Given a collection Γ of n objects in $\mathbb{R}^d$, build a data structure so that, for a query ray ρ, we can quickly determine the first object of* Γ *intersected by* ρ.

The ray shooting problem has received much attention in the past few years because of its applications in graphics and other geometric problems [11], [20], [1], [13], [6], [4], [9], [7]. But most of the work done so far has been for the planar case where Γ is a collection of line segments in $\mathbb{R}^2$. Chazelle and Guibas proposed an optimal algorithm for the special case where Γ is the boundary of a simple polygon [11]. Their algorithm answers a ray shooting query in $O(\log n)$ time using $O(n)$ space (see also [9]). If Γ is a collection of arbitrary segments in the plane, the best known algorithm answers a ray shooting query in time $O(\frac{n}{\sqrt{m}} \log^{O(1)} n)$ using $O(m^{1+\varepsilon})$ space and preprocessing[1] [1], [6], [4]. Although no lower bound is known for this case, it is conjectured that this bound is close to optimal. However, in three and higher dimensions the problem is far from being solved. For example, no efficient ray shooting algorithm is known for a convex d-polytope for $d > 3$. Even in $\mathbb{R}^3$, nontrivial solutions have been obtained only very recently (cf. [4], [7]).

In this paper, we present ray shooting algorithms for several cases in higher dimensions ($d \ge 3$), including a convex polytope, a collection of n hyperplanes in $\mathbb{R}^d$, and a collection of n half planes in $\mathbb{R}^3$. We will use a unified approach for all cases, which is, roughly speaking, a binary search along the query ray ρ. In order to make this approach work, we need to handle two problems: (i) Perform a binary search without computing all intersection points of ρ and the objects of Γ, and (ii) given a point $x \in \rho$, determine whether the first intersection point of Γ and ρ lies before or after x.

To handle the first problem we perform an implicit binary search using the *parametric search* technique of Megiddo [26], and, to handle the second problem, we use suitable range searching structures for detecting an intersection between ρ and the objects of Γ. Our approach can be easily extended to report the first k objects hit by the query ray.

*Received by the editors June 10, 1991; accepted for publication (in revised form) May 20, 1992. Work by the first author has been supported by National Science Foundation grant CCR-91-06514. Part of the work was done while the second author was visiting School of Mathematical Sciences, Georgia Tech University, Atlanta, Georgia.

†Computer Science Department, Duke University, Durham, North Carolina 27706 (pankaj @cs. duke.edu).
‡Department of Applied Mathematics, Charles University, Prague, Czech.

[1]Throughout this paper, ε denotes an arbitrarily small positive constant. The multiplicative constants in the asymptotic bounds may depend on ε.

The range searching algorithms we have at our disposal usually admit space/query-time trade-offs: the more space (and preprocessing time) we use, the faster the queries can be answered. Such a trade-off then transfers to the ray shooting results. A usual form of this trade-off is the following: There are two fixed integers b, c (specific to the considered problem), such that with $O(m^{1+\varepsilon})$ space and preprocessing time, where $n \le m \le n^b$, a query can be answered in time $O(\frac{n}{m^{1/b}} \log^c n)$ [12]. Note that since we require $m \ge n$, the smallest amount of space we consider in this trade-off is $O(n^{1+\varepsilon})$. Hence the $\log^c n$ factor plays a role only when m is close to n^b; actually it expresses the query complexity for the maximum permissible amount of space. For the sake of brevity, we just say that such a problem admits a *standard trade-off* with a certain value of b. It is understood that c is always a (reasonably small) constant. For the sake of simplicity, we will not discuss the best value of c. Also, in many cases the space and/or preprocessing time can be somewhat reduced because of improved results in range searching (e.g., for $m = n$, or $m = n^b$); we will not pay too much attention to this either.

In this paper, we consider the following specific ray shooting problems:

(i) Ray shooting in an arrangement of n hyperplanes in $\mathbb{R}^d$. We get a standard trade-off with $b = d$. Previously, efficient solutions were known only for $d \le 3$. Moreover, they did not support insertion/deletion of planes for $d = 3$.

(ii) Ray shooting in a convex d-polytope defined as the intersection of n half spaces. We get a standard trade-off with $b = \lfloor d/2 \rfloor$. Previously, nontrivial solutions were known only for $d \le 3$ [17].

(iii) Ray shooting in an arrangement of n half planes in $\mathbb{R}^3$. We obtain a standard trade-off with $b = 3$. The previously best known result answered a query in time $O(n^{16/15+\varepsilon}/m^{4/15})$ using $O(m^{1+\varepsilon})$ space and preprocessing [4].

By a well-known reduction, a suitable ray shooting algorithm yields a procedure that can process a set of n points in $\mathbb{R}^d$ into a data structure, so that the nearest or the farthest neighbor of a query point can be computed quickly; we get a standard trade-off with $b = \lceil d/2 \rceil$ [18]. It can also report the first k neighbors, for any given $k \le n$, at an additional cost of $k \log^2 n$ in the query time.

Finally, we consider the following hidden surface removal problem: Given a set T of n nonintersecting triangles in $\mathbb{R}^3$ and a view point $z = +\infty$, compute the *visibility map* of T, that is, the subdivision of the viewing plane so that the same triangle is visible from all the points of a face. The goal is to come up with an "output-sensitive" algorithm. Although there have been several output-sensitive algorithms for special cases, only very recently de Berg et al. [7] presented an output-sensitive algorithm for the general case. Their algorithm relies on a fast procedure for ray shooting among a collection of "curtains"; a *curtain* is an unbounded vertical triangle with two of its edges being vertical rays extending to $-\infty$. We shall present a faster procedure for ray shooting among curtains, which will improve the running time of the hidden surface removal algorithm of [7].

Tables 1 and 2 offer a succinct comparison of our ray shooting and k-intersection results with the previously known ones. In these tables, the space and preprocessing time are usually $O(m^{1+\varepsilon})$ (and better in some cases; see the appropriate sections).

TABLE 1
Ray shooting results.

Objects	Previous results	New results
d-polytope	$O(\log n), d \le 3$ [17]	$O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$
hyperplanes in $\mathbb{R}^d$	$O(\frac{n^{1+\varepsilon}}{m^{1/d}}), d \le 3$ [4]	$O(\frac{n}{m^{1/d}} \log^4 n)$
half planes in $\mathbb{R}^3$	$O(\frac{n^{16/15+\varepsilon}}{m^{4/15}})$ [4]	$O(\frac{n}{m^{1/3}} \log^4 n)$
curtains in $\mathbb{R}^3$	$O(\frac{n^{2+\varepsilon}}{m})$ [7]	$O(\frac{n}{\sqrt{m}} \log^6 n)$

TABLE 2
k-intersection results.

Objects	Previous results	New results
hyperplanes in $\mathbb{R}^d$	None	$O(\frac{n}{m^{1/d}} \log^4 n + k)$
segments in $\mathbb{R}^2$	$O(\frac{n}{\sqrt{m}} \log^{O(1)} n + k \log n)$	$O(\frac{n}{\sqrt{m}} \log^{O(1)} n + k)$
triangles in $\mathbb{R}^3$	None	$O(\frac{n^{16/15}}{m^{4/15}} \log^{O(1)} n + k)$

This paper is organized as follows. Section 2 introduces the ray shooting problem in a general setting and gives a reduction to the segment emptiness problem using Megiddo's parametric search technique. In §3, we discuss specific applications of this general result to ray shooting and related problems in various situations. Ray shooting algorithms are applied to proximity problems and to the hidden surface removal problem in §4 and §5, respectively. We finally conclude by mentioning some open problems.

2. Ray shooting using parametric search. It will be convenient to formulate the ray shooting problem in a reasonably general setting. Let $\mathcal{G}$ be a class of (topologically closed) geometric objects in $\mathbb{R}^d$ (in the examples we consider, these will be hyperplanes or parts of them), and let Γ be some set of n objects of $\mathcal{G}$. Further let $\mathcal{R}$ be a set of *admissible rays.* Let $o(\rho)$ denote the origin point of a ray ρ. The points of every ray will be ordered increasingly along the ray starting from its origin, i.e., for $p, q \in \rho$ we say $p < q$ if $o(\rho)$ is closer to p than to q. For a ray $\rho \in \mathcal{R}$ and $g \in \Gamma$ let $\varphi(g, \rho)$ denote the first point of g hit by ρ if it exists; otherwise $\varphi(g, \rho) = +\infty$. We set $\varphi(\Gamma, \rho) = \min_{g \in \Gamma} \varphi(g, \rho)$. We want to build a data structure that, given a ray $\rho \in \mathcal{R}$, computes $\varphi(\Gamma, \rho)$ quickly, together with a $g \in \Gamma$ such that $\varphi(g, \rho) = \varphi(\Gamma, \rho)$. Abusing the notation slightly, we shall use $\varphi(\Gamma, \rho)$ to denote the first intersection point as well as the object that contains the intersection point.

Let $\mathrm{Seg}(\mathcal{R})$ denote the set of all *initial segments* of the rays of $\mathcal{R}$, i.e.,

$$\mathrm{Seg}(\mathcal{R}) = \{ o(\rho)x;\ \ \rho \in \mathcal{R}, x \in \rho \}.$$

Suppose that we have an efficient algorithm that, given a segment $ox \in \mathrm{Seg}(\mathcal{R})$, decides whether it intersects any object $g \in \Gamma$. We refer to this procedure as the *segment emptiness* algorithm. We also assume that the algorithm can detect the case when an initial segment ox intersects Γ only at x and that it can identify the intersected object in this case.

Observe that, given a point x on a ray $\rho \in \mathcal{R}$, we can use the segment emptiness algorithm to decide the relative order of x and $\varphi(\Gamma, \rho)$: if the segment $o(\rho)x$ intersects Γ only at x, then $x = \varphi(\Gamma, \rho)$; if ox is empty then $x < \varphi(\Gamma, \rho)$; otherwise, $x > \varphi(\Gamma, \rho)$. As it is often the case in similar situations, the *parametric search technique* due to Megiddo [26] can be used to turn this "verification" algorithm into a "searching" algorithm. Let us outline this technique applied to our specific problem.

Let A be a segment emptiness algorithm. Let $\mathbf{v}$ be the unit direction vector of ρ and let $\{x(t) = o(\rho) + t\mathbf{v};\ t \in \mathbb{R}^+\}$ be a parametric representation of ρ. Let t^* denote the (yet unknown) value of the parameter t such that $x(t^*) = \varphi(\Gamma, \rho)$. The first idea of the parametric search technique is to run the algorithm A to decide the emptiness of the segment $o(\rho)x(t^*)$, and to run it "generically," without specifying the value of t^*. The execution of the algorithm A will, sooner or later, need some information about t^*. As observed earlier, we can gain some information about t^*: we can compare it with some given t, by running the segment emptiness algorithm on the segment $o(\rho)x(t)$ (for a reason which becomes apparent later, we shall think of this algorithm as another "template" of A and call it B).

Specifically, assume that the flow of execution of A depends on comparisons, each of which involves testing the sign of a low-degree polynomial in t whose coefficients may depend on ρ, on the objects of Γ but not on t^*. We maintain an interval I, which is either a singleton or an open interval that contains t^*. Each time a comparison is to be made, the few roots of the associated polynomial are computed, and we run the algorithm B "off line" at each of them. If one of the roots is t^* itself, we can stop; otherwise we determine the location of t^* among these roots, and thus also the sign of the polynomial at t^*. If we know the two consecutive roots β_i, β_{i+1} such that $t^* \in (\beta_i, \beta_{i+1})$, we can compute the sign of the polynomial at t^*, i.e., the outcome of the comparison at t^*. We now set I to $I \cap (\beta_i, \beta_{i+1})$. If $I = \emptyset$, we can conclude that t^* does not exist, and we stop. Otherwise we resume the execution of the generic algorithm A. As we proceed through this execution, each comparison that we resolve further constrains the range where t^* can lie, and we thus obtain a sequence of progressively smaller intervals, each known to contain t^*, until we either reach the end of A or hit t^* at one of the comparisons at A. Since the outcome of A changes at t^*, it can be shown that t^* is a root of a polynomial associated with one of the comparisons resolved by A (see [3] for a proof). This in turn implies that the computation will stop at the desired value t^*.

The most expensive steps in this computation are calls to the subroutine B for resolving comparisons. To reduce this cost, Megiddo suggests replacing the sequential algorithm A by its parallel version, A_p. If A_p uses p processors and runs in T_A parallel steps, then each such step involves at most p *independent* comparisons, that is, each can be carried out without having to know the outcome of the others. We can then compute the roots of all p polynomials associated with these comparisons, and perform a binary search to locate t^* among them (using the algorithm B at each binary search step). If T_B is the time complexity of the subroutine B, then the binary search requires $O(p + T_B \log p)$ time per parallel step, for a total of $O(pT_A + T_B T_A \log p)$ time for a sequential algorithm that simulates A_p. An improvement of this technique by Cole [15] can further reduce the running time in certain cases by another logarithmic factor (this, however, depends on the specific algorithm A_p).

Let us summarize our discussion in a (rather long) theorem.

THEOREM 2.1. *Let Γ be a set of objects and $\mathcal{R}$ a collection of rays. Suppose that we have a data structure Σ supporting segment emptiness queries with respect to Γ for the segments of* Seg($\mathcal{R}$). *Let A_p be a parallel algorithm for answering a segment emptiness query, which uses p processors and runs in T_A parallel steps*[2], *and such that for a query segment ox, the computation of A_p uses the information about x only in deciding the signs of certain fixed-degree polynomials in the coordinates of x. Let B be another version of segment emptiness algorithm, which can report an object $g \in \Gamma$ intersecting the endpoint of the query segment, provided that the segment is otherwise empty, and let T_B be the running time of B. Then the ray shooting problem for rays in $\mathcal{R}$ can be solved using the same data structure Σ, in time $O(pT_A + T_B T_A \log p)$.*

This approach can obviously be extended to find the first k objects of Γ intersected by the query ray. In this case, the generic algorithm A_p should decide whether the query segment $o(\rho)x(t^*)$ intersects exactly k objects of Γ, and algorithm B decides whether the number of the objects of Γ intersected by query segment are fewer than, equal to, or more than k. After having computed the value of t^*, the answer (the first k objects hit by ρ) can be computed by a segment range reporting algorithm C; usually a variant of A or B gives such an algorithm. The running time of the resulting algorithm will then be $O(pT_A + T_B T_A \log p + T_C)$, where T_A, T_B, and p have the same meaning as above and T_C is the running time of C (T_C may depend on k).

3. Specific results on ray shooting. In this section we apply the general technique described in the previous section to obtain fast solutions for some specific instances. In particular,

[2]Actually we count only the steps with comparisons involving x.

we present ray shooting algorithms in an arrangement of hyperplanes in d-dimensions (§3.1), a collection of half spaces (§3.2), and an arrangement of half planes in $\mathbb{R}^3$ (§3.3).

3.1. Ray shooting among hyperplanes. In this section we describe an efficient ray shooting algorithm for a collection H of n hyperplanes in $\mathbb{R}^d$. For $d \leq 3$, Agarwal and Sharir [4] gave an algorithm that gives a standard trade-off for ray shooting among hyperplanes with $b = d$. In this section we obtain similar bounds for higher dimensions. In view of Theorem 2.1, it suffices to describe an efficient procedure for the segment emptiness problem.

The dual of a hyperplane (respectively, segment) in $\mathbb{R}^d$ is a point (respectively, double wedge),[3] and a segment e intersects a hyperplane h if and only if the double wedge e^* contains the point h^*. Therefore, the segment emptiness problem for H is the same as detecting whether a query double wedge contains any point of H^*. This problem is a special case of the *simplex range searching problem*, where we want to report or count the points contained in a query simplex.

Chazelle, Sharir, and Welzl [12] showed that, using $O(m^{1+\varepsilon})$, $(n \leq m \leq n^d)$ space and preprocessing, we can answer a double wedge range query in time $O(\frac{n}{m^{1/d}} \log^2 n)$ (see Appendix). A simpler algorithm (with improved performance for the case of linear space) was given in [22]. More specific properties of this algorithm will be briefly described in the Appendix. With a knowledge of this data structure, it is straightforward to check that the algorithm can be run in $O(\log n)$ parallel steps using $O(\frac{n}{m^{1/d}} \log n)$ processors. Also, it is shown in [22] that the data structure can be maintained dynamically under insertions and deletions of points. Hence, in view of Theorem 2.1, we can conclude the following theorem.

THEOREM 3.1. *Given a set H of n hyperplanes in $\mathbb{R}^d$, a parameter m, $n \leq m \leq n^d$, we can build, in time $O(m^{1+\varepsilon})$, a data structure of size $O(m^{1+\varepsilon})$ that supports ray shooting queries in time $O(\frac{n}{m^{1/d}} \log^4 n)$. The data structure can be maintained dynamically, as hyperplanes are being inserted in or deleted from H, in $O(m^{1+\varepsilon}/n)$ amortized time per update operation.*

Remark 3.2. With a linear space and $O(n \log n)$ preprocessing time, we actually get query time $O(n^{1-1/d} \log^{O(1)} n)$. On the other hand, with $O(n^{d+\varepsilon})$ space and preprocessing, the query time becomes $O(\log^2 n)$. This and other possibilities of small improvements follow from more specific results of [12], [22].

In the above result, the simplex range searching results were applied only to decide the emptiness of a query segment. However, these algorithms can also count the number of hyperplanes intersecting a query segment within the same time bound, or can report all such k hyperplanes at an additional cost of $O(k)$. The remarks at the end of §2 thus imply the following.

THEOREM 3.3. *Given a set H of n hyperplanes in $\mathbb{R}^d$, a parameter m, $n \leq m \leq n^d$, we can build, in time $O(m^{1+\varepsilon})$, a data structure so that, the first k hyperplanes of H intersected by a query ray can be reported in time $O(\frac{n}{m^{1/d}} \log^4 n + k)$. This data structure can be maintained dynamically, as hyperplanes are being inserted in or deleted from H, in $O(m^{1+\varepsilon}/n)$ amortized time per update operation.*

Agarwal and Sharir [4] showed that if Γ is a set of n segments in $\mathbb{R}^2$ or a set of n triangles in $\mathbb{R}^3$, we can count the number of objects in Γ intersected by a query segment in time $O(n^{1+\varepsilon}/\sqrt{m})$ or $O(n^{16/15+\varepsilon}/m^{4/15})$, respectively, using $O(m^{1+\varepsilon})$ space and preprocessing, where $n \leq m \leq n^2$ and $n \leq m \leq n^4$, respectively. Hence, we obtain similar bounds for reporting the first k objects of Γ hit by a query ray. A further improvement is possible using recent results of Aronov and Sharir [5] on the complexity of a zone of an algebraic surface in a hyperplane arrangement; we will not elaborate on this in this paper.

[3]Throughout this paper, we will denote by γ^* the dual of an object γ, and by Γ^* the set $\{\gamma^*;\ \gamma \in \Gamma\}$.

3.2. Ray shooting in a convex polytope. In this section we consider the ray shooting problem for a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$. We assume that $\mathcal{P}$ is described as the intersection of n half spaces in $\mathbb{R}^d$. Let H denote the set of hyperplanes bounding these half spaces. For $d = 2$ a straightforward binary search can answer a ray shooting query in $O(\log n)$ time, and for $d = 3$ we can use the Dobkin–Kirkpatrick hierarchical representation of a 3 polytope to obtain an optimal algorithm [17]. But for $d > 3$, the Dobkin–Kirkpatrick hierarchical representation does not work, and no efficient algorithm is known for ray shooting in higher dimensions. Even in $\mathbb{R}^3$ no efficient ray shooting algorithm is known if the polytope $\mathcal{P}$ changes dynamically.

We first describe the algorithm for the special case when o, the origin point of the ray, lies inside $\mathcal{P}$. This is simpler and sufficient in many applications. After a suitable projective transformation, we can assume that $\mathcal{P}$ is the region lying above all hyperplanes of H. (Actually, the algorithm can be modified suitably, so that it works even if $\mathcal{P}$ does not lie above all hyperplanes of H.) By Theorem 2.1, we are interested in a data structure that, for a query segment ox, detects whether ox intersects the boundary of $\mathcal{P}$, which, by our assumption, is equivalent to whether there is a hyperplane of H lying above x. In the dual setting, this means that the half space lying above the hyperplane x^* contains at least one point of H^*.

We thus want to preprocess H^* for *half-space emptiness* queries, i.e., we need a data structure deciding whether there is a point of H^* in a query half space. In [23] it is shown that the half-space emptiness queries can be answered in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ using $O(m^{1+\varepsilon})$ space and preprocessing (the algorithm includes the data structure due to Clarkson [14] for the "large space" case). It also allows the insertion or deletion of a half space in $O(m^{1+\varepsilon}/n)$ amortized time, see, e.g., [2] for details. A parallel implementation with $O(\log n)$ time and number of processors bounded by the sequential running time is quite straightforward. Furthermore, the algorithm can also detect the case when the interior of the query half space is empty but its boundary contains a point (see original papers [14], [23] for details). We thus have the following lemma.

LEMMA 3.4. *Given a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$, described as the intersection of n half spaces, and a parameter m, $n \le m \le n^{\lfloor d/2 \rfloor}$, we can construct in time $O(m^{1+\varepsilon})$ a data structure of size $O(m^{1+\varepsilon})$ so that, for a query ray whose origin point lies inside $\mathcal{P}$, we can determine the first point of the polytope boundary hit by the ray in $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$ time. This data structure can be maintained dynamically, as half spaces are being inserted in or deleted from $\mathcal{P}$, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

This result has one surprising consequence. Namely, the half-space emptiness algorithm of [23] sometimes concludes that the query half space is nonempty, but does not exhibit a "witness point" contained in the query half space (which may be required in some applications). Our ray shooting result allows us to obtain such a witness: Let h denote the hyperplane bounding the query half space and, without loss of generality, assume that the query half space lies above h. In the dual setting, we thus want to return a hyperplane of H lying above the point h^* if it exists. We shoot a vertical ray in the $-x_d$ direction from the point $(h_1^*, \dots, h_{d-1}^*, +\infty)$ (where $(h_1^*, \dots, h_d^*)$ is the coordinate vector of h^*). If h^* lies below the first hyperplane hit by the ray, then that hyperplane is the desired witness; otherwise h^* lies above all hyperplanes of H. Hence we have the following corollary.

COROLLARY 3.5. *Given a set S of n points in $\mathbb{R}^d$, we can construct in time $O(m^{1+\varepsilon})$ $(n \le m \le n^{\lfloor d/2 \rfloor})$ a data structure of size $O(m^{1+\varepsilon})$, such that, for a query half space γ, we can determine in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$ a point of $\gamma \cap S$ or conclude that $\gamma \cap S = \emptyset$.*

Next, we extend the ray shooting algorithm for the case when the origin point o of ρ does not lie in $\mathcal{P}$. Note that this does not quite fall into our general framework (at least not if we take the set of hyperplanes for Γ), so we must exhibit a specific (although very similar) solution using parametric search. It suffices to find a point $x^* = x(t^*)$ of the query ray inside $\mathcal{P}$ if it exists (then the previous ray shooting result can be applied); in our setting this means a point x^* of the ray lying above all hyperplanes of H. For this problem, the generic algorithm A will check whether $x(t) \in \mathcal{P}$. The oracle B should decide on which side of a point $x = x(t)$ the potential intersection of the query ray with $\mathcal{P}$ lies. We let B be the algorithm dual to the one from Corollary 3.5, i.e., it checks whether $x \in \mathcal{P}$ (if *yes* the computation may finish), and if not, it exhibits a hyperplane $h \in H$ lying above x. The crucial observation is that at least one of the two portions of the query ray determined by $x(t)$ also lies below h, and therefore $\rho \cap \mathcal{P}$ is bound to lie in the other portion (provided it exists at all). As a result the algorithm B can still resolve comparisons. Clearly, if ρ intersects $\mathcal{P}$, a point in $\rho \cap \mathcal{P}$ will be found. On the other hand, if ρ does not intersect $\mathcal{P}$, the answers given by B will become inconsistent (i.e., the interval for t^* becomes empty). By Corollary 3.5 and Theorem 2.1, we can conclude the following theorem.

THEOREM 3.6. *Given a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$ described as the intersection of n half spaces, and a parameter m ($n \le m \le n^{\lfloor d/2 \rfloor}$), we can preprocess $\mathcal{P}$ in time $O(m^{1+\varepsilon})$ into a data structure of size $O(m^{1+\varepsilon})$, so that for a query ray the first point of $\mathcal{P}$ hit by the ray can be determined in $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^5 n)$ time. The data structure can be maintained dynamically, as half spaces are being inserted in or deleted from $\mathcal{P}$, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

Remark 3.7. This algorithm can also be improved in some special cases. For instance, if we do not require a dynamic structure, we may apply a result of [23]: Given $O(n)$ space and $O(n^{1+\varepsilon})$ preprocessing time, we can answer half space emptiness queries in $O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)})$ time; the algorithm can be implemented with $O(\log\log n)$ parallel steps. Similarly, if we allow $O(n^{\lfloor d/2 \rfloor+\varepsilon})$ space, the query time in the above theorem can be improved to $O(\log^2 n)$.

3.3. Ray shooting among half planes. In this section we consider the case when Γ is a set of half planes in $\mathbb{R}^3$. Although a ray shooting query among planes in $\mathbb{R}^3$ can be answered in roughly $n/m^{1/3}$ time, no such procedure is known for half planes. The query time of the best known algorithm for half spaces is close to $n^{16/15}/m^{4/15}$, which is the same as the query time for ray shooting among triangles in $\mathbb{R}^3$ [4]. Here we improve the query time to $O(\frac{n}{m^{1/3}} \log^{O(1)} n)$; as usual, we will describe an efficient procedure for the segment emptiness problem. We use a "multilevel" range searching structure tailored to this specific application. General principles of building such structures are outlined in the Appendix.

Let H denote the set of planes supporting the half planes of Γ. We construct a partition tree $\mathcal{T}$ on the set of points in H^*. For a node v of $\mathcal{T}$, let Γ_v be the set of half planes corresponding to the points of the canonical subset associated with v, and let L_v be the set of lines bounding the half planes of Γ_v. We orient each line ℓ of L_v so that the half plane bounded by ℓ lies to its right (i.e., in the clockwise direction). L_v is thus a set of oriented lines.

As a secondary data structure in each node v, we construct a data structure for deciding whether a query line ℓ (in our application, the line carrying the query segment) has positive (or negative) orientation with respect to all lines of L_v. Chazelle et al. [10] give a data structure for this problem with space and preprocessing time $O(n^{2+\varepsilon})$ and with $O(\log n)$ query time. They actually reduce the problem to answering a half-space emptiness query in five dimensions[4]. Combining their reduction with the already mentioned results of [23] for

[4]The reduction uses so-called Plucker coordinates of lines. Every line ℓ' of the set L_v is mapped to a Plucker point $\pi(\ell')$ in projective five-space, and a query line ℓ is mapped to a Plucker hyperplane $\varpi(\ell)$ in projective five-space. The query line ℓ has positive (respectively, negative) orientation with respect to all lines in L_v if and only if the hyperplane $\varpi(\ell)$ lies above (respectively, below) the Plucker points of all lines in L_v, see [29], [10] for details. The problem thus reduces to determining whether the half space lying above (respectively, below) $\varpi(\ell)$ is empty, which can be done using the data structure of Clarkson [14].

the half-space emptiness problem, we get a data structure for the segment emptiness problem (with respect to half planes in three-dimensions) that admits a standard trade-off with $b = 2$.

When answering a segment emptiness query for a segment $e = pq$ with this two-level data structure, we first query the first level structure with the double wedge e^* dual to e. It gives the set of half planes, whose supporting planes intersect e, as a pairwise disjoint union of $O(\frac{n}{m^{1/3}} \log n)$ canonical subsets. Let Γ_v be a first-level canonical subset of the query output. Then either p lies below all the planes containing the half planes of Γ_v and q lies above all of them, or vice versa. Without loss of generality, assume that p lies below all of them.

The query segment e intersects a half plane $\gamma \in \Gamma_v$ if and only if ℓ, the line supporting e and oriented from p to q, intersects γ. If the xy-projection of ℓ is in clockwise (respectively, counterclockwise) direction to the xy-projection of the line $\partial\gamma$, then ℓ intersects γ if and only if ℓ lies below (respectively, above) $\partial\gamma$, i.e., ℓ has negative orientation with respect to $\partial\gamma$ (see Fig. 1). In other words, ℓ does not intersect any half plane of Γ_v if and only if ℓ has positive orientation with respect to all lines of L_v. Similarly, if p lies above all the planes containing the half planes of Γ_v, then e does not intersect any half plane of Γ_v if and only if ℓ has negative orientation with respect to all lines of L_v. We can perform this test using the secondary structure stored at v. By repeating this step for all first-level canonical subsets of the query output, we can answer the emptiness query. This data structure gives a standard trade-off with $b = 3$ for the segment emptiness problem. We thus obtain the following result.

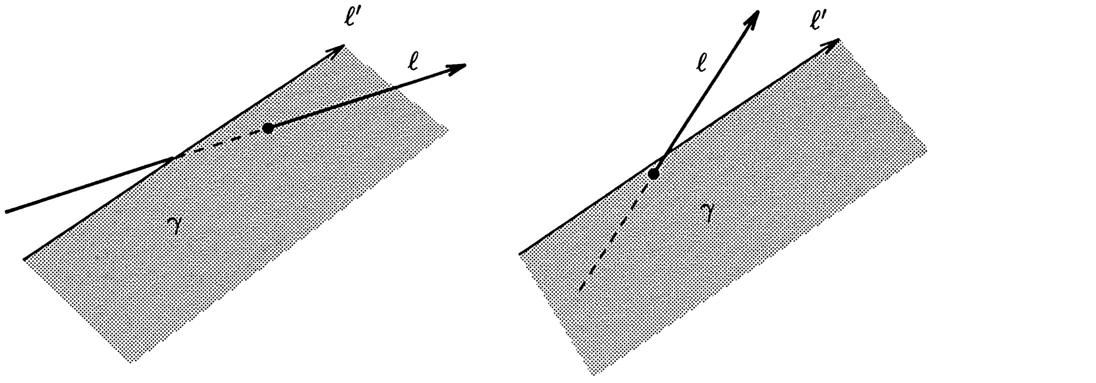


FIG. 1. *Query line ℓ and half plane γ.*

THEOREM 3.8. *Given a set Γ of n half planes in $\mathbb{R}^3$ and a parameter m, $n \le m \le n^3$, we can preprocess Γ, in time $O(m^{1+\varepsilon})$, into a data structure of size $O(m^{1+\varepsilon})$, so that a ray shooting query can be answered in time $O(\frac{n}{m^{1/3}} \log^{O(1)} n)$. The data structure can be maintained dynamically, as half planes are being inserted in or deleted from Γ, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

4. Nearest and farthest neighbors searching. We now divert our attention to the neighbor searching problems: *Given a set S of n points in $\mathbb{R}^d$, store S in a data structure so that, for a query point ξ, we can quickly compute a point of S closest to (or farthest from) ξ.*

Clarkson showed that if we allow $O(n^{\lceil d/2 \rceil + \varepsilon})$ space, we can compute a nearest or farthest neighbor of a query point in S in time $O(\log n)$, but no efficient algorithm is known, if we allow, say, $O(n \log n)$ space. In this section we apply the ray shooting algorithms to search for the closest and farthest neighbors.

We map each point $p = (p_1, p_2, \ldots, p_d)$ of S to the hyperplane $\mathcal{E}(p)$ in $\mathbb{R}^{d+1}$, which is the graph of a d-variate linear function

$$h_p(x_1, x_2, \ldots, x_d) = 2p_1x_1 + 2p_2x_2 + \cdots + 2p_dx_d - (p_1^2 + p_2^2 + \cdots + p_d^2).$$

It is well known that $p \in S$ is a closest neighbor of a point $\xi = (\xi_1, \xi_2, \ldots, \xi_d)$ if and only if

$$h_p(\xi_1, \xi_2, \ldots, \xi_d) = \max_{q \in S} h_q(\xi_1, \xi_2, \ldots, \xi_d)$$

(see [18], [19] for details). The problem of computing a closest neighbor thus reduces to finding a hyperplane in the upper envelope of $\mathcal{E}(S)$ hit by the vertical ray ρ, emanating from the point $(\xi_1, \xi_2, \ldots, \xi_d, +\infty)$ in $-x_{d+1}$ direction. Since the upper envelope of $\mathcal{E}(S)$ is a convex polytope defined as the intersection of n half spaces and the origin point of ρ lies in $\mathcal{E}(S)$, we can use Lemma 3.4. Similarly, a farthest neighbor of ξ in S is a point p if

$$h_p(\xi_1, \xi_2, \ldots, \xi_d) = \min_{q \in S} h_q(\xi_1, \xi_2, \ldots, \xi_d),$$

and the problem of computing a farthest neighbor thus reduces to shooting a vertical ray in the lower envelope of $\mathcal{E}(S)$ from $(\xi_1, \xi_2, \ldots, \xi_d, -\infty)$ (in $+x_{d+1}$ direction). Hence, by Lemma 3.4, we obtain the following result.

THEOREM 4.1. *Given a set S of n points in $\mathbb{R}^d$ and a parameter $n \le m \le n^{\lceil d/2 \rceil}$, we can preprocess S in time $O(m^{1+\varepsilon})$ into a data structure of size $O(m^{1+\varepsilon})$, so that, for a query point ξ, we can compute its closest or farthest neighbor in S in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n)$. Moreover, the data structure can be maintained dynamically, as points are being inserted in or deleted from S, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

We can extend this algorithm to report k nearest or farthest neighbors of a query point. We will restrict ourselves to nearest neighbors; the farthest neighbors can be handled analogously. The k nearest neighbors of ξ are the same as the first k hyperplanes of $\mathcal{E}(S)$ intersected by the vertical ray ρ emanating from $(\xi_1, \xi_2, \ldots, \xi_d, +\infty)$. Therefore, by Theorem 3.3, we can find k nearest neighbors of ξ in time $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$, but we can do better when k is not very large.

By our reductions, it suffices to have an algorithm deciding whether a query point lies below at most k hyperplanes, and also to have a suitable reporting algorithm. A result of [23] in a dual setting shows that, in $\mathbb{R}^d$, all k hyperplanes lying above a query point can be reported in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log n + k)$ using $O(m^{1+\varepsilon})$ space and preprocessing.

Such a reporting algorithm can be turned into an algorithm that checks whether there are at most k hyperplanes above a query point: Given an integer k, let t_k be (an upper bound for) the running time of the reporting algorithm provided that it reports at most k points. One runs the reporting algorithm, terminating it if it does not stop within t_k time units. If it has not stopped by this time, we know that the number of hyperplanes lying above the query hyperplane is larger than k; otherwise we can check the number of reported hyperplanes directly. Hence, we can check in time $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$ whether a query point lies below at least k hyperplanes. A parallel implementation of the reporting algorithm with $O(\log n)$ parallel steps and $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$ processors is again straightforward, and in the checking version, we terminate it when it tries to use more than t_k processors. Plugging these algorithms into the parametric search technique, we obtain a data structure that reports the first k hyperplanes hit by a ray originating in the polytope in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k \log^2 n)$. As usual, the space and preprocessing time of the algorithm are $O(m^{1+\varepsilon})$.

THEOREM 4.2. *Given a set S of n points in $\mathbb{R}^d$, we can preprocess it, in time $O(m^{1+\varepsilon})$, into a data structure of size $O(m^{1+\varepsilon})$ so that, for a query point ξ, we can compute its k closest (or farthest) neighbors in S in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k \log^2 n)$. The data structure can be maintained dynamically, as points are being inserted in or deleted from S, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

5. Hidden surface removal. Consider the following problem: "Given a set T of n triangles in $\mathbb{R}^3$ and a viewpoint p at $z = +\infty$, we want to compute the *visibility map* $\mathcal{M}(T)$ of T, i.e., the subdivision of the viewing plane $z = +\infty$ such that the same triangle of T is visible in $-z$ direction from all points of a face." Note that if we are given an arbitrary viewpoint p, we can apply an appropriate transformation so that p maps to $z = +\infty$.

In the last few years much work has been done on various special cases of the hidden surface removal problem, see [27], [4], [8], [7]. Most of these algorithms apply efficient ray shooting procedures to compute the visibility map. Recently, de Berg et al. [7] gave an $O(n^{1+\varepsilon}\sqrt{k}+k)$ algorithm for the general case, where k is the size of the visibility map. Their algorithm uses a ray shooting procedure for a collection of curtains as a subroutine; a *curtain* is an unbounded vertical triangle with two of its edges being vertical rays extending to $-\infty$. The overall running time of their algorithm is governed by the time complexity (query time plus preprocessing) of the ray shooting procedure. In this section we will present a faster solution for the ray shooting among a family of curtains, which in turn will improve the time complexity of their hidden surface removal algorithm.

Let Γ be a collection of curtains in $\mathbb{R}^3$. As usual, we describe a data structure for the segment emptiness problem for Γ. Let ℓ denote the line containing e, ℓ_γ the line containing the bounded segment of a curtain γ, and $\bar{\delta}$ the xy-projection of an object δ. The segment e intersects a curtain γ if and only if (i) $\bar{e}$ intersects $\bar{\gamma}$, and (ii) ℓ lies below ℓ_γ. Let $\bar{\Gamma} = \{\bar{\gamma} \mid \gamma \in \Gamma\}$ denote a set of n segments in $\mathbb{R}^2$. It is known (see, e.g., [4]) that $\bar{e}$ intersects $\bar{\gamma}$ if and only if $\bar{\ell}$ separates the endpoints of $\bar{\gamma}$ and $\bar{e}$ intersects $\bar{\ell}_\gamma$. We construct a four-level data structure (similar to the one used in [4]; see Appendix for details) on $\bar{\Gamma}$.

We first construct a partition tree on the left endpoints of the segments of $\bar{\Gamma}$ and then, for each node z of the tree, we construct a secondary partition tree on the right endpoints of the segments of $\bar{\Gamma}$ whose left endpoints are in the canonical subset associated with z. Next, for each second-level node we extend the segments of $\bar{\Gamma}$, whose right endpoints are stored at that node, to full lines in the plane and dualize them to points in $\mathbb{R}^2$. We construct a third-level partition tree on these points. For a third-level node v, let $\bar{\Gamma}_v$ (respectively, Γ_v) be the set of segments (respectively, curtains) corresponding to the points associated with v. Finally, at each third level node v, we store a data structure that, for a query line ℓ (the one extending a query segment e), decides whether it lies above all lines of $\{\ell_\gamma;\ \gamma \in \Gamma_v\}$ (see §3.3).

To detect an intersection between e and the curtains of Γ, we search the first two levels of the above structure with $\bar{\ell}$, the third level structure with the double-wedge dual to $\bar{e}$, and finally the fourth level structure with the Plücker hyperplane of ℓ (see the Appendix for details). If the tests come out positive for the fourth-level structures of every third-level node, filtered out by the previous levels of the partition tree, our query segment is empty. Putting all this together, we get a standard trade-off for the segment emptiness problem among curtains with $b = 2$.

Plugging the resulting ray shooting procedure into the hidden surface removal algorithm of [7] and choosing $m = \lceil n^{2/3}k^{2/3}\rceil$, we can compute the visibility map in time $O(n^{2/3+\varepsilon}k^{2/3} + n^{1+\varepsilon})$. Note that we do not know the value of k in advance, therefore we must guess an initial value of m and update it as the algorithm proceeds, as in [7], [27]; see any of these papers for details. Hence, we can conclude the following theorem.

THEOREM 5.1. *The visibility map of a given set of n triangles in $\mathbb{R}^3$ with respect to a viewing point can be computed in time $O(n^{2/3+\varepsilon}k^{2/3} + n^{1+\varepsilon})$, where k is the output size.*

6. Conclusion. In this paper we have applied the parametric search technique to obtain efficient procedures for the ray shooting and related problems. Recently, further progress was made in the construction of (static) range searching algorithms (see [24] for simplex range searching algorithms and multilevel range searching structures and [28] for half-space range reporting with polylogarithmic query time). As a result, the n^ε factors appearing in the complexity bounds for our static ray shooting algorithms can be replaced by polylogarithmic factors. Also, in the case of ray shooting in a convex polytope, we can remove some of the extra logarithmic factors in the query time arising by a straightforward application of the parametric search method; see [25], [28].

Finally, we conclude by mentioning some open problems.

(i) Can the ray shooting algorithm for half planes be extended to triangles in $\mathbb{R}^3$? The currently best known algorithm admits the standard trade-off with $b = 4$.

(ii) What about ray shooting in a collection of spheres or other nonlinear objects in $\mathbb{R}^3$? The authors are not aware of any efficient procedure for ray shooting or even intersection detection algorithms for nonlinear objects.

(iii) Can the running time of k-nearest neighbor searching be improved to

$$O\left(\frac{n}{m^{1/\lceil d/2\rceil}} \log^3 n + k\right)?$$

Appendix. Multilevel partition trees. In some of our ray shooting problems (e.g., §§3.3 and 5), we needed "tailored" range searching structures for the corresponding segment emptiness problems. These particular data structures are not explicitly described in the literature, but they can be easily designed using *multilevel* (*recursive*) *partition trees*. Multilevel partition trees were originally introduced by Dobkin and Edelsbrunner [16] and they have been successfully applied to several geometric problems [12], [22], [4].

Let us start by describing a partition tree for the half-space range searching. We will consider only some abstract properties without going into specific implementation details. Thus our description encompasses the partition scheme of [12], as well as a somewhat simpler and better partition scheme of [22].

Let S be a set of n points in $\mathbb{R}^d$. A partition tree on S consists of a collection of certain distinguished subsets of S, called the *canonical subsets*. The task of a partition tree is to express the intersection of S with a given half space γ as a disjoint union of canonical subsets; such a decomposition of $S \cap \gamma$ is called a *canonical decomposition*. A partition tree is implemented as a tree-like data structure, each of whose node is associated with canonical subsets. At each node, we also store certain auxiliary information that allows us to find a canonical decomposition of $S \cap \gamma$ efficiently.

In order to answer the queries efficiently, we need a worst-case upper bound on the maximum number of canonical subsets used in a canonical decomposition of $S \cap \gamma$. On the other hand, the sum of sizes of all canonical subsets essentially determines the space requirements of the partition tree. The results of [12], [22] show that, for any $n \leq m \leq n^d$, we can build a partition tree of size (i.e., the sum of the size of its canonical subsets is) $O(m^{1+\varepsilon})$, which can decompose $S \cap \gamma$ into $O(\frac{n}{m^{1/d}} \log n)$ canonical subsets, for any query half space γ.

Depending on the purpose of a partition tree, we also store some kind of additional information along with each canonical subset. For example, we need to store the cardinality of each canonical subset if we want to use it for answering queries of the form "how many points of S are contained in a query half space."

To introduce the concept of a multilevel partition tree, let us consider a slightly more complicated example, where we want to count the number of points of S in a query wedge, i.e., in an intersection of two half-spaces[5]. We thus take every canonical subset C in our partition tree, and build a "secondary" partition tree for C; this data structure corresponds to the auxiliary information associated with C in the "primary" partition tree. The auxiliary information stored at each node of the secondary partition tree is still the cardinality of the

[5]The partition trees of [22] allow us to handle such a query directly, since the points of S in every *simplex* can be decomposed into the appropriate number of canonical subsets, but we want to illustrate the principle here.

corresponding secondary canonical subset. Given a wedge $w = \gamma \cap \gamma'$, where γ, γ' are two half-spaces, we first obtain a canonical decomposition of $S \cap \gamma$ with respect to the primary tree. Then, for every canonical subset C in this decomposition, we use the corresponding secondary data structure to count the number of points in $C \cap \gamma'$. Finally we sum up all the answers to compute the value of $|S \cap w|$. Hence, the first level of our data structure sifts out the points contained in γ, and the second level sifts the points contained in γ'. Thus, the two levels together filter out the points of S that lie in $w = \gamma \cap \gamma'$.

We need not limit ourselves to just two levels; for instance, we need $d+1$ levels to handle simplex range counting queries in this spirit. The examples described in this paper illustrate that the secondary structure need not deal with the points of canonical subsets directly, but rather it can be set up for some objects in one-to-one correspondence with these points (e.g., in §5, the primary partition tree is constructed on the left endpoints of some set of segments, the secondary partition trees are constructed on right endpoints of the appropriate segments, and the tertiary partition trees are constructed on the points dual to some set of lines).

In order to analyze the performance of such multilevel structures, we apparently need more information about the distribution of sizes of the canonical subsets. From the analysis in [12], [22], we can obtain the following convenient form.

THEOREM A.1. *For an n-point set in $\mathbb{R}^d$ and a parameter m, $n \le m \le n^d$, we can construct, in time $O(m^{1+\varepsilon})$, a partition tree that contains at most $O(\max\{\frac{n}{s}, \frac{m^{1+\varepsilon}}{s^d}\})$ canonical subsets of size exceeding s for any $s \le n$ and, for any half-space γ, the canonical decomposition of $\gamma \cap S$ contains at most*

$$O\left(\min\left\{\frac{n^{1-1/d+\varepsilon}}{s^{1-1/d}}, \frac{n}{m^{1/d}}\log n\right\}\right)$$

canonical subsets of size exceeding s.

Using these estimates, the analysis of multilevel partition trees becomes straightforward, especially if we ignore $O(n^\varepsilon)$ factors. Let us consider the above outlined two-level example for wedge range counting. For the sake of simplicity, let us assume that $m = n$ for the primary partition tree, and that, for every canonical subset C of size c, we build a secondary partition tree with $m = c$. The secondary structure constructed on C requires $O(c^{1+\varepsilon})$ space. The total space required by this two-level structure is thus at most

$$\sum_{i=0}^{\log_2 n} 2^{i(1+\varepsilon)} O\left(\max\left\{\frac{n}{2^i}, \frac{n^{1+\varepsilon}}{2^{di}}\right\}\right) = O(n^{1+\varepsilon}),$$

and the number of second-level canonical subsets used for decomposition of a wedge is at most

$$\sum_{i=0}^{\log_2 n} 2^{i(1-1/d)} \log n \cdot O\left(\min\left\{\frac{n^{1-1/d+\varepsilon}}{2^{i(1-1/d)}}, n^{1-1/d}\log n\right\}\right) = O(n^{1-1/d}\log^2 n).$$

As for other values of m, a similar calculation shows that the total space is $O(m^{1+\varepsilon})$ and a query output consists of $O(\frac{n}{m^{1/d}}\log^2 n)$ canonical subsets. In general, the space requirements and the query complexity of a multilevel partition tree are essentially determined by the least efficient level. The basic analysis of this kind is given in [4], [12], and finer complexity questions (the tuning of subpolynomial factors) are touched in [22].

Acknowledgments. The authors thank Marco Pellegrini for pointing out an error in an earlier version of the paper, Micha Sharir for several helpful discussions, and the referees for useful comments.

REFERENCES

[1] P.K. AGARWAL, *Ray shooting and other applications of spanning trees with low stabbing number*, SIAM J. Comput., 21 (1992), pp. 540–570.

[2] P. K. AGARWAL AND J. MATOUŠEK, *Dynamic half-space range reporting and its applications*, Tech. Report CS-1991-43, Dept. of Computer Science, Duke University, Durham, NC, 1991; Algorithmica, to appear.

[3] P. K. AGARWAL AND M. SHARIR, *Planar geometric location problems*, Tech. Report 90-58, DIMACS, Rutgers University, August, 1990. (Also to appear in *Algorithmica.*)

[4] ———, *Applications of a new partitioning scheme*, Discrete Comput. Geom., 9 (1993), pp. 11–38.

[5] B. ARONOV, M. PELLEGRINI, AND M. SHARIR, *On the zone of a surface in a hyperplane arrangement*, Discrete Comput. Geom., to appear.

[6] R. BAR YEHUDA AND S. FOGEL, *Good splitters with applications to ray shooting*, in Proc. 2nd Canadian Conf. on Computational Geometry, 1990, pp. 81–85.

[7] M. DE BERG, D. HALPERIN, M. OVERMARS, J. SNOEYINK, AND M. VAN KREVELD, *Efficient ray shooting and hidden surface removal*, in Proc. 7th Symposium on Computational Geometry, 1991, pp. 51–60.

[8] M. DE BERG AND M. OVERMARS, *Hidden surface removal for axis-parallel polyhedra*, Comput. Geom.: Theory and Appl., 1 (1992), pp. 247–268.

[9] B. CHAZELLE, H. EDELSBRUNNER, M. GRIGNI, L. GUIBAS, J. HERSHBERGER, M. SHARIR, AND J. SNOEYINK, *Ray shooting in polygons using geodesic triangulations*, Proc. 17th International Colloquium on Automata, Languages and Programming, 1991, pp. 661–673.

[10] B. CHAZELLE, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND J. STOLFI, *Lines in space: Combinatorics and algorithms*, Proc. 21st ACM Symposium on Theory of Computing, 1989, pp. 382–393. Full version: Tech. Report 491, Dept. of Computer Science, New York University, New York, February, 1990.

[11] B. CHAZELLE AND L. GUIBAS, *Visibility and intersection problems in plane geometry*, Discrete Comput. Geom., 4 (1989), pp. 551–589.

[12] B. CHAZELLE, M. SHARIR, AND E. WELZL, *Quasi-optimal upper bounds for simplex range searching and new zone theorems*, Algorithmica, 8 (1992), pp. 407–430.

[13] S.W. CHENG AND R. JANARDAN, *Space efficient ray shooting and intersection searching: Algorithms, dynamization, and applications*, J. Algorithms, 3 (1992), pp. 670–692.

[14] K. CLARKSON, *A randomized algorithm for closest point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.

[15] R. COLE, *Slowing down sorting networks to obtain faster sorting algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 200–208.

[16] D. DOBKIN AND H. EDELSBRUNNER, *Space searching for intersecting objects*, J. Algorithms, 8 (1987), pp. 348–361.

[17] D. DOBKIN AND D. KIRKPATRICK, *Determining the separation of preprocessed polyhedra: a unified approach*, Proc. 17th International Colloquium on Automata, Languages and Programming, 1990, pp. 400–413.

[18] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, New York, 1987.

[19] H. EDELSBRUNNER AND R. SEIDEL, *Voronoi diagrams and arrangements*, Discrete Comput. Geom., 1 (1985), pp. 25–44.

[20] L. GUIBAS, M. OVERMARS, AND M. SHARIR, *Ray shooting, implicit point location, and related queries in arrangements of segments*, Tech. Report 433, Courant Institute, New York University, New York, 1989.

[21] J. MATOUŠEK, *Approximations and optimal geometric divide-and-conquer*, Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 506–511.

[22] ———, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.

[23] ———, *Reporting points in halfspaces*, Comput. Geom.: Theory and Appl., 2 (1992), pp. 169–186.

[24] ———, *Range searching with efficient hierarchical cuttings*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 276–285.

[25] J. MATOUŠEK AND O. SCHWARZKOPF, *Linear optimization queries*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 16–25.

[26] N. MEGIDDO, *Applying parallel computation algorithms in the design of serial algorithms*, J. Assoc. Comput. Mach., 30 (1983), pp. 852–865.

[27] M. OVERMARS AND M. SHARIR, *Output-sensitive hidden surface removal*, Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 598–603.

[28] O. SCHWARZKOPF, *Ray shooting in convex polytopes*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 286–295.

[29] D. SOMMERVILLE, *Analytical Geometry in Three Dimensions*, Cambridge University Press, London, 1951.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 807–837, August 1993

LEARNING IN THE PRESENCE OF MALICIOUS ERRORS*

MICHAEL KEARNS† AND MING LI‡

Abstract. In this paper an extension of the distribution-free model of learning introduced by Valiant [*Comm. ACM*, 27(1984), pp. 1134–1142] that allows the presence of malicious errors in the examples given to a learning algorithm is studied. Such errors are generated by an adversary with unbounded computational power and access to the entire history of the learning algorithm's computation. Thus, a worst-case model of errors is studied.

The results of this research include general methods for bounding the rate of error tolerable by any learning algorithm, efficient algorithms tolerating nontrivial rates of malicious errors, and equivalences between problems of learning with errors and standard combinatorial optimization problems.

1. Introduction. In this paper, we study a practical extension to Valiant's distribution-free model of learning: the presence of errors (possibly maliciously generated by an adversary) in the sample data. The distribution-free model typically makes the idealized assumption that the oracles *POS* and *NEG* (returning positive and negative examples of the unknown target concept) always faithfully return untainted examples of the target representation drawn according to the target distributions. In many environments, however, there is always some chance that an erroneous example is given to the learning algorithm. In a training session for an expert system, this might be due to an occasionally faulty teacher; in settings where the examples are being transmitted electronically, it might be due to unreliable communication equipment.

Since one of the strengths of Valiant's model is the lack of assumptions on the probability distributions from which examples are drawn, we seek to preserve this generality by making no assumptions on the *nature* of the errors that occur. That is, we wish to avoid demanding algorithms that work under any target distributions while at the same time assuming that the errors in the examples have some "nice" form. Thus, we study a worst-case or *malicious* model of errors, in which the errors are generated by an adversary whose goal is to foil the learning algorithm.

The study of learning from examples with malicious errors was initiated by Valiant [24], where it is assumed that there is a fixed probability β of an error occurring independently on each request for an example. This error may be of an arbitrary nature—in particular, it may be chosen by an adversary with unbounded computational resources and exact knowledge of the target representation, the target distributions, and the current internal state of the learning algorithm.

In this paper we study the *optimal malicious error rate* $E_{MAL}(C)$ for a representation class C—that is, the largest value of β that can be tolerated by any learning algorithm (not necessarily polynomial time) for C. Note that we expect the optimal error rate to depend on ϵ and δ (and n in the case of a parameterized target class C). An upper bound on $E_{MAL}(C)$ corresponds to a hardness result placing limitations on the rate of error that can be tolerated;

*Received by the editors July 2, 1990; accepted for publication (in revised form) March 10, 1992. A preliminary version of this research appears in the *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing*.

†AT&T Bell Laboratories, 600 Mountain Ave., Room 2A-423, P.O. Box 636, Murray Hill, New Jersey 07974-0636. This research was done while the author was a graduate student at Harvard University and visiting the University of California at Santa Cruz. This research was supported by an AT&T Bell Laboratories scholarship and grants N00014-85-K-0445 and N00014-86-K-0454 from the Office of Naval Research and DCR-8606366 from the National Science Foundation.

‡Department of Computer Science, University of Waterloo, Davis Center, Room 2339, Waterloo, Ontario N2L 3G1, Canada. This research was done while the author was at Harvard University. This research was supported by grants N00014-85-K-0445 from the Office of Naval Research and DCR-8606366 from the National Science Foundation.

lower bounds on $E_{MAL}(C)$ are obtained by giving algorithms that tolerate a certain rate of error.

Using a proof technique called the method of induced distributions, we obtain general upper bounds on $E_{MAL}(C)$ and apply these results to many representation classes. We also obtain lower bounds on $E_{MAL}^{poly}(C)$ (the largest rate of malicious error tolerated by a polynomial-time learning algorithm for C) by giving efficient learning algorithms for these same classes and analyzing their error tolerance. In several cases the upper and lower bounds on $E_{MAL}^{poly}(C)$ meet. A canonical method of transforming standard learning algorithms into error-tolerant algorithms is given, and we give approximation-preserving reductions between standard combinatorial optimization problems such as set cover and natural problems of learning with errors. Several of our results also apply to a more benign model of *classification noise* defined by Angluin and Laird [1], in which the underlying target distributions are unaltered, but there is some probability that a positive example is incorrectly classified as being negative and vice versa.

Several themes are brought out. One is that error tolerance need not come at the expense of efficiency or simplicity. We show that there are representation classes for which the optimal malicious error rate can be achieved by algorithms that run in polynomial time and are easily coded. For example, we show that a polynomial-time algorithm for learning monomials with errors due to Valiant [24] tolerates the largest malicious error rate possible for any algorithm, polynomial-time or otherwise, that uses only positive examples. We give an efficient learning algorithm for the class of symmetric functions that tolerates the optimal malicious error rate and uses an optimal number of examples.

Another theme is the importance of using both positive and negative examples whenever errors (either malicious errors or classification noise errors) are present. Several existing learning algorithms use only positive examples or only negative examples (see, e.g., Valiant [23] and Blumer et al. [5]). We demonstrate strong upper bounds on the tolerable error rate when only one type is used and show that this rate can be provably increased when both types are used. In addition to proving this for the class of symmetric functions, we give an efficient algorithm that provides a provable increase in the malicious error rate over the positive-only algorithm of Valiant [24] for the class of monomials.

A third theme is that there are strong ties between learning with errors and more traditional problems in combinatorial optimization. We give a reduction from learning monomials with errors to a generalization of the weighted set cover problem and an approximation algorithm for this problem (generalizing the greedy algorithm analyzed by several authors [7], [11], [18]) that is of independent interest. This approximation algorithm is used as a subroutine in a learning algorithm that tolerates an improved error rate for monomials. In the other direction, we prove that for the class of monomials M, approaching the optimal error rate $E_{MAL}(M)$ with a polynomial-time algorithm using hypothesis space M is at least as hard as finding an efficient approximation algorithm with an improved performance guarantee for the set cover problem. This suggests that there are classes for which the optimal error rate that can be tolerated *efficiently* may be considerably smaller than the optimal *information-theoretic* rate. The best approximation known for the set cover problem remains the greedy algorithm analyzed by Chvatal [7], Johnson [11], Lovasz [17], and Nigmatullin [18]. Finally, we give a canonical reduction that allows many learning with errors problems to be studied as equivalent optimization problems, thus allowing one to sidestep some of the difficulties of analysis in the distribution-free model. Similar results are given for the error-free model by Haussler et al. [10].

We now give a brief survey of other studies of error in the distribution-free model. Valiant [24] modified his initial definitions of learnability to include the presence of errors in the examples. He also gave a generalization of his algorithm for learning monomials from positive examples, and analyzed the rate of malicious error tolerated by this algorithm. Valiant's results led him to suggest the possibility that "the learning phenomenon is only feasible with very low error rates" (at least in the distribution-free setting with malicious errors); some of the results presented in this paper can be viewed as giving formal verification of this intuition. On the other hand, some of our algorithms provide hope that if one can somehow reliably control the *rate* of error to a small amount, then errors of an arbitrary nature can be compensated for by the learning process.

Angluin and Laird [1] subsequently modified Valiant's definitions to study a nonmalicious model of errors, defined in §2.3 as the *classification noise model.* Their results demonstrate that under stronger assumptions on the nature of the errors, large rates of error can be tolerated by polynomial-time algorithms for nontrivial representation classes. Shackelford and Volper [21] investigate a model of random noise in the instances rather than the labels, and Sloan [22] and Laird [15] discuss a number of variants of both the malicious error and classification noise models.

2. Definitions for distribution-free learning.

In this section we give definitions and motivation for the model of machine learning we study. This model was first defined by Valiant [23] in 1984; he then went on to generalize his definitions to allow errors in 1985 [24]. In addition to the basic definitions and notation, we give the form of Chernoff bounds we use, define the Vapnik–Chervonenkis dimension, and define a number of classes of representations whose error-tolerant learnability we will study.

2.1. Representing subsets of a domain.

Concept classes and their representation. Let X be a set called a *domain* (also sometimes referred to as the *instance space*). We think of X as containing encodings of all objects of interest to us in our learning problem. For example, each instance in X may represent a different object in a particular room, with discrete attributes representing properties such as color, and continuous values representing properties such as height. The goal of a learning algorithm is then to infer some unknown subset of X, called a *concept*, chosen from a known *concept class.*

For computational purposes we always need a way of *naming* or *representing* concepts. Thus, we formally define a *representation class over* X to be a pair (σ, C), where $C \subseteq \{0, 1\}^*$ and σ is a mapping $\sigma : C \to 2^X$ (here 2^X denotes the power set of X). For $c \in C$, $\sigma(c)$ is called a *concept* over X; the image space $\sigma(C)$ is the *concept class* that is *represented* by (σ, C). For $c \in C$, we define $pos(c) = \sigma(c)$ (the *positive examples* of c) and $neg(c) = X - \sigma(c)$ (the *negative examples* of c). The domain X and the mapping σ will usually be clear from the context, and we will simply refer to the *representation class* C. We will sometimes use the notation $c(x)$ to denote the value of the characteristic function of $\sigma(c)$ on the domain point x; thus $x \in pos(c)(x \in neg(c)$, respectively) and $c(x) = 1(c(x) = 0$, respectively) are used interchangeably. We assume that domain points $x \in X$ and representations $c \in C$ are efficiently encoded using any of the standard schemes (see [9]) and denote by $|x|$ and $|c|$ the length of these encodings measured in bits.

Parameterized representation classes. In this paper we will study *parameterized* classes of representations. Here we have a stratified domain $X = \cup_{n\geq 1} X_n$ and representation class $C = \cup_{n\geq 1} C_n$. The parameter n can be regarded as an appropriate measure of the complexity of concepts in $\sigma(C)$ (such as the number of domain attributes), and we assume that for a representation $c \in C_n$ we have $pos(c) \subseteq X_n$ and $neg(c) = X_n - pos(c)$. For example, X_n may be the set $\{0, 1\}^n$, and C_n the class of all Boolean formulae over n variables whose length is at most n^2. Then for $c \in C_n, \sigma(c)$ would contain all satisfying assignments of the formula c.

Efficient evaluation of representations. In general, we will be primarily concerned with learning algorithms that are computationally efficient. In order to prevent this demand from being vacuous, we need to ensure that the *hypotheses* output by a learning algorithm can be efficiently evaluated as well. Thus if C is a representation class over X, we say that C is *polynomially evaluatable* if there is a (probabilistic) polynomial-time *evaluation algorithm* A that on input a representation $c \in C$ and a domain point $x \in X$ outputs $c(x)$. All representation classes considered here are polynomially evaluatable. It is worth mentioning at this point that Schapire [20] has shown that if a representation class is not polynomially evaluatable, then it is not efficiently learnable in our model. Thus, perhaps not surprisingly we see that classes that are not polynomially evaluatable constitute "unfair" learning problems.

Samples. A *labeled example* from a domain X is a pair $\langle x, b\rangle$, where $x \in X$ and $b \in \{0, 1\}$. A *labeled sample* $S = \langle x_1, b_1\rangle, \ldots, \langle x_m, b_m\rangle$ from X is a finite sequence of labeled examples from X. If C is a representation class, a *labeled example of* $c \in C$ is a labeled example $\langle x, c(x)\rangle$, where $x \in X$. A *labeled sample of* c is a labeled sample S where each example of S is a labeled example of c. In the case where all labels b_i or $c(x_i)$ are 1(0, respectively), we may omit the labels and simply write S as a list of points $x_1, \ldots, x_m$, and we call the sample a *positive* (*negative*, respectively) sample.

We say that a representation h and an example $\langle x, b\rangle$ *agree* if $h(x) = b$; otherwise they *disagree*. We say that a representation h and a sample S are *consistent* if h agrees with each example in S; otherwise they are *inconsistent*.

2.2. Distribution-free learning.

Distributions on examples. On any given execution, a learning algorithm for a representation class C will be receiving examples of a single distinguished representation $c \in C$. We call this distinguished c the *target representation*. Examples of the target representation are generated probabilistically as follows: Let D_c^+ be a fixed but arbitrary probability distribution over $pos(c)$, and let D_c^- be a fixed but arbitrary probability distribution over $neg(c)$. We call these distributions the *target distributions*. When learning c, learning algorithms will be given access to two oracles, *POS* and *NEG*, that behave as follows: oracle *POS* (*NEG*, respectively) returns in unit time a positive (negative, respectively) example of the target representation, drawn randomly according to the target distribution D_c^+ (D_c^-, respectively).

The distribution-free model is sometimes defined in the literature with a single target distribution over the entire domain; the learning algorithm is then given labeled examples of the target concept drawn from this distribution. We choose to explicitly separate the distributions over the positive and negative examples to facilitate the study of algorithms that learn using only positive examples or only negative examples. These models, however, are equivalent with respect to polynomial-time computation, as is shown by Haussler et al. [10].

Given a fixed target representation $c \in C$, and given fixed target distributions D_c^+ and D_c^-, there is a natural measure of the *error* (with respect to c, D_c^+, and D_c^-) of a representation h from a representation class H. We define $e_c^+(h) = D_c^+(neg(h))$ (i.e., the weight of the set $neg(h)$ under the probability distribution D_c^+) and $e_c^-(h) = D_c^-(pos(h))$ (the weight of the set $pos(h)$ under the probability distribution D_c^-). Note that $e_c^+(h)$ (respectively, $e_c^-(h)$) is simply the probability that a random positive (respectively, negative) example of c is identified as negative (respectively, positive) by h. If both $e_c^+(h) < \epsilon$ and $e_c^-(h) < \epsilon$, then we say that h is an ϵ*-good* hypothesis (with respect to c, D_c^+, and D_c^-); otherwise, h is ϵ*-bad*. We define the *accuracy* of h to be the value $\min(1 - e_c^+(h), 1 - e_c^-(h))$.

It is worth noting that our definitions so far assume that the hypothesis h is deterministic. However, this need not be the case; for example, we can instead define $e_c^+(h)$ to be the probability that h classifies a random positive example of c as negative, where the probability is now over both the random example and the coin flips of h. All of the results presented here hold under these generalized definitions.

When the target representation c is clear from the context, we will drop the subscript c and simply write D^+, D^-, e^+, and e^-.

In the definitions that follow, we will demand that a learning algorithm produce with high probability an ϵ-good hypothesis regardless of the target representation and target distributions. Although at first this may seem like a strong criterion, note that the error of the hypothesis output is always measured with respect to the same target distributions on which the algorithm was trained. Thus, while it is true that certain examples of the target representation may be extremely unlikely to be generated in the training process, these same examples intuitively may be "ignored" by the hypothesis of the learning algorithm, since they contribute a negligible amount of error.

Learnability. Let C and H be representation classes over X. Then C is *learnable from examples by H* if there is a (probabilistic) algorithm A with access to *POS* and *NEG*, taking inputs ϵ, δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any inputs $0 < \epsilon, \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^-(h_A) < \epsilon$.

We call C the *target class* and H the *hypothesis class*; the output $h_A \in H$ is called the *hypothesis* of A. A will be called a *learning algorithm* for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $1/\epsilon$, $1/\delta$, and $|c|$, then we say that C is *polynomially learnable from examples by H*; if C is parameterized, we also allow the running time of A to have polynomial dependence on the parameter n.

We will drop the phrase "from examples" and simply say that C is *learnable by H*, and C is *polynomially learnable by H*. We say C is *polynomially learnable* to mean that C is polynomially learnable by H for some polynomially evaluatable H. We will sometimes call ϵ the *accuracy parameter* and δ the *confidence parameter*.

Thus, we ask that for any target representation and any target distributions, a learning algorithm finds an ϵ-good hypothesis with probability at least $1-\delta$. A primary goal of research in this model is to discover which representation classes C are polynomially learnable.

We refer to Valiant's model as the *distribution-free* model to emphasize that we seek algorithms that work for any target distributions. It is also known in the literature as the *probably approximately correct* model.

Positive-only and negative-only learning algorithms. We will sometimes study learning algorithms that need only positive examples or only negative examples. If A is a learning algorithm for a representation class C, and A makes no calls to the oracle NEG (respectively, *POS*), then we say that A is a *positive-only* (respectively, *negative-only*) learning algorithm, and C is *learnable from positive examples* (*respectively, learnable from negative examples*). Note that although the learning algorithm receives only one type of examples, the hypothesis output must still be accurate with respect to *both* the positive and negative distributions.

Several learning algorithms in the distribution-free model are positive-only or negative-only. The study of positive-only and negative-only learning is interesting for at least two reasons. First, it helps to quantify more precisely what kind of information is required for learning various representation classes. Second, it may be important for situations where, for instance, negative examples are rare but must be classified accurately when they do occur.

2.3. Definitions for learning with errors.

Oracles with malicious errors. Let C be a representation class over a domain X, and let $c \in C$ be the target representation with target distributions D^+ and D^-. For $0 \leq \beta < \frac{1}{2}$, we define two *oracles with malicious errors*, POS^{β}_{MAL} and NEG^{β}_{MAL}, that behave as follows: When oracle POS^{β}_{MAL} (respectively, NEG^{β}_{MAL}) is called, with probability $1 - \beta$, a point $x \in pos(c)$ (respectively, $x \in neg(c)$) randomly chosen according to D^+ (respectively, D^-) is returned, as in the error-free model, but with probability β, a point $x \in X$ about which absolutely no assumptions can be made is returned. In particular, this point may be dynamically and maliciously chosen by an adversary who has knowledge of c, D^+, D^-, β, and the internal state of the learning algorithm. This adversary also has unbounded computational resources. For convenience we assume that the adversary does not have knowledge of the outcome of future coin flips of the learning algorithm or the points to be returned in future calls to POS_{MAL}^{β} and NEG_{MAL}^{β} (other than those that the adversary may decide to generate on future errors). These assumptions may in fact be removed, as our results will show, resulting in a stronger model where the adversary may choose to modify in any manner a fixed fraction β of the sample to be given to the learning algorithm. Such a model realistically captures situations such as "error bursts," which may occur when transmission equipment malfunctions repeatedly for a short amount of time.

Learning from oracles with malicious errors. Let C and H be representation classes over X. Then for $0 \leq \beta < \frac{1}{2}$, we say that C is *learnable by H with malicious error rate β* if there is a (probabilistic) algorithm A with access to POS_{MAL}^{β} and NEG_{MAL}^{β}, taking inputs ϵ, δ, and β_0, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input values $0 < \epsilon, \delta < 1$ and $\beta \leq \beta_0 < \frac{1}{2}$, algorithm A halts and outputs a representation $h_A \in H$ that with probability at least $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$.

We will also say that A is a *β-tolerant learning algorithm for C*. In this definition of learning, polynomial-time means polynomial in $1/\epsilon$, $1/\delta$, and $1/(\frac{1}{2}-\beta_0)$, as well as polynomial in n in the case of parameterized C.

The input β_0 is intended to provide an upper bound on the error rate for the learning algorithm, since in practice we do not expect to have exact knowledge of the "true" error rate β (for instance, it is reasonable to expect the error rate to vary somewhat with time). The dependence on $1/(\frac{1}{2}-\beta_0)$ for polynomial-time algorithms provides the learning algorithm with more time as the error rate approaches $\frac{1}{2}$, since an error rate of $\frac{1}{2}$ renders learning impossible for any algorithm, polynomial-time or otherwise (this is because the labels are essentially the outcomes of the flip of a fair coin). However, we will shortly see that the input β_0 and the dependence of the running time on $1/(\frac{1}{2}-\beta_0)$ are usually superfluous, since for learning under arbitrary target distributions to be possible we must have $\beta < \epsilon/(1+\epsilon)$ (under very weak restrictions on C). This is Theorem 1. However, we include β_0 in our definitions since these dependencies may be meaningful for learning under restricted target distributions.

It is important to note that in this definition, we are *not* asking learning algorithms to "fit the noise" in the sense of achieving accuracy in predicting the behavior of the tainted oracles POS_{MAL}^{β} and NEG_{MAL}^{β}. Rather, the conditions $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$ require that the algorithm find a good predictive model of the true underlying target distributions D^+ and D^-, as in the error-free model.

In general, we expect the achievable malicious error rate to depend upon the desired accuracy ϵ and confidence δ, as well as on the parameter n in the case of parameterized representation classes. We now make definitions that will allow us to study the largest rate $\beta = \beta(\epsilon, \delta, n)$ that can be tolerated by any learning algorithm and by learning algorithms restricted to run in polynomial time.

Optimal malicious error rates. Let A be a learning algorithm for C. We define $E_{MAL}(C, A)$ to be the largest β such that A is a β-tolerant learning algorithm for C; note that $E_{MAL}(C, A)$ is actually a function of ϵ and δ (and n in the case of parameterized C). In the case that the largest such β is not well defined (for example, A could tolerate progressively larger rates if allowed more time), then $E_{MAL}(C, A)$ is the supremum over all malicious error rates tolerated by A. Then we define the function $E_{MAL}(C)$ to be the pointwise (with respect to

ϵ, δ and n in the parameterized case) supremum of $E_{MAL}(C, A)$, taken over all learning algorithms A for C. More formally, if we write $E_{MAL}(C, A)$ and $E_{MAL}(C)$ in functional form, then $E_{MAL}(c)(\epsilon, \delta, n) = \sup_A\{E_{MAL}(C, A)(\epsilon, \delta, n)\}$. Notice that this supremum is taken over *all* learning algorithms, regardless of computational complexity. We will use the notation E^{poly}_{MAL} to denote these same quantities when the quantification is only over polynomial-time learning algorithms—thus, for instance, $E^{poly}_{MAL}(C, A)$ is the largest β such that A is a β-tolerant learning polynomial-time learning algorithm for C, and $E^{poly}_{MAL}(C)$ is the largest malicious error rate tolerated by any polynomial-time learning algorithm for C.

$E_{MAL,+}(c)$ will be used to denote E_{MAL} with quantification only over positive-only learning algorithms for C. Similar definitions are made for the negative-only malicious error rate $E_{MAL,-}$, and polynomial-time positive-only and polynomial-time negative-only malicious error rates $E^{poly}_{MAL,+}$ and $E^{poly}_{MAL,-}$.

Oracles with classification noise. Some of our results will also apply to a more benign model of errors defined by Angluin and Laird [1], which we will call the *classification noise* model. Here we have oracles POS^{β}_{CN} and NEG^{β}_{CN} that behave as follows: As before, with probability $1 - \beta$, POS^{β}_{CN} returns a point drawn randomly according to the target distribution D^+. However, with probability β, POS^{β}_{CN} returns a point drawn randomly according to the *negative* target distribution D^-. Similarly, with probability $1 - \beta$, NEG^{β}_{CN} draws from the correct distribution D^- and with probability β draws from D^+. This model is easily seen to be equivalent (modulo polynomial time) to a model in which a learning algorithm asks for a labeled example without being allowed to specify whether this example will be positive or negative; then the noisy oracle draws from the underlying target distributions (each with equal probability), but with probability β returns an incorrect classification with the example drawn.

These oracles are intended to model a situation in which the learning algorithm's "teacher" occasionally misclassifies a positive example as negative and vice versa. However, this misclassification is benign in the sense that the erroneous example is always drawn according to the "natural" environment as represented by the target distributions; thus, only the classification label is subject to error. In contrast, errors in the malicious model may involve not only misclassification, but alteration of the examples themselves, which may not be generated according to any probability distribution at all. As an example, the adversary generating the errors may choose to give significant probability to examples that have zero probability in the true target distributions. We will see throughout the paper that these added capabilities of the adversary have a crucial effect on the error rates that can be tolerated.

Learning from oracles with classification noise. Let C and H be representation classes over X. Then for $0 \le \beta < \frac{1}{2}$, we say that C is *learnable by H with classification noise rate β* if there is a (probabilistic) algorithm A with access to POS^{β}_{CN} and NEG^{β}_{CN}, taking inputs ϵ, δ, and β_0, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input values $0 < \epsilon, \delta < 1$, and $\beta \le \beta_0 < \frac{1}{2}$, algorithm A halts and outputs a representation $h_A \in H$ that with probability at least $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$.

Polynomial time here means polynomial in $1/\epsilon$, $1/\delta$, and $1/(\frac{1}{2} - \beta_0)$, as well as the polynomial in n in the case of parameterized C. As opposed to the malicious case, the input β_0 is relevant here, even in the case of arbitrary target distributions, since classification noise rates approaching $\frac{1}{2}$ can be tolerated by polynomial-time algorithms for some nontrivial representation classes [1].

Optimal classification noise rates. Analogous to the malicious model, we define *classification noise rates E_{CN}, $E_{CN,+}$,* and $E_{CN,-}$ for an algorithm A and representation class C, as well as polynomial-time classification noise rates E^{poly}_{CN}, $E^{poly}_{CN,+}$, and $E^{poly}_{CN,-}$.

2.4. Other definitions and notation.

Sample complexity. Let A be a learning algorithm for a representation class C. Then we denote by $S_A(\epsilon, \delta)$ the number of calls to the oracles *POS* and *NEG* made by A on inputs ϵ, δ; this is a worst-case measure over all possible target representations in C and all target distributions D^+ and D^-. In the case that C is a parameterized representation class, we also allow S_A to depend on the parameter n. We call the function S_A the *sample complexity* or *sample size* of A. We denote by S_A^+ and S_A^- the number of calls of A to *POS* and *NEG*, respectively.

Chernoff bounds. We shall make extensive use of the following bounds on the area under the tails of the binomial distribution. For $0 \leq p \leq 1$ and m a positive integer, let $LE(p, m, r)$ denote the probability of at most r successes in m independent trials of a Bernoulli variable with probability of success p, and let $GE(p, m, r)$ denote the probability of at least r successes. Then for $0 \leq \alpha \leq 1$,

Fact CB1. $LE(p, m, (1-\alpha)mp) \leq e^{-\alpha^2 mp/2}$.

Fact CB2. $GE(p, m, (1+\alpha)mp) \leq e^{-\alpha^2 mp/3}$.

These bounds in the form they are stated are from the paper of Angluin and Valiant [2]; see also Chernoff [6]. Although we will make frequent use of Fact CB1 and Fact CB2, we will do so in varying levels of detail, depending on the complexity of the calculation involved. However, we are primarily interested in Chernoff bounds for the following consequence of Fact CB1 and Fact CB2: Given an event E of probability p, we can obtain an estimate $\hat{p}$ of p by drawing m points from the distribution and letting $\hat{p}$ be the frequency with which E occurs in this sample. Then for m polynomial in $1/p$ and $1/\alpha$, $\hat{p}$ satisfies $p/2 < \hat{p} < 2p$ with probability at least $1-\alpha$. If we also allow m to depend polynomially on $1/\beta$, we can obtain an estimate $\hat{p}$ such that $p - \beta < \hat{p} < p + \beta$ with probability at least $1 - \alpha$.

The Vapnik–Chervonenkis dimension. Let C be a representation class over X. Let $Y \subseteq X$, and define

$$\prod_C(Y) = \{Z \subseteq Y : Z = Y \cap pos(c) \text{ for some } c \in C\}.$$

If we have $\prod_C(Y) = 2^Y$, then we say that Y is *shattered* by C. Then we define

$$\mathrm{VCD}(C) = \max\{|Y| : Y \text{ shattered by } C\}.$$

If this maximum does not exist, then VCD(C) is infinite. The Vapnik–Chervonenkis was originally introduced in the paper of Vapnik and Chervonenkis [25] and was first studied in the context of the distribution-free model by Blumer et al. [5].

Notational conventions. Let $E(x)$ be an event and $\psi(x)$ a random variable that depend on a parameter x that takes on values in a set X. Then for $X' \subseteq X$, we denote by $\mathbf{Pr}_{x\in X'}[E(x)]$ the probability that E occurs when x is drawn uniformly at random from X'. Similarly, $\mathbf{E}_{x\in X'}[\psi(x)]$ is the expected value of ψ when x is drawn uniformly at random from X'. We also need to work with distributions other than the uniform distribution; thus if P is a distribution over X we use $\mathbf{Pr}_{x\in P}[E(x)]$ and $\mathbf{E}_{x\in P}[\psi(x)]$ to denote the probability of E and the expected value of ψ, respectively, when x is drawn accordingly to the distribution P. When E or ψ depend on several parameters that are drawn from different distributions we use multiple subscripts. For example, $\mathbf{Pr}_{x_1\in P_1, x_2\in P_2, x_3\in P_3}[E(x_1, x_2, x_3)]$ denotes the probability of event E when x_1 is drawn from distribution P_1, x_2 from P_2, and x_3 from P_3.

2.5. Some representation classes. We now define the parametrized representation classes whose error-tolerant learnability we will study. Here the domain X_n is always $\{0, 1\}^n$ and the mapping σ simply maps each formula to its set of satisfying assignments. The classes defined below are all parameterized; for each class we will define the subclasses C_n, and then C is defined by $C = \cup_{n \geq 1} C_n$.

Monomials. The representation class M_n consists of all conjunctions of literals over the Boolean variables $x_1, \ldots, x_n$.

k**CNF.** For any constant k, the representation class $k\text{CNF}_n$ consists of all Boolean formulae of the form $C_1 \wedge \cdots \wedge C_l$, where each clause C_i is a disjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$. Note that $M_n = 1\text{CNF}_n$.

k**DNF.** For any constant k, the representation class $k\text{DNF}_n$ consists of all Boolean formulae of the form $T_1 \vee \cdots \vee T_l$, where each term T_i is a conjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$.

Symmetric functions. A *symmetric function* over the Boolean variables $x_1, \ldots, x_n$ is a Boolean function whose output is invariant under all permutations of the input bits. Such a function can be represented by a Boolean array of size $n + 1$, where the ith entry indicates whether the function is 0 or 1 on all inputs with exactly i bits set to 1. We denote by SF_n the class of all such representations.

Decision lists. A *decision list* [19] is a list $L = \langle (T_1, b_1), \ldots, (T_l, b_l) \rangle$, where each T_i is a monomial over the Boolean variables $x_1, \ldots, x_n$ and each $b_i \in \{0, 1\}$. For $\vec{v} \in \{0, 1\}^n$, we define $L(\vec{v})$ as follows: $L(\vec{v}) = b_j$, where $1 \leq j \leq l$ is the least value such that $\vec{v}$ satisfies the monomial T_j; if there is no such j, then $L(\vec{v}) = 0$. We denote the class of all such representations by DL_n. For any constant k, if each monomial T_i has at most k literals, then we have a *k-decision list*, and we denote the class of all such representations by $k\text{DL}_n$.

3. Absolute limits on learning with errors. In this section we prove theorems bounding the achievable error rate for both the malicious error and classification noise models. These bounds are absolute in the sense that they apply to *any* learning algorithm, regardless of its computational complexity, the number of examples it uses, the hypothesis space it uses, and so on. Our first such result states that the malicious error rate must be smaller than the desired accuracy ϵ. This is in sharp contrast to the classification noise model, where Angluin and Laird [1] proved, for example, $E_{CN}^{poly}(k\text{DNF}_n) \geq c_0$ for all n and any constant $c_0 < \frac{1}{2}$.

Let us call a representation class C *distinct* if there exist representations $c_1, c_2 \in C$ and points $u, v, w, x \in X$ satisfying $u \in pos(c_1), u \in neg(c_2), v \in pos(c_1), v \in pos(c_2), w \in neg(c_1), w \in pos(c_2)$, and $x \in neg(c_1), x \in neg(c_2)$.

THEOREM 1. *Let C be a distinct representation class. Then*

$$E_{MAL}(C) < \frac{\epsilon}{1 + \epsilon}.$$

Proof. We use a technique that we will call the *method of induced distributions*: We choose $l \geq 2$ representations $\{c_i\}_{i \in \{1,\ldots,l\}} \subseteq C$, along with l pairs of target distributions $\{D_{c_i}^+\}_{i \in \{1,\ldots,l\}}$ and $\{D_{c_1}^-\}_{i \in \{1,\ldots,l\}}$. These representations and target distributions are such that for any $i \neq j$, $1 \leq i, j \leq l$, c_j is ϵ-bad with respect to the distributions $D_{c_i}^+, D_{c_i}^-$. Then adversaries $\{ADV_{c_1}\}_{i \in \{1,\ldots,l\}}$ are constructed for generating any errors when c_i is the target representation such that the behavior of the oracle POS_{MAL}^{β} is identical regardless of which c_i is the target representation; the same is true for the oracle NEG_{MAL}^{β}, thus making it impossible for any learning algorithm to distinguish the true target representation, and essentially forcing the algorithm to "guess" one of the c_i.

In the case of Theorem 1, this technique is easily applied, with $l = 2$, as follows: Let $c_1, c_2 \in C$ and $u, v, w, x \in X$ be as in the definition of distinct. Define the following target distributions for c_1:

$$D^+_{c_1}(u) = \epsilon,$$
$$D^+_{c_1}(v) = 1 - \epsilon,$$

and

$$D^-_{c_1}(w) = \epsilon,$$
$$D^-_{c_1}(x) = 1 - \epsilon.$$

For c_2, the target distributions are

$$D^+_{c_2}(v) = 1 - \epsilon,$$
$$D^+_{c_2}(w) = \epsilon,$$

and

$$D^-_{c_2}(u) = \epsilon,$$
$$D^-_{c_2}(x) = 1 - \epsilon.$$

Note that these distributions are such that any representation that disagrees with the target representation on one of the points u, v, w, x is ϵ-bad with respect to the target distributions. Now if c_1 is the target representation, then the adversary ADV_{c_1} behaves as follows: On calls to POS^β_{MAL}, ADV_{c_1} always returns the point w whenever an error occurs; on calls to NEG^β_{MAL}, ADV_{c_1} always returns the point u whenever an error occurs. Under these definitions, the oracle POS^β_{MAL} draws a point from an *induced* distribution $I^+_{c_1}$ that is determined by the joint behavior of the distribution $D^+_{c_1}$ and the adversary ADV_{c_1} and is given by

$$I^+_{c_1}(u) = (1 - \beta)\epsilon,$$
$$I^+_{c_1}(v) = (1 - \beta)(1 - \epsilon),$$
$$I^+_{c_1}(w) = \beta,$$

where β is the malicious error rate. Similarly, the oracle NEG^β_{MAL} draws from an induced distribution $I^-_{c_1}$:

$$I^-_{c_1}(u) = \beta,$$
$$I^-_{c_1}(w) = (1 - \beta)\epsilon,$$
$$I^-_{c_1}(x) = (1 - \beta)(1 - \epsilon).$$

For target representation c_2, the adversary ADV_{c_2} always returns the point u whenever a call to POS^β_{MAL} results in an error and always returns the point w whenever a call to NEG^β_{MAL} results in an error. Then the oracle POS^β_{MAL} draws from the induced distribution

$$I^+_{c_2}(u) = \beta,$$
$$I^+_{c_2}(v) = (1 - \beta)(1 - \epsilon),$$
$$I^+_{c_2}(w) = (1 - \beta)\epsilon,$$

and the oracle NEG^{β}_{MAL} from the induced distribution

$$I^{-}_{c_2}(u) = (1-\beta)\epsilon,$$
$$I^{-}_{c_2}(w) = \beta,$$
$$I^{-}_{c_2}(x) = (1-\beta)(1-\epsilon).$$

It is easily verified that if $\beta = \epsilon/(1+\epsilon)$, then the distributions $I^{+}_{c_1}$ and $I^{+}_{c_2}$ are identical, and that $I^{-}_{c_1}$ and $I^{-}_{c_2}$ are identical; if $\beta > \epsilon/(1+\epsilon)$, the adversary may always choose to flip a biased coin and be "honest" (i.e., draw from the correct target distribution) when the outcome is heads, thus reducing the effective error rate to exactly $\epsilon/(1+\epsilon)$. Thus, under these distributions and adversaries, the behavior of the oracles POS^{β}_{MAL} and NEG^{β}_{MAL} is identical regardless of the target representation. This implies that any algorithm that produces an ϵ-good hypothesis for target representations c_1 with probability at least $1-\delta$ under the distributions $D^{+}_{c_1}$ and $D^{-}_{c_1}$ must fail to output an ϵ-good hypothesis for target representation c_2 with probability at least $1-c$ under the distributions $D^{+}_{c_2}$ and $D^{-}_{c_2}$, thus proving the theorem. □

An intuitive interpretation of the result is that if we desire 90 percent accuracy from the hypothesis, there must be less than about 10 percent error.

We emphasize that Theorem 1 bounds the achievable malicious error rate for *any* learning algorithm, regardless of computational complexity, sample complexity, or the hypothesis class. Thus, for distinct C, we always have $E_{MAL}(C) \leq \epsilon/(1+\epsilon) = O(\epsilon)$. All of the representation classes studied here are distinct. We shall see in Theorem 7 of §4 that any hypothesis that nearly minimizes the number of disagreements with a large enough sample from POS^{β}_{MAL} and NEG^{β}_{MAL} is ϵ-good with high probability provided $\beta < \epsilon/4$. Thus, for the finite representation classes we study here (such as all the classes over the Boolean domain $\{0,1\}^n$), there is always a (possibly super-polynomial-time) exhaustive search algorithm A achieving $E_{MAL}(C, A) = \Omega(\epsilon)$; combined with Theorem 1, this gives $E_{MAL}(c) = \Theta(\epsilon)$ for these classes. However, we will primarily be concerned with achieving the largest possible malicious error rate in polynomial time.

We now turn our attention to positive-only and negative-only learning in the presence of errors, where we will see that for many representation classes, the absolute bounds on the achievable error rate are even stronger than those given by Theorem 1.

Let C be a representation class. We will call C *positive t-splittable* if there exist representations $c_1, \ldots, c_t \in C$ and points $u_1 \ldots, u_t \in X$ and $v \in X$ satisfying all of the following conditions:

$$u_i \in pos(c_j), i \neq j, 1 \leq i, j \leq t,$$
$$u_j \in neg(c_j), 1 \leq j \leq t,$$
$$v \in pos(c_i), 1 \leq i \leq t.$$

Similarly, C is *negative t-splittable* if we have

$$v_i \in neg(c_j), i \neq j, 1 \leq i, j \leq t,$$
$$u_j \in pos(c_j), 1 \leq j \leq t,$$
$$v \in neg(c_i), 1 \leq i \leq t.$$

Note that if $\mathrm{VCD}(C) = d$, then C is both positive and negative d-splittable. The converse does not necessarily hold.

THEOREM 2. *Let C be positive t-splittable (respectively, negative t-splittable). Then for $\delta \leq 1/t$,*

$$E_{MAL,+}(C) < \frac{\epsilon}{t-1}$$

(respectively, $E_{MAL,-}(c) < (\epsilon/t - 1)$).

Proof. The proof is by the method of induced distributions. We prove only the case that C is positive t-splittable; the proof for C negative t-splittable is similar. Let $c_1, \ldots, c_t \in C$ and $u_1, \ldots, u_t, v \in X$ be as in the definition of positive t-splittable. For target representation c_j, define the target distributions $D^+_{c_j}$ over $pos(c_j)$ and $D^-_{c_j}$ over $neg(c_j)$ as follows:

$$D^+_{c_j}(u_i) = \frac{\epsilon}{t-1}, \qquad 1 \leq i \leq t, \qquad i \neq j,$$
$$D^+_{c_j}(v) = 1 - \epsilon,$$

and

$$D^-_{c_j}(u_j) = 1.$$

For target representation c_j, the errors on calls to POS^{β}_{MAL} are generated by an adversary ADV_{cj} who always returns the point u_j whenever an error occurs. Then under these definitions, POS^{β}_{MAL} draws a point from a distribution $I^+_{c_j}$ induced by the distribution $D^+_{c_j}$ and the adversary ADV_{c_j}. This distribution is

$$I^+_{c_j}(u_i) = (1-\beta)\frac{\epsilon}{t-1}, \qquad 1 \leq i \leq t, \qquad i \neq j,$$
$$I^+_{c_j}(v) = (1-\beta)(1-\epsilon),$$
$$I^+_{c_j}(u_j) = \beta.$$

If $\beta = (1-\beta)(\epsilon/(t-1))$, then the induced distributions $I^+_{c_j}$ are all identical for $1 \leq j \leq t$. Solving, we obtain $\beta = (\epsilon/(t-1))/(1+\epsilon/(t-1)) < \epsilon/(t-1)$. Now let $\beta \geq \epsilon/(t-1)$, and assume A is a β-tolerant positive-only learning algorithm for C. If c_j is the target representation, then with probability at least $1-\delta$, $u_i \in pos(h_A)$ for some $i \neq j$, otherwise $e^+(h_A) \geq \epsilon$ under the induced distribution $I^+_{c_j}$. Let k be such that

$$\mathbf{Pr}[u_k \in pos(h_A)] = \max_{1 \leq i \leq t}\{\mathbf{Pr}[u_i \in pos(h_A)]\},$$

where the probability is taken over all sequences of examples given to A by the oracle POS^{β}_{MAL} and the coin tosses of A. Then we must have

$$\mathbf{Pr}[u_k \in pos(h_A)] \geq \frac{1-\delta}{t-1}.$$

Choose $\delta < 1/t$. Then with probability at least δ, $e^-(h_A) = 1$ when c_k is the target representation, with distributions $D^+_{c_k}$ and $D^-_{c_k}$ and adversary ADV_{c_k}. This contradicts the assumption that A is a β-tolerant learning algorithm, and the theorem follows. □

Note that the restriction $\delta < 1/t$ in the proof of Theorem 2 is apparently necessary, since a learning algorithm may always randomly choose a u_j to be a positive example and make all other u_i negative examples; the probability of failing to learn under the given distributions is then only $1/t$. It would be interesting to find a different proof that removed this restriction or to prove that it is required.

As in the case of Theorem 1, Theorem 2 is an upper bound on the achievable malicious error rate for *all* learning algorithms, regardless of hypothesis representation, number of examples used, or computation time. For any representation class C, by computing a value T such that C is t-splittable, we can obtain upper bounds on the positive-only and negative-only error rates for that class. As examples, we state such results as corollaries for a few of the representation classes studied here. Even in cases where the representation class is known to be not learnable from only positive or only negative examples in polynomial time (for example, it is shown in Kearns et al. [13] that monomials are not polynomially learnable from negative examples), the bounds on $E_{MAL,+}$ and $E_{MAL,-}$ are relevant since they also hold for algorithms that do not run in polynomial time.

COROLLARY 3. *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E_{MAL,+}(M_n) < \frac{\epsilon}{n-1}$$

and

$$E_{MAL,-}(M_n) < \frac{\epsilon}{n-1}.$$

COROLLARY 4. *Let SF_n be the class of symmetric functions over $x_1, \ldots, x_n$. Then*

$$E_{MAL,+}(\mathrm{SF}_n) < \frac{\epsilon}{n-1}$$

and

$$E_{MAL,-}(\mathrm{SF}_n) < \frac{\epsilon}{n-1}.$$

Proofs of these corollaries follow from the Vapnik–Chervonenkis dimension of the representation classes and Theorem 2. Note that the proof of Theorem 2 shows that these corollaries actually hold for any fixed ϵ and n.

We note that Theorem 2 and its corollaries also hold for the classification noise model. To see this it suffices to notice that the adversaries ADV_{c_j} in the proof of Theorem 2 simulated the classification noise model. Thus, for classification noise we see that the power of using both positive and negative examples may be dramatic: for kCNF we have $E_{CN}^{poly}(k\mathrm{CNF}_n) \geq c_0$ for any $c_0 < 1/2$ due to Angluin and Laird [1] but $E_{CN,+}(k\mathrm{CNF}_n) = O(\epsilon/n^k)$ by Theorem 2. (Kearns et al. [13] show that kCNF is not learnable in polynomial time from negative examples even in the error-free model.) In fact, we can give a bound on $E_{CN,+}$ and $E_{CN,-}$ that is weaker but more general and applies to almost any representation class. Note that by exhaustive search techniques, we have that for any small constant α, $E_{CN}(C) \geq 1/2 - \alpha$ for any finite representation class C. Thus the following result demonstrates that for representation classes over finite domains in the classification noise model, the advantage of using both positive and negative examples is almost always significant.

We will call a representation class C *positive* (respectively, *negative*) *incomparable* if there are representations $c_1, c_2 \in C$ and points $u, v, w \in X$ satisfying $u \in pos(c_1), u \in neg(c_2), v \in pos(c_1), v \in pos(c_2)$ (respectively, $v \in neg(c_1), v \in neg(c_2)$), $w \in neg(c_1)$, $w \in pos(c_2)$.

THEOREM 5. *Let C be positive (respectively, negative) incomparable. Then*

$$E_{CN,+}(C) < \frac{\epsilon}{1+\epsilon}$$

(respectively, $E_{CN,-}(C) < \frac{\epsilon}{1+\epsilon}$).

Proof. By the method of induced distributions. We do the proof for the case that C is positive incomparable; the proof when C is negative incomparable is similar. Let $c_1, c_2 \in C$ and $u, v, w \in X$ be as in the definition of positive incomparable. For target representation c_1, we define distributions

$$D^+_{c_1}(u) = \epsilon,$$
$$D^+_{c_1}(v) = 1 - \epsilon,$$

and

$$D^-_{c_1}(w) = 1.$$

Then in the classification noise model, the oracle POS^{β}_{CN} draws from the induced distribution

$$I^+_{c_1}(u) = (1 - \beta)\epsilon,$$
$$I^+_{c_1}(v) = (1 - \beta)(1 - \epsilon),$$
$$I^+_{c_1}(w) = \beta,$$

and

$$D^-_{c_1}(w) = 1.$$

Then in the classification noise model, the oracle POS^{β}_{CN} draws from the induced distribution

$$I^+_{c_1}(u) = (1 - \beta)\epsilon,$$
$$I^+_{c_1}(v) = (1 - \beta)(1 - \epsilon),$$
$$I^+_{c_1}(w) = \beta.$$

For target representation c_2, define distributions

$$D^+_{c_2}(v) = 1 - \epsilon,$$
$$D^+_{c_2}(w) = \epsilon,$$

and

$$D^-_{c_2}(u) = 1.$$

Then for target representation c_2, oracle POS^{β}_{CN} draws from the induced distribution

$$I^+_{c_2}(u) = \beta,$$
$$I^+_{c_2}(v) = (1 - \beta)(1 - \epsilon),$$
$$I^+_{c_2}(w) = (1 - \beta)\epsilon.$$

For $\beta = \epsilon/(1+\epsilon)$, distributions $I^+_{c_1}$ and $I^+_{c_2}$ are identical. Any positive-only algorithm learning c_1 under $D^+_{c_1}$ and $D^-_{c_1}$ with probability at least $1 - \delta$ must fail with probability at least $1 - \delta$ when learning c_2 under $D^+_{c_2}$ and $D^-_{c_2}$. □

Thus, for positive (respectively, negative) incomparable C, $E_{CN,+}(C) = O(\epsilon)$ (respectively, $E_{CN,-}(C) = O(\epsilon)$). All of the representation classes studied here are both positive and negative incomparable. Note that the proof of Theorem 5 depends upon the assumption that a learning algorithm has only an upper bound on the noise rate, not the exact value; thus, the *effective* noise rate may be less than the given upper bound. This issue does not arise in the malicious model, where the adversary may always choose to draw from the correct target distribution with some fixed probability, thus reducing the effective error rate to any value less than or equal to the given upper bound.

4. Efficient error-tolerant learning.

Given the absolute upper bounds on the achievable malicious error rate of §3, we now wish to find *efficient* algorithms tolerating a rate that comes as close as possible to these bounds or give evidence for the computational difficulty of approaching the optimal error rate. In this section we give efficient algorithms for several representation classes and analyze their tolerance to malicious errors.

We begin by giving a generalization of Occam's razor [4] for the case when errors are present in the examples.

Let C and H be representation classes over X. Let A be an algorithm accessing POS^{β}_{MAL} and NEG^{β}_{MAL} and taking inputs $0 < \epsilon, \delta < 1$. Suppose that for target representation $c \in C$ and $0 \leq \beta < \epsilon/4$, A makes m calls to POS^{β}_{MAL} and receives points $u_1, \ldots, u_m \in X$, and m calls to NEG^{β}_{MAL} and receives points $v_1, \ldots, v_m \in X$ and outputs $h_A \in H$ satisfying with probability at least $1 - \delta$:

$$|\{u_i : u_i \in neg(h_A)\}| \leq \frac{\epsilon}{2}m, \tag{1}$$

$$|\{v_i : v_i \in pos(h_A)\}| \leq \frac{\epsilon}{2}m. \tag{2}$$

Thus, with high probability, h_A is consistent with at least a fraction $1 - \epsilon/2$ of the sample received from the faulty oracles POS^{β}_{MAL} and NEG^{β}_{MAL}. We will call such an A a *β-tolerant Occam algorithm for C by H.*

THEOREM 6. *Let $B < \epsilon/4$, and let A be a β-tolerant Occam algorithm for C by H. Then A is a β-tolerant learning algorithm for C by H; the sample size required is $m = O(1/\epsilon \ln 1/\delta + 1/\epsilon \ln |H|)$. If A is such that only Condition 1 (respectively, Condition 2) above holds, then $e^+(h_A) < \epsilon$ (respectively, $e^-(h_A) < \epsilon$) with probability at least $1 - \delta$.*

Proof. We prove the statement where A meets Condition 1; the case for Condition 2 is similar. Let $h \in H$ be such that $e^+(h) \geq \epsilon$. Then the probability that h agrees with a point received from the oracle POS^{β}_{MAL} is bounded above by

$$(1 - \beta)(1 - \epsilon) + \beta \leq 1 - \frac{3\epsilon}{4}$$

for $\beta < \epsilon/4$. Thus the probability that h agrees with at least a fraction $1 - \epsilon/2$ of m examples received from POS^{β}_{MAL} is

$$LE\left(\frac{3\epsilon}{4}, m, \frac{\epsilon}{2}m\right) \leq e^{-m\epsilon/24}$$

by Fact CB1. From this it follows that the probability that *some* $h \in H$ with $e^+(h) \geq \epsilon$ agrees with a fraction $1 - \epsilon/2$ of the m examples is at most $|H|e^{-m\epsilon/24}$. Solving $|H|e^{-m\epsilon/24} \leq \delta/2$, we obtain $m \geq 24/\epsilon(\ln |H| + \ln 2/\delta)$. This proves that any h meeting Condition 1 is with high probability ϵ-good with respect to D^+, completing the proof. □

To demonstrate that the suggested approach of finding a nearly consistent hypothesis is in fact a feasible one, we note that if c is the target representation, then the probability that c fails to agree with at least a fraction $1 - \epsilon/2$ of m examples received from POS^{β}_{MAL} is

$$GE\left(\frac{\epsilon}{4}, m, \frac{\epsilon}{2}m\right) \leq \frac{\delta}{2}$$

for $\beta \leq \epsilon/4$ and m as in the statement of Theorem 6 by Fact CB2.

Thus, in the presence of errors of any kind, finding an $\epsilon/2$-good hypothesis is as good as learning, provided that $\beta < \epsilon/4$. This fact can be used to prove the correctness of the learning algorithms of the following two theorems due to Valiant.

THEOREM 7 [24]. *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,+}(M_n) = \Omega\left(\frac{\epsilon}{n}\right).$$

THEOREM 8 [24]. *For fixed k, let kDNF$_n$ be the class of kDNF formulae over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,-}(k\text{DNF}_n) = \Omega\left(\frac{\epsilon}{n^k}\right).$$

Similar results are obtained by duality for the class of disjunctions (learnable from negative examples) and kCNF (learnable from positive examples); that is, $E^{poly}_{MAL,-}(1\text{DNF}_n) = \Omega(\epsilon/n)$ and $E^{poly}_{MAL,+}(k\text{CNF}_n) = \Omega(\epsilon/n^k)$. Note that the class of monomials (respectively, kDNF) is not polynomially learnable even in the error-free case from negative (respectively, positive) examples [13].

Combining Corollaries 8 and 9 with Corollaries 3 and 4, we have $E^{poly}_{MAL,+}(M_n) = \Theta(\epsilon/n)$ and $E^{poly}_{MAL,-}(k\text{DNF}_n) = \Theta(\epsilon/n^k)$, thus proving that the algorithms of Valiant [24] tolerate the optimal malicious error rate with respect to positive-only and negative-only learning. The algorithm given in the following theorem, similar to those of Valiant [24], proves an analogous result for efficiently learning symmetric functions from only one type of examples in the presence of errors.

THEOREM 9. *Let SF$_n$ be the class of symmetric functions over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,+}(\text{SF}_n) = \Omega\left(\frac{\epsilon}{n}\right).$$

Proof. Let $\beta \leq \epsilon/8n$. The positive-only algorithm A maintains an integer array P indexed $0, \ldots, n$ and initialized to contain 0 at each location. A takes m (calculated below) examples from POS^{β}_{MAL}, and for each vector $\vec{v}$ received, increments $P[index(\vec{v})]$, where $index(\vec{v})$ is the number of bits set to 1 in $\vec{v}$. The hypothesis h_A is defined as follows: All vectors of index i are contained in $pos(h_A)$ if and only if $P[i] \geq (\epsilon/4n)m$; otherwise all vectors of index i are negative examples of h_A.

Note that h_A can disagree with at most a fraction $(\epsilon/4n)(n+1) < \epsilon/2$ of the m vectors received from POS^{β}_{MAL}, so $e^{+}(h_A) < \epsilon$ with high probability by Theorem 7. To prove that $e^{-}(h_A)$ with high probability, suppose that all vectors of index i are negative examples of the target representation (call such an i a *negative index*). Then the probability that a vector of index i is received on a call to POS^{β}_{MAL} is at most $\beta \leq \epsilon/8n$, since this occurs only when there is an error on a call to POS^{β}_{MAL}. Thus the probability of receiving $(\epsilon/4n)m$ vectors of index i in m calls to POS^{β}_{MAL} is

$$GE\left(\frac{\epsilon}{8n}, m, \frac{\epsilon}{4n}m\right) \leq e^{-m\epsilon/24n}$$

by Fact CB2. The probability that some negative index is classified as a positive index by h_A is thus at most

$$(n+1)e^{-m\epsilon/24n} \leq \frac{\delta}{2}$$

for $m = O((n/\epsilon)(\ln n + \ln 1/\delta))$. Thus with high probability, $e^{-}(h_A) = 0$, completing the proof. □

Thus, with Corollary 5 we have $E^{poly}_{MAL,+}(\mathrm{SF}_n) = \Theta(\epsilon/n)$. We can give a dual of the above algorithm to prove $E^{poly}_{MAL,-}(\mathrm{SF}_n) = \Theta(\epsilon/n)$ as well. The number of examples required by the algorithm of Theorem 10 is a factor of n larger than the lower bound given by Ehrenfeucht et al. [8] for the error-free case; whether this increase is necessary for positive-only algorithms in the presence of malicious errors is an open problem.

The next theorem demonstrates that using both positive and negative examples can significantly increase the tolerated error rate in the malicious model.

THEOREM 10. *Let* SF_n *be the class of symmetric functions* (SF) *over* $x_1, \ldots, x_n$. *Then*

$$E^{poly}_{MAL}(\mathrm{SF}|_n) = \Omega(\epsilon).$$

Proof. Algorithm A maintains two integer arrays P and N, each indexed $0, \ldots, n$ and initialized to contain 0 at each location. A first takes m (calculated below) examples from POS^{β}_{MAL} and for each vector $\vec{v}$ received, increments $P[index\ \vec{v}]$, where $index(\vec{v})$ is the number of bits set to 1 in $\vec{v}$. A then takes m examples from NEG^{β}_{MAL} and increments $N[index(\vec{v})]$ for each vector $\vec{v}$ received. The hypothesis h_A is computed as follows: all vectors of index i are contained in $pos(h_A)$ if and only if $P[i] \geq N[i]$; otherwise, all vectors of index i are contained in $neg(h_A)$.

We now show that for sufficiently large m, A is an $\epsilon/8$-tolerant Occam algorithm. For $0 \leq i \leq n$, let $d_i = \min(P[i], N[i])$. Then $d = \sum_{i=0}^{n} d_i$ is the number of vectors in the sample of size $2m$ with which h_A disagrees. Now for each i, either $P[i]$ or $N[i]$ is a lower bound on the number e_i of malicious errors received that have index i; let $e = \sum_{i=0}^{n} e_i$. Note that $e \geq d$. Now the probability that e exceeds $(\epsilon/4)(2m)$ in m calls POS^{β}_{MAL} and m calls to NEG^{β}_{MAL} for $\beta \leq \epsilon/8$ is

$$GE\left(\frac{\epsilon}{8}, 2m, \frac{\epsilon}{4}2m\right) \leq \delta$$

for $m = O(1/\epsilon \ln 1/\delta)$ by Fact CB2. Thus, with high probability the number of disagreements d of h_A on the examples received is less than $(\epsilon/2)m$. This shows that A is an $\epsilon/8$-tolerant Occam algorithm for SF and thus is a learning algorithm for SF by Theorem 7 for $m = O(1/\epsilon \ln 1/\delta + n/\epsilon)$. □

Thus, by Theorems 1 and 11 we have $E^{poly}_{MAL}(\mathrm{SF}_n) = \Theta(\epsilon)$ in contrast with $E^{poly}_{MAL,+}(\mathrm{SF}_n) = \Theta(\epsilon/n)$ and $E^{poly}_{MAL,-}(\mathrm{SF}_n) = \Theta(\epsilon/n)$, a provable increase by using both types of examples. This is also our first example of a nontrivial class for which the optimal error rate $\Theta(\epsilon)$ of Theorem 1 can be achieved by an efficient algorithm. Furthermore, the sample complexity of algorithm A above meets the lower bound (within a constant factor) for the error-free case given by Ehrenfeucht et al. [8]; thus we have an algorithm with optimal sample complexity that tolerates the largest possible malicious error rate. This also demonstrates that it may be difficult to prove general theorems providing hard trade-offs between sample size and error rate.

We note that the proof of Theorem 11 relies only on the fact that there is a small number of equivalence classes of $\{0, 1\}^n$ (namely, the sets of vectors with an equal number of bits set to 1) on which each symmetric function is constant. The same result thus holds for any Boolean representation class with this property.

Now that we have given some simple and efficient error-tolerant algorithms, we turn to the more abstract issue of general-purpose methods of making algorithms more tolerant to errors. It is reasonable to ask whether for an arbitrary representation class C, polynomial learnability of C implies polynomial learnability of C with malicious error rate β, for some nontrivial value of β that depends on C, ϵ, and δ. The next theorem answers this in the affirmative by giving an efficient technique for converting any learning algorithm into an error-tolerant learning algorithm.

THEOREM 11. *Let A be a polynomial-time learning algorithm for C with sample complexity $S_A(\epsilon, \delta)$, and let $s = S_A(\epsilon/8, \frac{1}{2})$. Then for $\epsilon \leq \frac{1}{2}$.*

$$E_{MAL}^{poly}(C) = \Omega\left(\frac{\ln s}{s}\right).$$

Proof. We describe a polynomial-time algorithm A' that tolerates the desired error rate and uses A as a subroutine. Note that S_A (and, hence, s) may also depend upon n in the case of parameterized C.

Algorithm A' will run algorithm A many times with accuracy parameter $\epsilon/8$ and confidence parameter $\frac{1}{2}$. The probability that no errors occur during a single such run is $(1-\beta)^s$. For $\beta \leq \ln s/s$ we have

$$(1-\beta)^s \geq \left(1 - \frac{\ln s}{s}\right)^s \geq \frac{1}{s^2}.$$

(This lower bound can be improved to $1/s^\alpha$ for any constant $\alpha > 1$ provided there is a sufficiently small constant upper bound on ϵ.) Thus, on a single run of A there is probability at least $(1-\delta)1/s^2 = 1/2s^2$ that no errors occur and A outputs an $\epsilon/8$-good hypothesis h_A (call a run of A when this occurs a *successful* run). A' will run A r times. In r runs of A, the probability that no successful run of A occurs is at most

$$\left(1 - \frac{1}{2s^2}\right)^r < \frac{\delta}{3}$$

for $r > 2s^2 \ln 3/\delta$. Let $h_A^1, \ldots, h_A^r$ be the hypotheses output by A on these r runs. Suppose h_A^i is an ϵ-bad hypothesis with respect to the target distributions; without loss of generality, suppose $e^+(h_A^i) \geq \epsilon$. Then the probability that h_A^i agrees with an example returned by the oracle POS_{MAL}^β is then at most $(1-\beta)(1-\epsilon) + \beta \leq 1 - 3\epsilon/4$ for $\beta \leq \epsilon/8$. Thus, the probability that h_A^i agrees with at least a fraction $1-\epsilon/2$ of m examples returned by POS_{MAL}^β is

$$LE\left(\frac{3\epsilon}{4}, m, \frac{\epsilon}{2}m\right) \leq e^{-m\epsilon/24}$$

by Fact CB1. Then it follows that the probability that *some* h_A^i with $e^+(h_A^i) \geq \epsilon$ agrees with a fraction $1-\epsilon/2$ of the m examples returned by POS_{MAL}^β is at most

$$re^{-m\epsilon/24} < \frac{\delta}{3}$$

for $m = O(1/\epsilon \ln r/\delta)$. Using Fact CB2, it can be shown that for $\beta \leq \epsilon/8$ the probability of an $\epsilon/8$-good h_A^i failing to agree with at least a fraction $1 - \epsilon/2$ of the m examples is smaller than $\delta/3$.

Thus, if A is run r times and the resulting hypotheses are tested against m examples from both POS_{MAL}^{β} and NEG_{MAL}^{β}, then with probability at least $1-\delta$ the hypothesis with the fewest disagreements is in fact an ϵ-good hypothesis. Note that if A runs in polynomial time, A' also runs in polynomial time. □

Note that the trick used in the proof of Theorem 12 to eliminate the dependence of the tolerated error rate on δ is general: We may always set $\delta = \frac{1}{2}$ and run A repeatedly to get a good hypothesis with high probability (provided we are willing to sacrifice a possible increase in the number of examples used). This technique has also been noted in the error-free setting by Haussler et al. [10].

It is shown by Ehrenfeucht et al. [8] that any learning algorithm A for a representation class C must have sample complexity

$$S_A(\epsilon, \delta) - \Omega\left(\frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \mathrm{VCD}(C)\right)\right).$$

Suppose that a learning algorithm A achieves this optimal sample complexity. Then applying Theorem 12, we immediately obtain an algorithm for C that tolerates a malicious error rate of

$$\Omega\left(\frac{\epsilon}{\mathrm{VCD}(C)} \ln \frac{\mathrm{VCD}(C)}{\epsilon}\right).$$

This rate is also the best that can be obtained by applying Theorem 12. By applying this technique to the algorithm of Valiant [23] for the class of monomials in the error-free model, we obtain the following corollary.

COROLLARY 12. *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E_{MAL}^{poly}(M_n) = \Omega\left(\frac{\epsilon}{n} \ln \frac{n}{\epsilon}\right).$$

This improves the malicious error rate tolerated by the polynomial-time algorithm of Valiant [24] in Theorem 8 by a logarithmic factor. Furthermore, since $E_{MAL,+}^{poly}(M) = \Theta(\epsilon/n)$ this proves that, as in the case of symmetric functions, using both oracles improves the tolerable error rate. Similarly, a slight improvement over the malicious error rate given in Theorem 9 for kDNF can also be shown. For decision lists, we can apply the algorithm of Rivest [19] and the sample size bounds given by Ehrenfeucht et al. [8] to obtain the following corollary.

COROLLARY 13. *Let $k\mathrm{DL}_n$ be the class of k-decision lists over $x_1, \ldots, x_n$. Then*

$$E_{MAL}^{poly}(k\mathrm{DL}_n) = \Omega\left(\frac{\epsilon}{n^k}\right).$$

Despite the small improvement in the tolerable error rate for monomials of Corollary 12, there is still a significant gap between the absolute upper bound of $\epsilon/(1+\epsilon)$ on the achievable malicious error rate for monomials implied by Theorem 1 and the $\Omega(\epsilon/n \ln n/\epsilon)$ polynomial-time error rate of Corollary 12. We now describe further improvements that allow the error rate to primarily depend only on the number of *relevant* variables. We describe an algorithm tolerating a larger error rate for the class M_n^s of monomials with at most s literals, where s may depend on n, the total number of variables. Our algorithm will tolerate a larger rate of error when the number s of relevant attributes is considerably smaller than the total number of variables n. Other improvements in the performance of learning algorithms in the presence of many irrelevant attributes are investigated by Littlestone [16] and Blum [3].

We note that by applying Theorem 2 we can show that even for M_n^1, the class of monomials of length 1, the positive-only and negative-only malicious error rates are bounded by $\epsilon/(n-1)$. This is again an absolute bound, holding regardless of the computational complexity of the learning algorithm. Thus, the positive-only algorithm of Valiant [24] in Theorem 8 cannot exhibit an improved error rate when restricted to the subclass M_n^s for *any* value of s.

Our error-tolerant learning algorithm for monomials is based on an approximation algorithm for a generalization of the set cover problem that we call the partial cover problem, which is defined below. This approximation algorithm is of independent interest and has found application in other learning algorithms [14], [26]. Our analysis and notation rely heavily on the work of Chvatal [7]; the reader may find it helpful to read his paper first.

The Partial Cover Problem

Input: Finite sets $S_1, \ldots, S_n$ with positive real costs $c_1, \ldots, c_n$, and a positive fraction $0 < p \le 1$.

We assume without loss of generality that $\cup_{i=1}^n S_i = \{1, \ldots, m\} = T$ and we define $J = \{1, \ldots, n\}$.

Output: $J^* \subseteq J$ such that

$$\left| \bigcup_{j \in J^*} S_j \right| \ge pm$$

(we call such a J^* a *p-cover* of the S_i) and such that $cost_{PC}(J^*) = \sum_{j \in J^*} c_j$ is minimized.

Following Chvatal [7], for notational convenience we identify a partial cover $\{S_{j_1}, \ldots, S_{j_s}\}$ with the index set $\{j_1, \ldots, j_s\}$.

The partial cover problem is NP-hard, since it contains the set cover problem as a special case ($p = 1$) [9]. We now give a greedy approximation algorithm G for the partial cover problem.

Algorithm G

Step 1. Initialize $J^* = \emptyset$.
Step 2. If $| \cup_{j \in J^*} S_j | \ge pm$, then halt and output J^*, since J^* is a p-cover.
Step 3. Set $q = pm - | \cup_{j \in J^*} S_j |$ (thus q is the number of still-uncovered elements that we must cover in order to have a p-cover). For each $j \notin J^*$, if $|S_j| > q$, delete any $|S_j| - q$ elements from S_j (delete excess elements from any remaining set that covers more than q elements).
Step 4. Find a k minimizing the ration $c_k/|S_k|$. Add k to J^*, and replace each S_j by $S_j - S_k$. Return to Step 2.

Chvatal [7] shows that the greedy algorithm for the set cover problem cannot do better than $H(m)$ times the cost of an optimal cover, where $H(m) = \sum_{i=1}^m 1/i = \Theta(\log m)$. By a padding argument, this can also be shown to hold for algorithm G above, for any fixed p. We now prove that G can always achieve this approximation bound within a constant factor.

THEOREM 14. *Let I be an instance of partial cover and let $opt_{PC}(I)$ denote the cost of an optimal p-cover for i. Then the cost of the p-cover J^* produced by algorithm G satisfies*

$$cost_{PC}(J^*) \le (2H(m) + 3) opt_{PC}(I).$$

Proof. Let J_{opt} be an optimal p cover (i.e., $cost_{PC}(J_{opt}) = opt_{PC}(I)$). Let

$$T_{opt} = \bigcup_{j \in J_{opt}} S_j$$

(these are the elements covered by J_{opt}) and

$$T^* = \bigcup_{j \in J^*} S_j$$

(these are the elements covered by J^*), where J^* is the p-cover output by algorithm G. Notice that $|T_{opt}| \geq pm$ since J_{opt} is a p-cover.

Let S_j^r be set of elements remaining in the set S_j immediately before Step 3 in algorithm G is executed for the rth time (i.e., at the start of the rth iteration of Steps 2–4). By appropriate renaming of the S_j, we may assume without loss of generality that $J^* = \{1, \ldots, r\}$ (recall that J^* is the set of *indices* of sets chosen by algorithm G) immediately after Step 4 is executed for the rth time (i.e., at the end of the rth iteration of Steps 2–4). Let $J^* = \{1, \ldots, t\}$ when G halts, so there are a total of t iterations.

Define $T^{**} = T^* - S_t'$, where S_t' is the union of all elements deleted from the set S_t on all executions of Step 3. Intuitively, T^{**} consists of those elements that algorithm G "credits" itself with having covered during its execution (as opposed to those elements regarded as "excess" that were covered because G may cover more than the required minimum fraction p). We say that a set S_j is *at capacity* when in Step 3, $|S_j| \geq q$. Note that once S_j reaches capacity, it remains at capacity until it is chosen in Step 4 or until G halts. This is because if l elements are removed from S_j on an execution of Step 4, the value of q in Step 3 will decrease by at least l on the next iteration. Furthermore, since G halts the first time a set at capacity is chosen, and by the above definitions S_t is the last set chosen by G, we have that $T^{**} = \bigcup_{\tau=1}^{t} S_\tau^\tau$. Thus, we have $|S_t'| = |T^*| - pm$ and $|T^{**}| = pm$.

The set S_τ^τ can be regarded as the set of previously uncovered elements that are added to T^{**} on the rth iteration. We wish to amortize the cost c_r over the elements covered. For each $i \in T^*$, we define a number y_i, which is intuitively the cost we paid to put i in T^*:

$$y_i = \begin{cases} c_r/|S_r^r| & \text{if for some } r,\ i \in S_r^r, \\ 0 & i \text{ is not in } T^{**}. \end{cases}$$

Since for $i \in T^* - T^{**}$, $y_i = 0$, we have

$$\begin{aligned} \sum_{i \in T^{**}} y_i &= \sum_{i \in T^*} y_i \\ &= \sum_{r=1}^{t} \sum_{i \in S_r^r} y_i \\ &= \sum_{r=1}^{t} c_r \\ &= \sum_{j \in J^*} c_j \\ &= cost_{PC}(J^*). \end{aligned}$$

Thus to bound $cost_{PC}(J^*)$, we now bound $\sum_{i \in T^{**}} y_i$ in two parts, first bounding $\sum_{i \in T^{**} - T_{opt}} y_i$ and then bounding $\sum_{i \in T^{**} \cap T_{opt}} y_i$.

LEMMA 15.

$$\sum_{i \in T^{**} - T_{opt}} y_i \leq (H(m) + 2) opt_{PC}(I).$$

Proof. If $T^{**} \subseteq T_{opt}$, then the lemma follows trivially. We therefore assume $T^{**} \not\subseteq T_{opt}$. Since $|T_{opt}| \geq pm$ and $|T^{**}| = pm$, this implies $T_{opt} - T^{**} \neq \emptyset$. Pick $j \in J_{opt}$ such that

$$\frac{c_j}{|S_j - T^{**}|}$$

is minimized. Now

$$\begin{aligned}\frac{opt_{PC}(I)}{|T_{opt} - T^{**}|} &= \frac{\sum_{i \in J_{opt}} c_i}{|\bigcup_{i \in J_{opt}} (S_i - T^{**})|} \\ &\geq \frac{\sum_{i \in J_{opt}} c_i}{\sum_{i \in J_{opt}} |S_i - T^{**}|} \\ &\geq \frac{c_j}{|S_j - T^{**}|}.\end{aligned}$$

Thus

$$opt_{PC}(I) \geq |T_{opt} - T^{**}| \frac{c_j}{|S_j - T^{**}|}.$$

Let r_0 be the first execution of Step 3 in which $|S_j| > q$ (i.e., S_j reaches capacity on the r_0th iteration). We will analyze the behavior of G before and after the r_0th iteration separately. Let T_0^{**} denote the set of elements that were added to T^{**} prior to the r_0 iteration. For each $i \in T_0^{**} - T_{opt}$, the cost y_i must satisfy

$$y_i \leq \frac{c_j}{|S_j - T^{**}|}$$

because otherwise G would have already added S_j to J^*. Since $|T_{opt} - T^{**}| \geq |T^{**} - T_{opt}|$ we have

$$\begin{aligned}\sum_{i \in T_0^{**} - T_{opt}} y_i &\leq \sum_{i \in T_0^{**} - T_{opt}} \frac{c_j}{|S_j - T^{**}|} \\ &\leq |T_{opt} - T^{**}| \frac{c_j}{|S_j - T^{**}|} \\ &\leq opt_{PC}(I).\end{aligned}$$

For iterations $r \geq r_0$, whenever an element i is added to $T_1^{**} = T^{**} - T_0^{**}$, an element is deleted from S_j in Step 3, since S_j is at capacity. We charge y_i to this element as follows:

$$\begin{aligned}\sum_{i \in T_1^{**} - T_{opt}} y_i &\leq \sum_{i \in T_1^{**}} y_i \\ &= \sum_{r=r_0}^{t} \sum_{i \in S_r^r} y_i \\ &\leq \sum_{r=r_0}^{t-1} \sum_{i \in S_r^r} y_i + c_j\end{aligned}$$

(because on iteration t, both S_j and S_t are at capacity, so $c_t \leq c_j$)

$$\leq \sum_{r=r_0}^{t-1} \frac{c_r}{|S_r^r|} |S_j^r - S_j^{r+1}| + c_j$$

(because since S_j is at capacity, $|S_j^r - S_j^{r+1}| = |S_r^r|$)

$$\le \sum_{r=r_0}^{t-1} \frac{c_j}{|S_j^r|}|S_j^r - S_j^{r+1}| + c_j$$

(because otherwise G would have chosen S_j at time r)

$$= c_j \sum_{r=r_0}^{t-1} \frac{1}{|S_j^r|}|S_j^r - S_j^{r+1}| + c_j$$
$$\le c_j H(|S_j|) + c_j$$
$$= c_j(H(|S_j|) + 1).$$

Combining the two parts, we have

$$\begin{aligned}\sum_{i\in T^{**}-T_{opt}} y_i &= \sum_{i\in T_0^{**}-T_{opt}} y_i + \sum_{i\in T_1^{**}-T_{opt}} y_i\\ &\le opt_{PC}(I) + c_j(H(m)+1)\\ &\le (H(m)+2)opt_{PC}(I).\end{aligned}$$ □

LEMMA 16.

$$\sum_{i\in T^{**}\cap T_{opt}} y_i \le (H(m)+1)opt_{PC}(I).$$

Proof. We generalize the idea used by Chvatal [7]. For $j \in J_{opt}$ and $S_j \cap T^{**} \neq \emptyset$,

$$\begin{aligned}\sum_{i\in S_j\cap T^{**}} y_i &= \sum_{r=1}^{t} \sum_{i\in S_j\cap S_r^r} y_i\\ &\le \sum_{r=1}^{t-1} \frac{c_r}{|S_r^r|}|S_j^r - S_j^{r+1}| + c_j\end{aligned}$$

(because the average cost of elements in S_t is lower than in S_j, and we are summing over at most $|S_j|$ elements)

$$\le \sum_{r=1}^{s} \frac{c_j}{|S_j^r|}|S_j^r - S_j^{r+1}| + c_j.$$ □

Proof of Theorem 14. Combining Lemmas 15 and 16, we have

$$\begin{aligned}\sum_{j\in J^*} c_j &= \sum_{i\in T^*} y_i\\ &= \sum_{i\in T^{**}} y_i\\ &= \sum_{i\in T^{**}-T_{opt}} y_i + \sum_{i\in T^{**}\cap T_{opt}} y_i\\ &\le (H(m)+2)opt_{PC}(I) + (H(m)+1)opt_{PC}(I)\\ &= (2H(m)+3)opt_{PC}(I).\end{aligned}$$

This completes the proof of Theorem 14. □

We now use algorithm G as a subroutine in constructing our error-tolerant learning algorithm for M_n^s.

THEOREM 17. *Let M_n^s be the class of monomials over $x_1, \ldots, x_n$ containing at most s literals. Then*

$$E_{MAL}^{poly}(M_n^s) = \Omega\left(\frac{\epsilon}{s \log \dfrac{s \log n}{\epsilon}}\right).$$

Proof. We construct an Occam algorithm A for M_n^s that tolerates the desired malicious error rate and uses the algorithm G for the partial cover problem as a subroutine.

Let $0 \leq \beta < \epsilon/8$, and let $c \in M_n^s$ be the target monomial. A first takes m_N points from the oracle NEG_{MAL}^{β}, where $m_N = O(1/\epsilon \ln 1/\delta + 1/\epsilon \ln |M_n^s|)$ as in the statement of Theorem 6. Let S denote the multiset of points received by A from NEG_{MAL}^{β}. For $1 \leq i \leq n$, define the multisets

$$S_i^0 = \{\vec{v} \in S : v_i = 0\}$$

and

$$S_i^1 = \{\vec{v} \in S : v_i = 1\}.$$

We now define a pairing between monomials and partial covers as follows: The literal x_i is paired with the partial cover consisting of the single set S_i^0 and the literal $\bar{x}_i$ is paired with the partial cover consisting of the single set S_i^1. Then any monomial c is paired with the partial cover obtained by including exactly those S_i^0 and S_i^1 that are paired with the literals appearing in c. Note that the multiset $neg(c) \cap S$ contains exactly those vectors that are covered by the corresponding partial cover.

Now with high probability, there must be some collection of the S_i^0 and S_i^1 that together form a $1-\epsilon/2$ cover of S; namely, if (without loss of generality) the target monomial $c \in M_n^s$ is

$$c = x_1 \cdots x_r \bar{x}_{r+1} \cdots \bar{x}_s$$

then with high probability the sets

$$S_1^0, \ldots, S_r^0, S_{r+1}^1, \ldots, S_s^1.$$

We now define a pairing between monomials and partial covers of the set S as follows: The literal x_i is paired with the partial cover consisting of the single set S_i^0 and the literal $\bar{x}_i$ is paired with the partial cover consisting of the single set S_i^1. Then any monomial c is paired with the partial cover obtained by including exactly those S_i^0 and S_i^1 that are paired with the literals appearing in c. Note that the multiset $neg(c) \cap S$ contains exactly those vectors that are covered by the corresponding partial cover.

Now with high probability, there must be some collection of the S_i^0 and S_i^1 that together form a $1-\epsilon/2$ cover of S: Namely, if (without loss of generality) the target monomial $c \in M_n^s$ is

$$c = x_1 \cdots x_r \bar{x}_{r+1} \cdots \bar{x}_s,$$

then with high probability the sets

$$S_1^0, \ldots, S_r^0, S_{r+1}^1, \ldots, S_s^1$$

form a $1 - \epsilon/2$ cover of S, since for $\beta \leq \epsilon/8$, the probability that the target monomial c disagrees with a fraction larger than $\epsilon/2$ of a sample of size m_N from NEG_{MAL}^{β} can be shown to be smaller than $\delta/2$ by Fact CB2.

Thus, A will input the sets $S_1^0, \dots, S_n^0, S_1^1, \dots, S_n^1$ and the value $p = 1-\epsilon/2$ to algorithm G. The costs for these sets input to G are defined below. However, note that regardless of these costs, if h_G is the monomial paired with the p-cover output by G, then since $|neg(h_G) \cap S| \geq (1-\epsilon/2)m_N$ (where $neg(h_G) \cap S$ is interpreted as a multiset), $e^-(h_G) < \epsilon$ with high probability by Theorem 6. We now show that for β as in the statement of the theorem, we can choose the costs input to G so as to force $e^+(h_G) < \epsilon$ as well.

For any monomial c, let $p(c)$ denote the probability that c disagrees with a vector returned by POS_{MAL}^{β},[1] and let $cost_{PC}(c)$ denote the cost of the partial cover that is paired with c. To determine the costs of the sets input to G, A next samples POS_{MAL}^{β} enough times (determined by application of Facts CB1 and CB2) to obtain an estimate for $p(x_i)$ and $p(\bar{x}_i)$ for $1 \leq i \leq n$ that is accurate within a multiplicative factor of 2, that is, if $\hat{p}(x_i)$ is the estimate computed by A, then $p(x_i)/2 \leq \hat{p}(x_i) \leq 2p(x_i)$ with high probability for each i. The same bounds hold for the estimate $\hat{p}(\bar{x}_i)$. Then the cost for set S_i^0 input to G by A is $\hat{p}(x_i)$ and the cost for set S_i^1 is $\hat{p}(\bar{x}_i)$.

Note that for any monomial $c = x_1 \cdots x_r \bar{x}_{r+1} \cdots \bar{x}_s$, we have with high probability

$$\begin{aligned} p(c) &\leq p(x_1) + \cdots + p(x_r) + p(\bar{x}_{r+1}) + \cdots + p(\bar{x}_s) \\ &\leq 2\hat{p}(x_1) + \cdots + 2\hat{p}(x_r) + 2\hat{p}(\bar{x}_{r+1}) + \cdots + 2\hat{p}(\bar{x}_s) \\ &= 2cost_{PC}(c). \end{aligned}$$

By Theorem 17, the output h_G of G must satisfy

$$cost_{PC}(h_G) \leq (H(m_N) + 2)cost_{PC}(c_{opt}) \tag{3}$$

where c_{opt} is the monomial paired with a p-cover of minimum cost. But for the target monomial c we have

$$p(c) \leq \beta, \tag{4}$$

$$2sp(c) \geq cost_{PC}(c), \tag{5}$$

where (4) holds absolutely and (5) holds with high probability, since c contains at most s literals.

From Equations 3, 4, and 5, we obtain with high probability

$$\begin{aligned} p(h_G) &\leq 2cost_{PC}(h_G) \\ &\leq 2(H(m_N) + 2)cost_{PC}(c_{opt}) \\ &\leq 2(H(m_N) + 2)cost_{PC}(c) \\ &\leq 4sp(c)(H(m_N) + 2) \\ &\leq 4s\beta(H(m_N) + 2). \end{aligned}$$

[1]Note that technically this probability may not be well defined since the behavior of the oracle POS_{MAL}^{β} may depend on the entire history of the computation so far. If this is the case, however, we may use the following trick: Rather than running the algorithm using POS_{MAL}^{β}, we instead take a sufficiently large number of examples l from POS_{MAL}^{β} and then run the algorithm using a uniform distribution over these l examples (treated as a multiset, not a set). The algorithm may need to be run more than once in order to find an appropriate setting of the error parameter used; this technique is detailed and shown correctly in Theorem 20. For the rest of the proof, therefore, we assume without loss of generality that $p(c)$ is well defined.

Thus, if we set

$$\beta = \frac{\epsilon}{4s(H(m_N)+2)} = \Omega\left(\frac{\epsilon}{s\log m_N}\right),$$

then $e^+(h_G) < \epsilon$ with high probability by Theorem 6. We can remove the dependence of β on δ by method used in the proof of Theorem 11, thus obtaining an error rate of

$$\Omega\left(\frac{\epsilon}{s\log\dfrac{s\log n}{\epsilon}}\right)$$

completing the proof. □

As an example, if $s = \sqrt{n}$, then Theorem 17 gives

$$E^{poly}_{MAL}(M_n^{\sqrt{n}}) = \Omega\left(\frac{\epsilon}{\sqrt{n}\log\dfrac{n}{\epsilon}}\right)$$

as opposed to the bound of $\Omega(\epsilon/n \ln \epsilon/n)$ of Theorem 12.

Littlestone [16] shows that the Vapnik–Chervonenkis dimension of M_n^s is $\Theta(s\ln(1+n/s))$. Since the algorithm of Valiant [23] can be modified to have optimal sample complexity for M_n^s, by applying Theorem 11 to this modified algorithm we obtain

$$E^{poly}_{MAL}(M_n^s) = \Omega\left(\frac{\epsilon\ln\left(\frac{s}{\epsilon}\ln\left(1+\frac{n}{s}\right)\right)}{s\ln\left(1+\frac{n}{s}\right)}\right).$$

This lower bound on $E^{poly}_{MAL}(M_n^s)$ is incomparable to that of Theorem 17. We may decide at run time which algorithm will tolerate the larger error rate, thus giving

$$E^{poly}_{MAL}(M_n^s) = \Omega\left(\min\left(\frac{\epsilon\ln\left(\frac{s}{\epsilon}\ln\left(1+\frac{n}{s}\right)\right)}{s\ln\left(1+\frac{n}{s}\right)}, \frac{\epsilon}{s\log\dfrac{s\log n}{\epsilon}}\right)\right).$$

By using transformation techniques similar to those described Kearns et al. [13] it can be shown that the algorithm of Theorem 17 (as well as that obtained from Theorem 11) can be used to obtain an improvement in the error rate over the negative-only algorithm of Valiant [24] for the class $k\mathrm{DNF}_{n,s}$ of kDNF formulae with at most s terms. Briefly, the appropriate transformation regards a kDNF formulae as a 1DNF formulae in a space of $\Theta(n^k)$ variables, one variable for each of the possible terms (monomials) of length at most k.

5. Limits on efficient learning with errors. In §3, we saw that there was an absolute bound of $\epsilon/(1+\epsilon)$ on the achievable malicious error rate for most interesting representation classes. It was also argued there that, at least for our finite representation classes over $\{0,1\}^n$, this bound could always be achieved by a super-polynomial-time exhaustive search learning algorithm. Then in §4 we gave polynomial-time learning algorithms that in some cases achieved the optimal error rate $O(\epsilon)$, but in other cases fell short. These observations raise the natural question of whether for some classes it is possible to prove bounds stronger than $\epsilon/(1+\epsilon)$ on the malicious error rate for learning algorithms constrained to run in polynomial time. In particular, for parameterized representation classes, under what conditions must the error rate tolerated by a polynomial-time learning algorithm decrease as the number of variables n increases? If we informally regard the problem of learning with malicious errors as an optimization problem where the objective is to maximize the achievable error rate in polynomial time, and $\epsilon/(1+\epsilon)$ is the optimal value, then we might expect such hardness results to take the form of hardness results for the approximation of NP-hard optimization problems. This is the approach we pursue in this section.

By reducing standard combinatorial optimization problems to learning problems, we state theorems indicating that efficiently learning with an error rate approaching $\Theta(\epsilon)$ is eventually as hard as approximations for NP-hard problems.

In §4 we gave an error-tolerant algorithm for learning monomials by monomials that was based on an approximation algorithm for a generalization of set cover. Our next theorem gives a reduction in the opposite direction: An algorithm learning monomials by monomials and tolerating a malicious error rate approaching $\Theta(\epsilon)$ can be used to obtain an improved approximation algorithm for set cover.

THEOREM 18. *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Suppose there is a polynomial-time learning algorithm A for M_n using hypothesis space M_n such that*

$$E_{MAL}^{poly}(M_n, A) = \frac{\epsilon}{r(n)}.$$

Then there is a polynomial-time algorithm for the weighted set cover problem that outputs (with high probability) a cover whose cost is at most $2r(n)$ times the optimal cost, where n is the number of sets.

Proof. We describe an approximation algorithm A' for set cover that uses the learning algorithm A as a subroutine. Given an instance I of set cover with sets $S_1, \ldots, S_n$ and costs $c_1, \ldots, c_n$, let $J_{opt} \subseteq \{1, \ldots, n\}$ be an optimal cover of $T = \cup_{j=1}^{n} S_j = \{1, \ldots, m\}$, where we identify a cover $\{S_{j1}, \ldots, S_{js}\}$ with its index set $\{j_1, \ldots, j_s\}$. Let $cost_{SC}(J)$ denote the set cover cost of any cover J of T, and let $opt_{SC}(I) = cost_{SC}(J_{opt})$. As in the proof of Theorem 17, we pair a cover $\{j_1, \ldots, j_s\}$ of T with the monomial $x_{j1} \ldots x_{js}$ over the variables $x_1, \ldots, x_n$. Let c_{opt} be the monomial paired with the optimal cover J_{opt}.

The goal of A' is to simulate algorithm A with the intention that c_{opt} is the target monomial, and use the monomial h_A output by A to obtain the desired cover of T. The examples given to A on calls to NEG_{MAL}^{β} during this simulation will be constructed so as to guarantee that the collection of sets paired with h_A is actually a cover of T, while the examples given to A on calls to POS_{MAL}^{β} guarantee that this cover has a cost within a multiplicative factor of $2r(n)$ of the optimal cost.

We first describe the examples A' generates for A on calls to NEG_{MAL}^{β}. For each $i \in T$, let $\vec{u}_i \in \{0,1\}^n$ be the vector whose jth bit is 0 if and only if $i \in S_j$, and let the multiset U be $U = \cup_{i \in T}\{\vec{u}_i\}$. Then $\{j_1, \ldots, j_s\}$ is a cover of T if and only if $U \subseteq neg(x_{j_1} \cdots x_{j_s})$. In particular, we must have $U \subseteq neg(c_{opt})$. Thus, define the target distribution D^- for c_{opt} to be uniform over U. Note that this distribution can be generated in polynomial time by A'. On calls of A to NEG_{MAL}^{β}, A' will simply draw from D^-; thus if we regard c_{opt} as the target monomial, there are no errors in the negative examples. A' will simulate A with accuracy parameter $\epsilon \leq 1/|U|$, thus forcing A to output an hypothesis monomial h_A such that $U \subseteq neg(h_A)$; by the above argument, this implies that the collection of sets paired with the monomial h_A is a cover of T. Note that $|U|$ (and therefore $1/\epsilon$) may be super-polynomial in n, but it is polynomial in the size of the instance I.

We now describe the examples A' generates for A on calls to POS_{MAL}^{β}. Instead of defining the target distribution D^+ for c_{opt}, we define an *induced* distribution I^+ from which the oracle POS_{MAL}^{β} will draw. Thus, I^+ will describe the joint behavior of the underlying distribution D^+

on c_{opt} and an adversary generating the malicious errors. For each $1 \leq j \leq n$, let $\vec{v}_j \in \{0, 1\}^n$ be the vector whose jth bit is 0, and all other bits are 1. Let $I^+(\vec{v}_j) = c_j$ for each j, where c_j is the cost of the set S_j, and we assume without loss of generality that $\sum_{j=1}^n c_j \leq \epsilon/r(n)$ (if not, we can normalize the weights without changing the relative costs of covers). We complete the definition of I^+ by letting $I^+((1, \ldots, 1)) = 1 - \sum_{j=1}^n c_j$. Then the probability that a monomial $x_{i_1} \cdots x_{i_s}$ disagrees with a point drawn from POS_{MAL}^{β} is exactly $c_{i_1} + \cdots + c_{i_s}$, the cost of the corresponding cover. Thus since $opt_{SC}(I) \leq \sum_{j=1}^n c_j \leq \epsilon/r(n) = \beta$, I^+ is an induced distribution for c_{opt}, with malicious error rate β. Note that I^+ can be generated by A' in polynomial time. When A requests an example from POS_{MAL}^{β}, A' will simply draw from I^+.

A' will run algorithm A many times with the oracles POS_{MAL}^{β} and NEG_{MAL}^{β} for c_{opt} described above, each time with a progressively smaller value for the accuracy parameter, starting with $\epsilon = 1/|U|$.

Now if $opt_{SC}(I) << \epsilon/r(n)$, then algorithm A may output a monomial h_A whose corresponding cover has a cost much larger than $4r(n) \cdot opt_{SC}(I)$, since h_A is only guaranteed to satisfy $e^+(h_A) < \epsilon$. We solve this problem by repeated scaling: A' first runs algorithm A with the oracles POS_{MAL}^{β} and NEG_{MAL}^{β} as they have been described. After each run, A' divides the accuracy parameter ϵ by 2, so that on some run $\epsilon/2r(n) \leq opt_{SC}(I) \leq \epsilon/r(n)$. On this run, we may regard I^+ as an induced distribution on the positive examples of c_{opt}, with malicious error rate at most $\beta = \epsilon/r(n) \leq 2opt_{SC}(I)$. Then the error $e^+(h_A)$ on the underlying distribution D^+ over $pos(c_{opt})$ is at most $\epsilon \leq 2r(n)opt_{SC}(I)$. The desired cover is thus the one paired with the monomial h_A. Note that without knowing c_{opt}, we have no way of knowing what the underlying target distribution D^+ is, but it is enough to know that I^+ is a "close" distribution. The only problem with the simulation described occurs when $opt_{SC}(I) << \sum_{j=1}^n c_j$, in which case it may take a superpolynomial number of runs of A to guarantee $\epsilon/2r(n) \leq opt_{SC}(I) \leq \epsilon/r(n)$. We solve this by preprocessing: Before running the described simulation, A' runs the greedy approximation algorithm analyzed by Chvatal [7] on the set cover instance I and removes any set whose cost is larger than the entire cost of the greedy cover. Then for the new (smaller) instance I', every cost is within a multiplicative factor of $\log m$ of every other cost. □

Thus, if $r(n) << \log n$, then Theorem 19 says that a polynomial time algorithm A for M_n (using hypothesis space M_n) tolerating $E_{MAL}^{poly}(M_n, A) \geq \epsilon/r(n)$ would imply a significant breakthrough in approximation algorithms for set cover, since the best algorithm for this problem remains the greedy method analyzed by Chvatal and others [7], [11], [17], [18]. Note that the proof of Theorem 18 in fact shows the result holds for the class of *monotone* monomials.

Theorem 17 took an approximation algorithm for an optimization problem (the partial cover problem), and used it as a subroutine in obtaining an error-tolerant learning algorithm for M_n^s. Theorem 18 proved that when learning algorithms are restricted to hypothesis class M, any learning algorithm for M yields an algorithm for set cover with only a constant factor blowup in the approximation. Thus, we see that there are strong ties between learning with errors and approximating combinatorial optimization problems. Our goal now is to generalize and strengthen these ideas. We show that for any representation class C, the problem of learning C with errors is equivalent to a combinatorial optimization problem with only a constant factor blowup in the approximation in each direction of the reduction.

For domain X, define a *balanced sample* of X to be a sample

$$S = \langle x_1, 1 \rangle, \ldots, \langle x_m, 1 \rangle, \langle y_1, 0 \rangle, \ldots, \langle y_m, 0 \rangle$$

where $x_i, y_i \in X$, $1 \leq i \leq m$. If C is a representation class over X and $c \in C$, define

$$cost_{MD}(c, S) = |\{\langle x_i, 1\rangle \in S : x_i \in neg(c)\}| + |\{\langle y_i, 0\rangle \in S : y_i \in pos(c)\}| + 1.$$

Thus, $cost_{MD}(c, S)$ is simply one more than the number of disagreements between the balanced sample S and the representation c. We now define the following optimization problem for C:

The Minimize Disagreements Problem for C (denoted $MD(C)$)
Input: Balanced sample S of X.
Output: Representation $c \in C$ such that $cost_{MD}(c, S)$ is minimized.

THEOREM 19. *Let C be a representation class over X. If there exists a polynomial-time algorithm A' for $MD(C)$ that outputs $h_{A'} \in C$ such that $cost_{MD}(h_{A'}, S)$ is at most r times the optimal cost, then C is learnable by C by an algorithm A that runs in time polynomial in $1/\epsilon$, $1/\delta$ and $\ln |C|$, and satisfies*

$$E_{MAL}^{poly}(C, A) \geq \frac{\epsilon}{8r}.$$

Conversely, is algorithm A learns C by C in polynomial time with error rate $E_{MAL}^{poly}(C, A) \geq \epsilon/r$, then there exists a polynomial-time algorithm A' for $MD(C)$ that outputs (with high probability) $h_{A'} \in C$ such that $cost_{MD}(h_{A'}, S)$ is at most $2r$ times the optimal cost.

Proof. Let S be a balanced sample of X, and let A' be an approximation algorithm for $MD(C)$ such that the output $h_{A'}$ satisfies $cost_{MD}(h_{A'}, S) \leq r \cdot opt_{MD}(S)$, where

$$opt_{MD} = \min_{h \in C}(cost_{MD}(h, S)).$$

Let $\beta = \epsilon/8r$. To learn C by C in polynomial time with error rate β, we take m random examples $x_1, \ldots, x_m$ from the oracle POS_{MAL}^{β} and m random examples $y_1, \ldots, y_m$ from the oracle NEG_{MAL}^{β}, where m is as in the statement of Theorem 6. Let S be the balanced sample consisting of the x_i and y_j. Now with probability at least $1 - \delta$, the target representation $c \in C$ disagrees with fewer than $4\beta m$ elements of S by Fact CB2, so $opt_{MD}(S) \leq 4\beta m$ with high probability. Thus, algorithm A', when given S as input, will satisfy $cost_{MD}(h_{A'}, S) \leq r(4\beta m) = (\epsilon/2)m$. This implies that $h_{A'}$ can disagree with at most a fraction $\epsilon/2$ of the x_i and at most a fraction $\epsilon/2$ of the y_i. By Theorem 6, $h_{A'}$ is an ϵ-good hypothesis with high probability.

For the other direction, we use an algorithm A for learning C by C with $\beta = \epsilon/r$ to obtain an approximation algorithm for $MD(C)$ as follows: Given the balanced sample S, let $h_{opt} \in C$ be such that $cost_{MD}(h_{opt}, S) = opt_{MD}(S)$ and assume without loss of generality that $m/r \geq opt_{MD}(S)$ (otherwise *any* hypothesis has cost at most $2r$ times the optimal). Define

$$c_0 = \max\{|\{x_i \in S : x_i \in neg(h_{opt})\}|, |\{y_i \in S : y_i \in pos(h_{opt})\}|\}.$$

Note that $opt_{MD}(S) \geq c_0 \geq opt_{MD}(S)/2$. Now let I^+ be the uniform distribution over the x_i, and let I^- be the uniform distribution over the y_i. Then I^+ and I^- can be regarded as induced distributions for h_{opt} with error rate $\beta' = c_0/m$. I^+ is induced by the joint behavior of the uniform distribution D^+ over $\{x_i \in S : x_i \in pos(h_{opt})\}$, and an adversary that draws a point uniformly from $\{x_i \in S : x_i \in neg(h_{opt})\}$; I^- can be decomposed over the y_i in a similar fashion.

Algorithm A' runs algorithm A many times, starting with accuracy parameter $\epsilon = 1$, and drawing from I^+ on each call to POS_{MAL}^{β} and from I^- on each call to NEG_{MAL}^{β}. Note that if h_A is an ϵ-good hypothesis with respect to D^+ and D^-, then we have $cost_{MD}(h_A, S) \leq 2\epsilon m + opt_{MD}(S)$. After each run, A' divides ϵ by 2. On some run of A, $\epsilon/r \leq c_0/2m$, and for this run we have $cost_{MD}(h_A, S) \leq (r + 1)opt_{MD}(S) \leq 2r opt_{MD}(S)$, as desired. □

The first direction of this equivalence is also given by Blumer et al. [5]. Note that this equivalence as it is stated is *representation based* in the sense that it relies on the learning algorithm representing its hypothesis as a monomial. With more technical definitions for the problem $MD(C, H)$, we can in fact give a straightforward generalization of Theorem 19 for the problem of learning C by H in the presence of malicious errors, giving an equivalent optimization problem. In addition to simplifying the analysis of learning with errors in the distribution-free model—we only need to look at the equivalent optimization problem—these results allow us to weaken our restrictions on the adversary generating the errors. In particular, since there is no guarantee in the Minimize Disagreements problem on how the errors in the input sample are generated, it can be shown that the adversary gains no power by being allowed to see all coin flips of the learning algorithm, and all examples to be received by the learning algorithm *before* he generates the errors. This allows our model to incorporate faults such as *error bursts*, where all examples are in error for a short amount of time.

Tables 1 and 2 summarize some of the results in this paper.

TABLE 1
Summary of general upper bounds on the optimal error rates for the malicious and noise models. We denote by $t(C)$ the largest value of t such that C is (positive or negative) t-splittable.

	$E_{MAL,+}(C)$ and $E_{MAL,-}(C)$	$E_{MAL}(C)$	$E_{CN,+}(C)$ and $E_{CN,-}(C)$	$E_{CN}(C)$
Upper bound on the optimal error rate	$\epsilon/(t(C)-1)$	$\epsilon/(1+\epsilon)$	$\epsilon/(1+\epsilon), \epsilon/(t(C)-1)$	1/2 [1]

TABLE 2
Summary of upper and lower bounds on the optimal polynomial time error rate (malicious and noise models) for the classes of monomials M_n, monomials of length s M_n^s, and symmetric functions SF_n.

Class C	$E^{poly}_{MAL,+}(C)$ $E^{poly}_{MAL,-}(C)$ and $E^{poly}_{CN,+}(C)$ $E^{poly}_{CN,-}(C)$	$E^{poly}_{MAL}(C)$	$E^{poly}_{CN}(C)$
Upper bound		$O(\epsilon)$	
M_n	$\Theta(\epsilon/n)$		$\Theta(1)$
Lower bound	[24]	$\Omega(\ln(n/\epsilon)\epsilon/n)$	[1]
Upper bound		$O(\epsilon)$	
M_n^s	$\Theta(\epsilon/n)$	$\Omega((\epsilon/s)\ln((s/\epsilon)\ln(1+n/s))/\ln(1+n/s))$	$\Theta(1)$
Lower bound	[24]	$\Omega((\epsilon/s)(1/\log((s\ln n)/\epsilon)))$	[1]
Upper bound			
SF_n	$\Theta(\epsilon/n)$	$\Theta(\epsilon)$	$\Theta(1)$
Lower bound			

REFERENCES

[1] D. ANGLUIN AND P. LAIRD, *Learning from noisy examples*, Mach. Learning, 2 (1988), pp. 343–370.

[2] D. ANGLUIN AND L. G. VALIANT, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[3] A. BLUM, *Learning in an infinite attribute space*, Proceedings of the 22nd ACM Symposium on the Theory of Computing, 1990, pp. 64–72.

[4] A. BLUMER, A. EHRENFEUCHT, D. HAUSSLER, AND M. WARMUTH, *Occam's razor*, Inform. Process. Lett., 24 (1987), pp. 377–380.

[5] ———, *Learnability and the Vapnik–Chervonenkis dimension*, J. Assoc. Comput. Mach., 36 (1989), pp. 929–965.

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

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

[8] A. EHRENFEUCHT, D. HAUSSLER, M. KEARNS, AND L. G. VALIANT, *A general lower bound on the number of examples needed for learning*, Inform. and Comput., 82 (1989), pp. 247–261.

[9] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, Freeman, San Francisco, 1979.

[10] D. HAUSSLER, M. KEARNS, N. LITTLESTONE, AND M. WARMUTH, *Equivalence of models for polynomial learnability*, Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, San Mateo, CA, 1988, pp. 42–55, and University of California at Santa Cruz Information Sciences Department, Tech. Report UCSC-CRL-88-06, 1988.

[11] D. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–276.

[12] M. KEARNS AND M. LI, *Learning in the presence of malicious errors*, Proceedings of the 20th ACM Symposium on the Theory of Computing, 1988, pp. 267–280.

[13] M. KEARNS, M. LI, L. PITT, AND L. G. VALIANT, *On the learnability of Boolean formulae*, Proceedings of the 19th ACM Symposium on the Theory of Computing, 1987, pp. 285–295.

[14] M. KEARNS AND L. PITT, *A polynomial-time algorithm for learning k-variable pattern languages from examples*, Proceedings of the 1989 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, San Mateo, CA, 1989, pp. 57–71.

[15] P. LAIRD, *Learning from Good and Bad Data*, Kluwer Academic Publishers, the Netherlands, 1988.

[16] N. LITTLESTONE, *Learning quickly when irrelevant attributes abound: A new linear threshold algorithm*, Mach. Learning, 2 (1988), pp. 245–318, and Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science, 1987, pp. 68–77.

[17] L. LOVASZ, *On the ratio of optimal integral and fractional covers*, Discrete Math, 13 (1975), pp. 383–390.

[18] R. NIGMATULLIN, *The fastest descent method for covering problems*, Proceedings of a Symposium on Questions of Precision and Efficiency of Computer Algorithms, Kiev, 1969. [In Russian.]

[19] R. RIVEST, *Learning decision lists*, Mach. Learning, 2 (1987), pp. 229–246.

[20] R. SCHAPIRE, *On the strength of weak learnability*, Proceedings of the 30th IEEE Symposium on the Foundations of Computer Science, 1989, pp. 28–33.

[21] G. SHACKELFORD AND D. VOLPER, *Learning k*-DNF *with noise in the attributes*, Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, San Mateo, CA, 1988, pp. 97–105.

[22] R. SLOAN, *Types of noise in data for concept learning*, Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, San Mateo, CA, 1988, pp. 91–96.

[23] L. G. VALIANT, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[24] ———, *Learning disjunctions of conjunctions*, Proceedings of the 9th International Joint Conference on Artificial Intelligence, 1985, pp. 560–566.

[25] V. N. VAPNIK AND A. YA. CHERVONENKIS, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.

[26] K. VERBEURGT, *Learning* DNF *under the uniform distribution in quasi-polynomial time*, Proceedings of the 1990 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 838–856, August 1993

SMALL-BIAS PROBABILITY SPACES: EFFICIENT CONSTRUCTIONS AND APPLICATIONS*

JOSEPH NAOR† AND MONI NAOR‡

Abstract. It is shown how to efficiently construct a small probability space on n binary random variables such that for every subset, its parity is either zero or one with "almost" equal probability. They are called ϵ-biased random variables. The number of random bits needed to generate the random variables is $O(\log n + \log \frac{1}{\epsilon})$. Thus, if ϵ is polynomially small, then the size of the sample space is also polynomial. Random variables that are ϵ-biased can be used to construct "almost" k-wise independent random variables where ϵ is a function of k.

These probability spaces have various applications:

1. Derandomization of algorithms: Many randomized algorithms that require only k-wise independence of their random bits (where k is bounded by $O(\log n)$), can be derandomized by using ϵ-biased random variables.
2. Reducing the number of random bits required by certain randomized algorithms, e.g., verification of matrix multiplication.
3. Exhaustive testing of combinatorial circuits. The smallest known family for such testing is provided.
4. Communication complexity: Two parties can verify equality of strings with high probability exchanging only a logarithmic number of bits.
5. Hash functions: A polynomial sized family of hash functions such that with high probability the sum of a random function over two different sets is not equal can be constructed.

Key words. randomized algorithms, derandomization, VLSI circuit testing, discrepancy

AMS subject classifications. 60C05, 60E15, 94C12, 68Q25, 68Q22, 68R10

1. Introduction.

Randomness plays a significant role in computer science. However, it is often desirable to reduce the amount of randomness required. The purpose of this paper is to construct small probability spaces that approximate larger ones. Let $x_1, \ldots, x_n$ be $\{0, 1\}$ Bernoulli random variables and let Ω be the probability space associated with them. If the random variables are independent, then Ω contains all 2^n possible assignments. Our goal is to construct a much smaller probability space that will behave similarly to Ω in certain respects. Such small probability spaces have proved to be very useful.

One of the main approaches taken by previous researchers to reduce the size of the sample space was to require only limited independence among the random variables (as opposed to full independence). Our approach is different; it is based on the equivalence of the following two conditions: (See [18] and [52].)

1. The random variables are independent and for all i, $\text{Prob}[x_i = 0] = \text{Prob}[x_i = 1]$.
2. For every subset $S \subseteq \{1, \ldots, n\}$, it is equally likely that the parity of the subset (i.e., the number of "ones") is either zero or one.

We are going to relax the second condition and construct a probability distribution such that for every subset $S \subseteq \{1, \ldots, n\}$, it is "almost" equiprobable that the parity of the subset is zero or one. To be more precise, we require that for every subset S,

$$\left|\text{Prob}\left[\sum_{i \in S} x_i = 0\right] - \text{Prob}\left[\sum_{i \in S} x_i = 1\right]\right| \le \epsilon.$$

*Received by the editors April 1, 1990; accepted for publication (in revised form) April 1, 1992. A preliminary version of this paper appeared in the Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, 1990, pp. 213–223.

†Computer Science Department, Technion, Haifa 32000, Israel. Most of this work was performed while the author was at the Computer Science Department, Stanford University, Stanford, California and was supported by Office of Naval Research contract N00014-88-K-0166.

‡Department of Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Most of this work was performed while the author was at the IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120.

Vazirani [52] called this quantity the *bias* of the subset S. The cardinality of the sample space we construct is $2^{O(\log \frac{1}{\epsilon}+\log n)}$. Hence, if ϵ is typically polynomially small, then the size of the sample space is also polynomial.

We also define random variables that are k-wise ϵ-biased. For them, only the bias of subsets that are smaller than k is guaranteed to be bounded by ϵ. We present a more efficient construction for such random variables: The logarithm of the cardinality of the sample space is $O(\log k + \log\log n + \log \frac{1}{\epsilon})$. In §5 we show how the kth moment of the sum of k-wise ϵ-biased random variables can be bounded via the kth moment of the binomial distribution on uniform independent Bernoulli variables. This is an important tool for analyzing the behavior of the distribution of the sum.

We can use k-wise ϵ-biased random variables to construct k-wise δ-dependent random variables, i.e., variables such that the variation distance between the distribution of any subset of k variables and the uniform distribution (on k variables) is at most δ. Their construction is described in §4.

The ϵ-biased random variables are constructed in three stages. In the first stage, we construct a sample space $\mathcal{F}$ of random variables such that the bias of every subset S is bounded by some constant. Sampling from $\mathcal{F}$ requires $O(\log n)$ random bits. In the second stage, $\mathcal{F}$ is sampled l times (not necessarily independently), where l depends on ϵ. In the third stage, the ϵ-biased random variables are generated by picking a linear combination of the assignments sampled in the second stage. Constructing the probability space is described in §3.

Another interpretation of our result is via Fourier transforms. (See §2 for precise definitions.) For the uniform distribution, all the coefficients of its Fourier transform are zero, except for the free coefficient. For an ϵ-biased distribution, the absolute value of each coefficient is at most $\frac{\epsilon}{2^n}$.

Derandomizing algorithms has attracted much attention in recent years. For the purpose of derandomization, our distribution can replace the uniform one in many cases, so as to allow an exhaustive search for a good point in a polynomial-size sample space. We exemplify this by providing a *polynomial* size sample space for the set balancing problem; this also yields an NC^1 algorithm. A previous approach to derandomizing the set balancing problem [13], [39] was to first construct an $n^{O(\log n)}$ sample space and then conduct a binary search for a good point. As a result, their time bounds are worse. Another problem we address is finding a heavy codeword in a linear code. Using ϵ-biased random variables, we provide the first NC algorithm to the problem.

Another application of our probability distribution is for reducing the number of random bits required for several randomized algorithms. Karp and Pippenger [32] suggested that random bits can be viewed as a resource (just as time and space) which is best to use as little as possible. One motivation to consider randomness as a resource is practical. Random bits are hard to produce and devices that generate them, such as Geiger counters and Zener diodes, are slow. Another reason is from the complexity theoretic point of view: To provide a full scale of options between an algorithm that is completely deterministic, and a randomized algorithm that consumes many bits.

Examples of previous work in reducing the number of random bits in randomized algorithms are [2], [11], [20], [28], [32], [33], [43], [47], [49]. Karloff and Raghavan [33], for example, studied several randomized algorithms for selection and sorting and showed that they can be run successfully when only $O(\log n)$ random bits are available.

In §7 we describe how to reduce the number of random bits for three problems. The first one is matrix multiplication verification. Given three $n \times n$ matrices A, B, and C, how can we verify that $A \cdot B = C$ without resorting to matrix multiplication? We show how to do that in $O(n^2)$ time using $O(\log n)$ random bits, thus improving on a previous algorithm of [26] that required $O(n)$ random bits. We next show how Adi Shamir's Boolean matrix multiplication algorithm can be implemented by using $O(\log n)$ random bits without an increase in the probability of error. The third problem is verifying n equalities of the form $a^{x_i} = y_i \bmod p$ where p is a prime. We show how to do that using only $O(\log n)$ random bits and n multiplications, instead of $n \log p$ multiplications needed for testing each equality separately.

In §8 we show how to apply our construction to generate fault-diagnostic tests for combinatorial circuits. A well-known problem in that area is constructing a small collection of assignments to inputs of a circuit such that, for any k inputs, all possible configurations appear. Using k-wise δ-dependent probability spaces, we can get the best explicit constructions. These are optimal up to the constant factor in the exponent.

Our techniques can be used to minimize the communication complexity of protocols for testing equality of two strings, while achieving a very low probability of error. Similarly, the techniques can be applied to construct a small family of hash functions with the property that summing the hash function over different sets yields a different value with high probability. These two applications are discussed in §9.

In §10 we briefly survey recent papers that have used the constructions described in this paper since it appeared in [42].

Independent of our work, Peralta [44] considered ϵ-bias probability spaces as well, and showed some applications to number theoretic algorithms. His construction is based on quadratic residues and Weil's theorem.

2. Preliminaries and definitions.

Let $\mathbf{x} = x_1, \ldots, x_n$ be $\{-1, 1\}$ random variables and D their joint probability distribution.

DEFINITION 2.1. The bias of a subset $S \subseteq \{1, \ldots, n\}$ for a distribution D is defined to be,

$$\text{bias}_D(S) = \left| \text{Prob}_D\left[\prod_{i \in S} x_i = -1\right] - \text{Prob}_D\left[\prod_{i \in S} x_i = 1\right] \right|.$$

DEFINITION 2.2. The variables $x_1, \ldots, x_n$ are ϵ-biased if for all S, $\text{bias}_D(S) \leq \epsilon$. They are said to be k-wise ϵ-biased if for all subsets S such that $|S| \leq k$, $\text{bias}_D(S) \leq \epsilon$.

Let U denote the uniform distribution and $D(S)$ the distribution D restricted to the subset S. The variation distance between two distributions D_1 and D_2 defined over the same probability space Ω is

$$||D_1 - D_2|| = \sum_{\omega \in \Omega} |D_1(\omega) - D_2(\omega)|.$$

DEFINITION 2.3. The variables $x_1, x_2, \ldots, x_n$ are defined to be k-wise δ-dependent if for all subsets S such that $|S| \leq k$,

$$||U(S) - D(S)|| \leq \delta.$$

A similar definition was made by Ben-Nathan [12].

The set $f : \{-1, 1\}^n \to \Re$ of real functions on the n-dimensional cube forms a 2^n-dimensional real vector space. The inner product of two functions f and g is defined as

$$< f, g >= \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)g(x).$$

The basis of this vector space is given by the following family of functions, the characters of Z_2^n. Define for each subset $S \subseteq \{1 \ldots n\}$,

$$\chi_S(x_1, \ldots, x_n) = \prod_{i \in S} x_i.$$

The above basis has the following properties:

1. For all $S \subseteq \{1 \ldots n\}$, the family $\{\chi_S\}$ forms an orthonormal basis: If $S_1 \neq S_2$, then $< \chi_{S_1}, \chi_{S_2} >= 0$, and for every S, $< \chi_S, \chi_S >= 1$.
2. For every S_1, S_2: $\chi_{S_1} \cdot \chi_{S_2} = \chi_{S_1 \Delta S_2}$, where $S_1 \Delta S_2$ is the symmetric difference of S_1 and S_2.

The Fourier transform of a function f is its expansion as a linear combination of the χ_S's. Every function has a unique expression and the coefficients in this expression are called the Fourier coefficients, denoted by $\hat{f}(S)$ (for $S \subseteq \{1 \ldots n\}$). Hence, $f = \sum_S \hat{f}(S)\chi_S$.

Since the family χ_S forms an orthonormal basis, Fourier coefficients are found via:

$$\hat{f}(S) =< f, \chi_S > .$$

For a probability distribution D, $D = \sum_S \hat{D}(S)\chi_S$. The following theorem was proved by Diaconis and Shahshahani (see [22, Lem. 1, p. 24]). We provide a proof for the sake of completeness. (A slightly weaker bound was actually proved by Vazirani [52]; it motivated us to consider ϵ-biased probability spaces.)

THEOREM 2.1. *Let $x_1, \ldots, x_n$ be $\{-1, 1\}$ random variables and let D be their joint probability distribution. Then,*

$$||D - U|| \leq 2^n \cdot \left(\sum_{S \subseteq \{1 \ldots n\}} \hat{D}(S)^2 \right)^{1/2} = \left(\sum_{S \subseteq \{1 \ldots n\}} bias_D^2(S) \right)^{1/2} .$$

Proof. We first evaluate $\hat{D}(\emptyset)$,

$$\hat{D}(\emptyset) =< D, \chi_\emptyset >= \sum_{x \in \{-1,1\}^n} \frac{D(x)}{2^n} = \frac{1}{2^n}.$$

The square of the variation distance is,

$$||D - U||^2 = \left(\sum_{x \in \{-1,1\}^n} \left| D(x) - \frac{1}{2^n} \right| \right)^2 .$$

By the Cauchy–Schwarz inequality,

$$\left(\sum_{x \in \{-1,1\}^n} \left| D(x) - \frac{1}{2^n} \right| \right)^2 \leq 2^n \cdot \sum_{x \in \{-1,1\}^n} \left(D(x) - \frac{1}{2^n} \right)^2$$

$$= 2^n \cdot \sum_{x \in \{-1,1\}^n} \left(\sum_S \hat{D}(S) \cdot \chi_S(x) - \frac{1}{2^n} \right)^2 .$$

Since $\hat{D}(\emptyset) = \frac{1}{2^n}$,

$$2^n \cdot \sum_{x \in \{-1,1\}^n} \left(\sum_S \hat{D}(S) \cdot \chi_S(x) - \frac{1}{2^n} \right)^2 = 2^n \cdot \sum_{x \in \{-1,1\}^n} \left(\sum_{S \neq \emptyset} \hat{D}(S) \cdot \chi_S(x) \right)^2 =$$

$$2^n \cdot \sum_{x \in \{-1,1\}^n} \sum_{S_1 \neq \emptyset} \sum_{S_2 \neq \emptyset} \hat{D}(S_1)\hat{D}(S_2)\chi_{S_1}(x)\chi_{S_2}(x) =$$

$$2^n \cdot \sum_{S_1 \neq \emptyset} \sum_{S_2 \neq \emptyset} \hat{D}(S_1)\hat{D}(S_2) \sum_{x \in \{-1,1\}^n} \chi_{S_1}(x)\chi_{S_2}(x) =$$

$$2^{2n} \cdot \sum_{S_1 \neq \emptyset} \sum_{S_2 \neq \emptyset} \hat{D}(S_1)\hat{D}(S_2) < \chi_{S_1}, \chi_{S_2} > .$$

If $S_1 \neq S_2$, then $<\chi_{S_1}, \chi_{S_2}>= 0$; otherwise, $<\chi_{S_1}, \chi_{S_2}>= 1$. Hence,

$$2^{2n} \cdot \sum_{S_1 \neq \emptyset} \sum_{S_2 \neq \emptyset} \hat{D}(S_1)\hat{D}(S_2) < \chi_{S_1}, \chi_{S_2} > = 2^{2n} \cdot \sum_S \hat{D}(S)^2,$$

which implies that,

$$||D - U|| \leq 2^n \cdot \left(\sum_{S \subseteq \{1 \ldots n\}} \hat{D}(S)^2 \right)^{1/2}.$$

To complete the proof, recall that the definition of the bias of a subset S is,

$$\text{bias}_D(S) = |\text{Prob}_D[\chi_S(x) = -1] - \text{Prob}_D[\chi_S(x) = 1]| =$$
$$\left| \sum_{x \in \{-1,1\}^n} D(x)\chi_S(x) \right| = 2^n | < D, \chi_S > | = |2^n \hat{D}(S)|$$

and hence,

$$2^n \cdot \left(\sum_{S \subseteq \{1 \ldots n\}} \hat{D}(S)^2 \right)^{1/2} = \left(\sum_{S \subseteq \{1 \ldots n\}} \text{bias}_D^2(S) \right)^{1/2}. \qquad \square$$

COROLLARY 2.2. *If the random variables $x_1, \ldots, x_n$ are ϵ-biased with respect to a distribution D, then they are also k-wise δ-dependent, for $\delta = 2^{k/2} \cdot \epsilon$.*

A *binary linear code* C is a linear subspace of $\{0, 1\}^m$. If C has dimension n then C is called an $[m, n]$ code and it transforms words of length n into codewords of length m. A *generator matrix* G for a linear code C is an $n \times m$ matrix for which the rows are a basis of C. If G is a generator matrix for C, then the code can be defined as

$$C = \{aG \mid a \in \{0, 1\}^n\}.$$

The distance between two codewords is defined as their *Hamming* distance. $C[m, n, \beta]$ denotes an $[m, n]$ code C for which the distance between any two codewords is at least $\beta \cdot m$. The *weight* of a codeword is the number of nonzero symbols that it contains. For more details on linear codes the reader is referred to [40].

3. Constructing the probability distribution. In this section we show how to construct a probability space Ω of $\{0, 1\}$ random variables $\mathbf{x} = x_1, \ldots, x_n$ which are ϵ-biased. The cardinality of the sample space will be $2^{O(\log \frac{1}{\epsilon} + \log n)}$. The joint probability distribution of the variables is denoted by D. We define the function $\chi'_S(x_1, \ldots, x_n)$ for $\{0, 1\}$ random variables as follows. For each subset $S \subseteq \{1 \ldots n\}$,

$$\chi'_S(x_1, \ldots, x_n) = \sum_{i \in S} x_i \pmod 2.$$

The construction consists of three stages:

1. A polynomial size family $\mathcal{F}$ of $\{0, 1\}^n$ vectors is generated with the following property. Let r be a vector chosen from $\mathcal{F}$ uniformly at random. For all subsets $S \subseteq \{1, \ldots, n\}$,

$$\text{Prob}[\chi'_S(r) = 1] \geq \beta$$

where β is some constant. Constructing such a family is discussed in §3.1.

2. The vectors $r_1, \dots, r_l$ are sampled from $\mathcal{F}$ (not necessarily independently, but via a Markov process) such that for all subsets $S \subseteq \{1, \dots, n\}$,

$$\text{Prob}[\text{For all } i,\ 1 \le i \le l,\ \chi'_S(r_i) = 0] \le \epsilon.$$

The value of l will turn out to be $O(\log \frac{1}{\epsilon})$. Sampling the family $\mathcal{F}$ is discussed in §3.2.

3. The assignment to the random variables $x_1, \dots, x_n$ is a combination of the vectors sampled at the previous stage. Let $\vec{a} = (a_1, \dots, a_l)$ be chosen uniformly at random from $\{0, 1\}^l$. Then,

$$\mathbf{x} = \sum_{i=1}^{l} a_i r_i.$$

In §3.3 we discuss how to choose $\vec{a}$ so that $\mathbf{x}$ is ϵ-biased.

3.1. Constructing the family $\mathcal{F}$. We will need a family $\mathcal{F}$ with the above properties for constructing ϵ-biased probability spaces and also for other applications as well. (See §7.) For the latter applications, we extend the requirements from $\mathcal{F}$ to any ring: Given a vector v of elements in the ring, a vector r such that $< v \cdot r > \ne 0$ is called a *distinguisher* with respect to v. The goal is to find a small collection of vectors (the family $\mathcal{F}$) such that for any nonzero vector v, if r is chosen at random, then $\text{Prob}[< v \cdot r > \ne 0] \ge \beta$. If the ring is GF[2], this requirement is exactly the one mentioned above, where v is the characteristic vector of the subset S. Henceforth, we describe the construction for GF[2]. It can easily be generalized for any ring by substituting 1 by some nonzero element of the ring.

We present two methods for constructing the family $\mathcal{F}$. The first method (§3.1.1) can be applied to any ring, whereas the second one (§3.1.2) is applicable only to GF[2]. Another advantage of the first method is that computing the value of a random variable $x_i \in \mathbf{x}$ can be done in $O(1)$ operations on words of length $\log n$. On the other hand, the second method provides a general context, i.e., *linear codes*.

PROPOSITION 3.1. *Suppose that r is a vector chosen uniformly at random from* $\{0, 1\}^n$. *Then for all vectors $v \in \{0, 1\}^n$, $v \ne \vec{0}$, Prob$[< v \cdot r > \ne 0] \ge \frac{1}{2}$.*

Proof. Let j denote the number of nonzero entries in v. The number of different vectors r such that $v \cdot r = 0$ is at most $2^{j-1} \cdot 2^{n-j} = 2^{n-1}$, whereas the number of distinct choices for r is 2^n. Hence, the probability that $v \cdot r = 0$ is at most $\frac{1}{2}$. $\square$

Unfortunately, this method of generating a distinguisher requires n random bits. Hence, our aim is to show that a distinguisher can be generated with probability at least β using only $O(\log n)$ random bits. This will guarantee that the size of $\mathcal{F}$ is polynomial. Note that $\mathcal{F}$ must contain at least n vectors (potential distinguishers). Otherwise, the rank of the matrix whose columns are the distinguishers is less than n.

For our purposes, a collection $\mathcal{F}$ of vectors has to be constructed such that:

1. The size of $\mathcal{F}$ is "small."
2. It is "easy" to sample uniformly from $\mathcal{F}$.
3. Given any nonzero vector v, a constant fraction of the vectors in $\mathcal{F}$ are distinguishers with respect to v.

3.1.1. Constructing a small set of distinguishers. A natural approach to the problem of reducing the number of random bits in a randomized algorithm is to show that limited independence of the random variables suffices to assure a high probability of success. (See, e.g., [5], [36], [38].) However, in our case, it is not clear from the proof of Proposition 3.1 how many vectors r remain distinguishers with respect to the vector v when the entries of R

are not completely independent. Moreover, an example can be constructed in which, if the entries of r are chosen pairwise independently, no distinguisher will be generated.

Though limited independence is not sufficient for our purposes, we make use of it in two ways: One, suggested in [17], is that if we sample a universe looking for elements of some fixed subset, and if the expected number of elements we hit is greater than 1, then by making our choices only pairwise independent, we are not decreasing the chances of hitting an element of the subset by much. The other is that if the vector v has at most c nonzero entries, then the elements of r can be chosen c-wise independent, and with probability at least $\frac{1}{2}$ r is a distinguisher.

We now describe how the above-mentioned difficulties for generating distinguishers can be overcome. In what follows we will need n random variables such that:

1. Each random variable is uniformly distributed in $\{1, \dots n\}$.
2. Every subset of the random variables of cardinality at most c is independent.

There are known methods of generating such random variables that use only $O(c \log n)$ random bits. (See [38], [5], [17].)

We first assume that l, the precise number of nonzero elements in v, is known to be in the range $[k \dots 2k-1]$. A two-step process is applied.

1. The vector v is replaced by a new vector $v' = (v'_1, \dots v'_n)$ that contains only c (for some constant) nonzero elements. (Any nonzero element of v' is also nonzero in v.)
2. Now, the elements of the vector r can be chosen c-wise independent, yet Proposition 3.1 still holds.

As we do not have direct access to the elements of v, we show instead how to emulate the above Step 1 with high probability. Let $u = (u_1, \dots, u_n)$ and $w = (w_1, \dots, w_n)$ be two random vectors such that:

1. The entries of u are c-wise independent (c is a constant whose value will be specified later) where for all $1 \le i \le n$, $\text{Prob}[u_i = 0] = \text{Prob}[u_i = 1] = \frac{1}{2}$.
2. The entries of w are pairwise independent where for all $1 \le i \le n$, $\text{Prob}[w_i = 1] = \frac{2}{k}$. (If $k = 1$, then $\text{Prob}[w_i = 1] = 1$.)

We can assume, without loss of generality, that $k|n$. To generate the vector w we generate n random variables $z_1, z_2, \dots, z_n$ that are pairwise independent, and each is uniformly distributed in $\{1, \dots, n\}$. We then set w_i to 1 if $z_i \le \frac{2n}{k}$.

Let us now define the random vector $r = (r_1, \dots, r_n)$ that will be used as a distinguisher. For all $1 \le i \le n$,

$$r_i = \begin{cases} 1 & \text{if } u_i = 1 \text{ and } w_i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

LEMMA 3.2. *The above vector r is a distinguisher with probability at least $\frac{1}{4}$ for any vector v for which l, the number of nonzero elements, is in the range $[k, 2k]$.*

Proof. Let us define the vector v' (from Step 1): For all $1 \le i \le n$,

$$v'_i = \begin{cases} 1 & \text{if } v_i = 1 \text{ and } w_i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

It is clear that the lemma will follow if we show that the vector u is a distinguisher with respect to v' with probability at least $\frac{1}{4}$. To do that, it suffices to prove that v' will contain at least one nonzero element of v, and at most c nonzero elements of v with probability at least $\frac{1}{2}$. As the elements of the vector u are c-wise independent, the proof of Proposition 3.1 still holds when the number of nonzero elements is less than c.

Generating the vector v' can be thought of as a binomial random variable where each nonzero entry of v decides with probability p (to be specified later) whether it remains nonzero in v'. The random choices are pairwise independent. Let h be a random variable that denotes the number of nonzero elements in v'. It is well known that $E[h] = pl$ and $\mathrm{Var}[h] = p(1-p)l$. We chose the value of p such that $pk = 2$.

CLAIM. $\mathrm{Prob}[0 < h \leq 7] \geq \frac{1}{2}$.

We prove the claim by Chebyshev's inequality [25] which states that

$$\mathrm{Prob}[|X - E[X]| \geq \lambda] \leq \frac{\mathrm{Var}[X]}{\lambda^2}$$

where X is a random variable. It is enough to verify the claim in the two extreme cases when $l = k$ and $l = 2k$. Thus, substituting $\lambda = 2$ and $\lambda = 3$, we get that

$$\mathrm{Prob}[|h - pk| \geq 2] \leq \frac{pk(1-p)}{4} \leq \frac{1}{2},$$

$$\mathrm{Prob}[|h - 2pk| \geq 3] \leq \frac{2pk(1-p)}{9} \leq \frac{4}{9}.$$

Hence, we can choose $c = 2kp + 3 = 7$, and this is enough to insure success with probability at least $\frac{1}{2}$. □

We conclude that if the approximate number of nonzero entries in v is known, then $O(\log n)$ random bits suffice to insure high probability of success. What can we do if this is not known? We follow the above algorithm and construct $\log n$ collections $\mathcal{F}_1, \mathcal{F}_2, \ldots \mathcal{F}_{\log n}$, where $\mathcal{F}_i$ is generated under the assumption that the number of nonzero entries in v is between 2^{i-1} and 2^i. Lemma 3.1 implies that at least $\frac{1}{4}$ of the members of at least one collection will be distinguishers.

The same random bits can be used to sample the $\log n$ collections, and we obtain a set of $\log n$ vectors $r^1, r^2 \ldots r^{\log n}$ such that at least one of them is a distinguisher with probability at least $\frac{1}{4}$. Instead of testing each vector separately, we can generate from them a single vector that is a distinguisher with probability at least $\frac{1}{8}$.

Let S be a subset of $\{1, \ldots, \log n\}$ which is chosen uniformly at random and let r be defined by

$$r = \sum_{i \in S} r^i.$$

LEMMA 3.3. *The vector r is a distinguisher with respect to v with probability at least $\frac{1}{8}$.*

Proof. We need the following claim.

CLAIM. *Let v be a vector such that the vector r_1 is a distinguisher and the vector r_2 is not a distinguisher with respect to v. Then the vector $r_1 + r_2$ is a distinguisher with respect to v.*

Proof. $(r_1 + r_2) \cdot v = r_1 \cdot v + r_2 \cdot v = r_1 \cdot v \neq 0$.

Let the number of nonzero entries in v be between 2^{j-1} and 2^j and assume that r^j is a distinguisher. This will happen with probability at least $\frac{1}{4}$. If $\sum_{i \in S - \{j\}} r^i$ is not a distinguisher, then with probability $\frac{1}{2}$, $j \in S$, and according to the above claim, r will be a distinguisher. Otherwise, again with probability $\frac{1}{2}$, $j \notin S$, and according to the above claim, r will be a distinguisher. □

To summarize, we describe the algorithm to generate a random vector r. We assume that n is a power of 2.

1. Generate the following random variables:

   (a) $z_1, z_2, \ldots, z_n$: n random variables that are pairwise independent and uniformly distributed in $\{1, \ldots, n\}$.
   (b) $u = (u_1, \ldots, u_n)$ a vector whose entries are seven-wise independent and uniformly distributed in $\{0, 1\}$.
   (c) A random subset $S \subseteq \{1, \ldots, \log n\}$.
2. For each $1 \leq i \leq n$, compute $j_i = \log n - \max\{j | 2^j < z_i\}$.
3. For $1 \leq l \leq \log n$, compute $c_l = |S \bigcap \{1, \ldots, l\}| \cdot 1$. (This is scalar multiplication in the ring, e.g., in GF[2] it is $|S \bigcap \{1, \ldots, l\}| \bmod 2$.)
4. For each $1 \leq i \leq n$, compute:

$$r_i = \begin{cases} c_{j_i} & \text{if } u_i = 1\ , \\ 0 & \text{otherwise.} \end{cases}$$

THEOREM 3.4. *The algorithm described above uses $O(\log n)$ random bits and generates a distinguisher with probability at least $\beta = \frac{1}{8}$. For all i, $1 \leq i \leq n$, the complexity of computing the value of the random variable x_i is $O(1)$ operations on words of size $O(\log n)$.*

Proof. Steps 1(a), 1(b), and 1(c) require each $O(\log n)$ bits. (In fact, Steps 1(b) and 1(c) can be implemented recursively using $O(\log \log n)$ bits.) Given i, to compute r_i, we need to know the value of z_i, c_{j_i}, and u_i; that requires a constant number of operations. In Step 3, computing c_{j_i} requires counting the number of ones in a word of length $\log n$ that describes S. $\square$

3.1.2. A construction based on linear codes. Here we describe how to generate a family $\mathcal{F}$ of size $O(n)$ via linear codes due to Bruck (private communication). The construction works for GF[2]. Let C be a linear code. The weight of a codeword is defined to be the number of nonzero entries.

PROPOSITION 3.5. *The minimum distance of a linear code is equal to the minimum weight of a codeword.*

Suppose we have a linear code $C[n, m, \beta]$, i.e., it maps $\{0, 1\}^n$ words into $\{0, 1\}^m$ codewords and its minimum distance is $\beta \cdot m$. Linear codes for which $m = O(n)$ and β is some constant exist and are constructible. (For example, Justesen codes [30], [40].) Let G be the generator matrix of this code. For any nonzero $\{0, 1\}^n$ vector v, $v \cdot G$ contains at least βm nonzero entries (by the above proposition). Hence, if we choose a column in G uniformly at random, $\text{Prob}[< v \cdot r >= 1] \geq \beta$. The family $\mathcal{F}$ is the set of columns in G. The relation with linear codes holds in the other direction as well. Given a family $\mathcal{F}$, consider its members as columns of a generator matrix of a linear code. It follows from the proposition that the minimum distance of this code is $\beta|\mathcal{F}|$. Hence, the construction in §3.1.1 can be regarded as a linear code.

Given that good codes exist, what advantages does the first method have? The first method has the property that it works for more general cases where the function χ'_S is defined on any group, not just addition modulo 2. This will be used in §7.1. Another advantage is in computing a single entry of the sampled vector r. This is very simple in the first method (Theorem 3.1) whereas all known methods for using (traditional linear codes) are more complicated and require exponentiation.

Using our techniques we can actually enhance the error-correction of a linear code without decreasing the rate by much. This is further investigated in [6].

3.2. Sampling the family $\mathcal{F}$. The problem of obtaining the vectors $r_1, \ldots, r_l$ in stage 2 with the desired property can be abstracted in the following way. Suppose that there is a universe, and we wish to find a member in a certain subset $\mathcal{S}$ of it. (In our case it is the set of vectors for which $\chi_S = 1$.) Suppose also that we have a sampling algorithm that uses k

random bits and has probability β of picking a member of the desired set. By sampling l times independently, a set of l elements is generated, such that with probability greater than $1-\beta^{-l}$, at least one of them is a member of the desired set. A straightforward implementation would require kl random bits.

Several papers have addressed the question of achieving the probability error while requiring fewer random bits [1], [20], [28], [32], [47], [49]. We consider the method of [1] which is used by [20], [28]: For a graph $G = (V, E)$, consider any 1-1 correspondence $f : V \to \{0,1\}^k$, the different assignments to the random bits of the sampling algorithm. To generate the l samples, choose a random vertex of G and perform a random walk of length l. Each vertex in the random walk corresponds to a sample point. The number of random bits required is the sum of those needed for sampling one vertex, and those needed for performing the random walk.

Let λ_0 be the largest eigenvalue of G and $\bar{\lambda}$ be the eigenvalue of second largest value in G. Let α denote the percentage of vertices that are not members in $\mathcal{S}$. Cohen and Wigderson [21, Thm. 4.5] show that if G is a d-regular graph such that

$$2\alpha \le \left(\frac{\bar{\lambda}}{\lambda_0}\right)^2$$

then the probability that at least one of the l samples is a member of $\mathcal{S}$ is at least

$$1-\left(\frac{\sqrt{2}|\bar{\lambda}|}{\lambda_0}\right)^l.$$

Constructions for regular graphs of constant degree such that the second eigenvalue is bounded away from λ_0 are known [27], [29], [37]. For degree regular graphs of degree d, $\lambda_0 = d$ and the value of $\bar{\lambda}$ can be almost $2\sqrt{d}$. For a given expander G, let

$$\alpha_G = \frac{1}{2}\left(\frac{\bar{\lambda}}{\lambda_0}\right)^2$$

and let

$$l = \frac{\log\frac{1}{\epsilon}}{\log\left(\frac{\sqrt{2}|\bar{\lambda}|}{\lambda_0}\right)}.$$

In our case, β, the precentage of good vectors, is too small to apply the method directly and we need some initial amplification. To do that, each vertex would now correspond to a set of assignments to the random bits, such that the probability that at least one element associated with a randomly chosen vertex is a member of $\mathcal{S}$ is at least $1-\alpha_G$. For the purposes of this paper, this can be done by letting each vertex correspond to h independent samples where $h = \log_{1-\beta}\alpha_G$.

To conclude, the number of random bits used is $\log|\mathcal{F}| + l\cdot\log d$.

3.3. Combining the samples. We have to specify how to choose a linear combination $\vec{a}$ of the vectors $r_1,\ldots,r_l$, given that for any subset S, Prob[for all i, $\chi'_S(r_i) = 0] \le \epsilon$. The simplest way to select $\vec{a}$ is to choose it uniformly at random.

CLAIM 3.1. *The random variables $x_1,\ldots,x_n$ generated by choosing $\vec{a}$ uniformly at random from $\{0,1\}^l$ are ϵ-biased.*

Proof. Let S be any subset of $\{1...n\}$. If there exists a vector r_j among the vectors $r_1, \ldots, r_k$ such that $\chi'_S(r_j) = 1$, then by arguments similar to those of Lemma 3.2,

$$\text{Prob}_D[\chi'_S(x_1, \ldots, x_n) = 1] = \text{Prob}_D[\chi'_S(x_1, \ldots, x_n) = 0].$$

Since the probability of this happening is at least $1 - \epsilon$, $\text{bias}_D(S) \leq \epsilon$. □

The number of random bits required is l. To conclude, we have the following theorem.

THEOREM 3.6. *Generating n $\{0, 1\}$ random variables that are ϵ-biased can be done using $O(\log n + \log \frac{1}{\epsilon})$ random bits. Thus, the size of the sample space is $2^{O(\log n + \log \frac{1}{\epsilon})}$. Given the random bits, computing the value of a random variable can be done in time polylogarithmic in n.*

4. Generating k-wise Δ-dependent random variables. We are insured by Corollary 2.1 that if the random variables $x_1, \ldots, x_n$ are ϵ-biased, then they are k-wise δ-dependent for $\delta = 2^{\frac{k}{2}} \cdot \epsilon$. However, there is a more efficient construction. This can be done by combining our methods with those of [5] for generating k-wise independent variables.

Suppose we want to generate $\{0, 1\}$ uniform random variables $y_1, \ldots, y_n$ that are k-wise independent. Reference [5] suggests that this can be done by taking n vectors $L_1, \ldots, L_n$ of length h such that the vectors are k-wise linearly independent over GF[2]. If the vectors $L_1, \ldots, L_n$ are columns of the parity check matrix of BCH codes, then h is $\frac{k}{2} \log n$. Let R be a vector chosen uniformly at random from $\{0, 1\}^h$; for all i, $1 \leq i \leq n$, let $y_i = L_i \cdot R$. The number of random bits required for the construction is $k \log n$.

In order to improve on Corollary 2.1, instead of choosing R uniformly at random, suppose that the entries of R are ϵ-biased random variables.

LEMMA 4.1. *Let $y_1, \ldots, y_n$ be random variables generated by the above method, where the entries of R are ϵ-biased random variables. Then, $y_1, \ldots, y_n$ are k-wise ϵ-biased.*

Proof. Let $S \subseteq \{1, \ldots, n\}$ be a subset of cardinality at most k. We bound bias(S). For all $i \in S$, $y_i = L_i \cdot R$. Hence,

$$\sum_{i \in S} y_i = \sum_{i \in S} L_i \cdot R = R \cdot \sum_{i \in S} L_i = R \cdot M.$$

For $i \in S$, the vectors L_i are linearly independent, and hence $M \neq 0$ and $(S) \leq \epsilon$. □

The improvement over Corollary 2.1 in the cardinality of the sample space is that now we have decreased the number of ϵ-biased random variables from n to $k \log n$. Recall that random variables that are k-wise ϵ-biased, are also k-wise $2^{\frac{k}{2}} \cdot \epsilon$-dependent.

LEMMA 4.2. *The logarithm of the cardinality of the sample space needed for constructing k-wise δ-dependent random variables is $O(k + \log \log n + \log \frac{1}{\delta})$.*

5. A moment inequality. Basic tools in probabilistic analysis are moment inequalities [25] that bound the deviation of a random variable from its expected value. More specifically, let y be a random variable such that $E[y] = 0$, then,

$$\text{Prob}[|y| \geq \lambda] \leq \frac{E[|y|^k]}{\lambda^k}.$$

Let $b_1, \ldots, b_n$ be random variables that have a binomial distribution, i.e., they are independent and take their values from $\{-1, 1\}$ uniformly. The kth moment of their sum will be denoted by B_k, that is $B_k = E[|b_1 + \ldots + b_n|^k]$.

We formulate a moment inequality for the sum of $\{-1, 1\}$ random variables $(x_1, \ldots, x_n)$ that are k-wise ϵ-biased. We denote their sum by $S = \sum_{i=1}^n x_i$. This will be done by bounding the kth moment of S via the kth moment of the binomial distribution on n uniform independent Bernoulli variables, denoted by B_k. (We assume here, without loss of generality, that k is even.)

The kth moment of S contains n^k terms, where each term contains at most k variables. More specifically,

$$E[S^k] = E\left[\sum_{i_1 i_2 \ldots i_k} x_{i_1}x_{i_2}\ldots x_{i_k}\right] = \sum_{i_1 i_2 \ldots i_k} E\left[x_{i_1}x_{i_2}\ldots x_{i_k}\right].$$

Each term in the above summation is of the form $T_i = x_{i_1}^{p_1}x_{i_2}^{p_2}\ldots x_{i_r}^{p_r}$, such that $\sum_{j=1}^{r} p_j = k$. If a term T_i contains a variable whose power is odd, then in B_k its expectation is 0. However, in our case, it follows from the definition of k-wise ϵ-biased random variables (Definitions 2.2 and 2.3), that the expected value of T_i can be at most ϵ. If all the powers in a term T_i are even, then the expected value in both cases is the same and equal to 1. Hence, we have the following theorem.

THEOREM 5.1. *Let S and B_k be as above, then*

$$Prob[|S| \geq \lambda] \leq \frac{B_k + \epsilon \cdot n^k}{\lambda^k}.$$

6. Derandomization. The probability distribution D constructed in §3 can be used for derandomizing algorithms. A randomized algorithm $\mathcal{A}$ has a probability space (Ω, P) associated with it, where Ω is the sample space and P is some probability measure. We call a point $w \in \Omega$ a *good* point for some input instance I, if $\mathcal{A}(I, w)$ computes the correct solution. A derandomization of an algorithm means searching the associated sample space Ω for a good point w with respect to a given input instance I. Given such a point w, the algorithm $\mathcal{A}(I, w)$ is now a deterministic algorithm and it is guaranteed to find the correct solution. The problem faced in searching the sample space is that it is generally exponential in size.

In [36], [38], [5] the following strategy was suggested: Show that the n probabilistic choices of certain randomized algorithm are only required to be k-wise independent. Hence, a sample space of size $O(n^k)$ suffices. This sample space can be exhaustively searched for a good point (even in parallel) when k is a constant. A similar strategy can be used with ϵ-biased random variables. First, show that a randomized algorithm has a nonzero probability of success when the probabilistic choices are ϵ-biased. Then, conduct a search of the sample space associated with the variables to find a good point.

This scheme can in principle be applied to all the randomized algorithms for which the limited independence approach was applied. However, we do not necessarily get better algorithms. An attractive feature of our scheme is that random variables that are $\log n$-wise δ-dependent, for δ which is polynomially small, can be constructed with a *polynomial* sample space. Intuitively, this means that we can achieve "almost" $\log n$-wise independence with a polynomial sample space (as opposed to $n^{O(\log n)}$).

The k-wise ϵ-biased random variables are especially useful when the proof that a randomized algorithm is successful involves any moment inequality. In that case, we should compute what is the appropriate ϵ such that the error incurred leaves the probability of success nonzero. We exemplify this by showing how a RNC algorithm for the *set balancing* problem can be converted into an NC algorithm.

The second problem we address is finding a heavy codeword in a linear code (§6.1). Such a codeword can be found by a simple randomized algorithm. We show how to derandomize it in parallel. The importance of this problem is that it demonstrates that for certain problems it is important to bound the bias of all subsets and not just those of small cardinality.

6.1. Set balancing. The set balancing problem is defined as follows. A collection of subsets $\mathcal{S} = \{S_1, S_2, \cdots S_n\}$ defined over a base set $B = \{b_1, b_2, \cdots b_n\}$ is given such that the cardinality of each subset is δ. The output is a $\{-1, 1\}$ coloring of B into two sets. Let the two-coloring of B be denoted by $\mathbf{x} = (x_1, x_2, \cdots x_n)$. The discrepancy of a subset S_i, $\Delta(S_i, \mathbf{x})$, with respect to $\mathbf{x}$, is defined as $\sum_{j\in S_i} x_j$. The discrepancy of the set family $\mathcal{S}$ is defined as,

$$\Delta(\mathcal{S}, \mathbf{x}) = \max_{S_i} \Delta(S_i, \mathbf{x}).$$

Spencer [50], [9] showed that for each family $\mathcal{S}$, there exists a two-coloring $\mathbf{x}$ such that $\Delta(\mathcal{S}, \mathbf{x}) \leq 6\sqrt{n}$. This result is the best possible up to constant factors, but it has the drawback of being nonconstructive, i.e., does not even imply a probabilistic algorithm. Using the method of conditional probabilities (sequentially), Spencer devised a polynomial-time deterministic algorithm which guarantees a two-coloring $\mathbf{x}$ such that $\Delta(\mathcal{S}, \mathbf{x}) = O(\sqrt{n \log n})$. This was improved to $O(\sqrt{\delta \log n})$ by Raghavan [45]. For parallel algorithms we cannot guarantee as small a discrepancy as in the sequential case. However, we can come arbitrarily close to the sequential bounds by computing a τ-good coloring.

DEFINITION 6.1. A two-coloring $\mathbf{x}$ is τ-good for a set family $\mathcal{S}$ if, for $0 < \tau < \frac{1}{2}$,

$$\Delta(\mathcal{S}, \mathbf{x}) \leq \delta^{0.5+\tau}\sqrt{\log n}.$$

There is a simple RNC algorithm to find a τ-good coloring for any $\mathcal{S}$. Pick a random $\mathbf{x}$ uniformly at random from $\{0, 1\}^n$. It turns out that for any $\mathcal{S}$, the two-coloring is τ-good with sufficiently high probability. (See, for example, [34].) This algorithm was derandomized by [13], [39] by proving that $\log n$-wise independence suffices, and then showing how to conduct a binary search in a sample space of size $n^{O(\log n)}$ (the method of conditional probabilities). We present a direct derandomization which can be implemented in NC^1.

Assume that B is colored at random; [13], [39] prove the following lemma.

LEMMA 6.1. *Let $k = a \log n / \log \delta$ be even, where $a > \frac{1}{\tau}$, and let $\mathbf{x}$ be k-wise independent uniform $\{-1, 1\}$ random variables. Then, for any input set family $\mathcal{S}$, the probability that $\mathbf{x}$ is τ-good is nonzero.*

We sketch very briefly the proof idea. Let T_i denote the sum of the random variables in the subset S_i. The main tool used is the kth moment inequality. (See §5.) More specifically, [13], [39] show that for a given subset S_i,

$$\text{Prob}\left[\left|T_i - \frac{\delta}{2}\right| > \delta^{0.5+\tau}\sqrt{\log n}\right] \leq \frac{E[T_i^k]}{(\delta^{0.5+\tau}\sqrt{\log n})^k} < \frac{1}{\delta^{\tau k}} < \frac{1}{n}.$$

Summing over all subsets, we get that for any set family $\mathcal{S}$, the probability that a random k-wise coloring is τ-good is nonzero.

What happens when the random variables in $\mathbf{x}$ are k-wise ϵ-biased? We want to choose ϵ such that the probability that the discrepancy in a subset is large, is smaller than $\frac{1}{n}$. Substituting in Theorem 5.1, ϵ and k must be chosen such that

$$\frac{1}{\delta^{\tau k}} + \frac{\epsilon \delta^k}{(\delta^{0.5+\tau}\sqrt{\log n})^k} < \frac{1}{n}.$$

We choose $k = \frac{\log 2n}{\tau \log \delta}$, and the above inequality holds if

$$\epsilon < \frac{1}{2n^{1+\frac{1}{\tau}}}.$$

Hence, the sample space will be at most of cardinality $n^{O(\frac{1}{\tau})}$ and if τ is a constant, an exhaustive search of the sample space can be conducted. Both the construction and the search can be done in NC^1.

For the relationship between the set balancing problem and the lattice approximation problem [45], the reader is referred to [39].

6.2. Finding heavy codewords. Let $C[n, k, d]$ be a linear code. In this section we show how to find a codeword whose weight is at least as heavy as the expected weight of a word in C. We call such a codeword a *heavy* codeword. This problem is a natural generalization of the following problem: Given a graph G, find a cut that contains at least half the edges. It is well known that the set of cuts in a graph forms a linear subspace. Let G be the generator matrix of the linear code C. It is easy to see that the expected weight of a codeword in C is $\frac{n}{2}$: If x is chosen uniformly at random, then $E[\text{weight of } Gx] = \frac{n}{2}$.

The latter observation implies a straightforward randomized algorithm for finding a heavy codeword. This algorithm can be made deterministic via the method of conditional probabilities [50], [9] where the entries of x are determined one at a time. How can a heavy codeword be found in parallel? One possible approach for reducing the sample space is by choosing the entries of x to be only k-wise independent (for some $k < n$) and not mutually independent. The difficulty is that the probability of getting a heavy codeword may vanish.

If x is chosen from an ϵ-biased probability space, then $E[\text{weight of } Gx] \geq n(\frac{1-\epsilon}{2})$. If $\epsilon < \frac{1}{2n}$, then there must be a codeword x in the ϵ-biased probability space whose weight is at least $\frac{n}{2}$. This places the problem in NC for the first time. More important, it exhibits that ϵ-biased random variables are also needed as opposed to just k-wise ϵ-biased random variables.

7. Reducing the number of random bits. In this section we present two algorithms for which the number of random bits required can be reduced from linear to logarithmic.

7.1. Matrix multiplication verification. Suppose that three $n \times n$ matrices A, B, and C over an arbitrary ring are given; what is the complexity of verifying whether $A \cdot B = C$? Can this be done by avoiding matrix multiplication? This problem was first considered by Freivalds [26] who suggested a randomized algorithm of complexity $O(n^2)$, but it required n random bits. In this section we show how to implement it using only $O(\log n)$ random bits with no time penalty. Our results can readily be generalized to nonsquare matrices as well.

Let us first review Freivalds's algorithm [26]. Choose a random vector $\vec{r}$ such that each entry in r is picked uniformly at random to be zero or some fixed nonzero element of the ring. Test whether $r \cdot A \cdot B - r \cdot C = 0$. The complexity of applying this procedure is that of multiplying a matrix by a vector which is obviously $O(n^2)$. For the sake of completeness, we present a proof of his algorithm.

THEOREM 7.1. *Suppose that r is a vector chosen in the manner described above. Then with probability at least $\frac{1}{2}$, if $A \cdot B \neq C$, then also $r \cdot A \cdot B \neq r \cdot C$.*

Proof. Let v be a nonzero column vector of $A \cdot B - C$ and let j denote the number of its nonzero entries. The number of different vectors r such that $v \cdot r = 0$ is at most $2^{j-1} \cdot 2^{n-j} = 2^{n-1}$, whereas the number of distinct choices for r is 2^n. Hence, the probability that $v \cdot r = 0$ is at most $\frac{1}{2}$. □

It follows from the above proof that we can restrict ourselves to the following problem: Given a vector v, check whether it is identically zero, where the only operation permitted on the vector is taking its inner product with another vector. Recall from §3.1 that a vector r that verifies that a particular vector v is nonzero is called a *distinguisher* (with respect to v). Hence, reducing the number of random bits for the above problem is equivalent to generating a small collection of vectors $\mathcal{F}$, such that the inner product of a constant fraction of them with *any* nonzero vector is not zero.

The family $\mathcal{F}$ that is constructed in §3.1 has this property: With probability at least β, a vector r sampled uniformly will be a distinguisher. The number of random bits needed for sampling $\mathcal{F}$ is $O(\log n)$ and the complexity of the algorithm remains $O(n^2)$.

Notice that if the matrices are over an arbitrary ring, then the construction presented in §3.1.2 cannot be used. We can only use the one presented in §3.1.1.

7.2. Boolean matrix multiplication. In this section we show how Shamir's Boolean matrix multiplication algorithm (see [19, pp. 772–773]) can be implemented with few random bits without increasing the probability of error.

Let $A = (a_{ij})$ and $B = (b_{ij})$ be $n \times n$ Boolean matrices and suppose we would like to multiply them in the quasi ring $Q = (\{0,1\}, \vee, \wedge, 0, 1)$ by using the algorithms for fast matrix multiplications, e.g., Strassen's method. The difficulty is that these methods require that the matrix multiplication be carried out in a ring. This can be handled by working over a large field, but makes multiplication more expensive. Instead, Shamir suggested a randomized algorithm that takes advantage of the fact that the fast methods for matrix multiplication can be carried out in the ring $R = (\{0,1\}, \oplus, \wedge, 0, 1)$.

We briefly summarize Shamir's algorithm. Let $C = (c_{ij}) = AB$ in the quasi ring Q. Generate $A' = (a'_{ij})$ from A using the following randomized procedure:

- If $a_{ij} = 0$, then let $a'_{ij} = 0$.
- If $a_{ij} = 1$, then let $a'_{ij} = 1$ with probability $1/2$ and let $a'_{ij} = 0$ with probability $1/2$. The random choices for each entry are independent.

Let $C' = A'B$ in the ring R. The following lemma is immediate.

LEMMA 7.2. *If* $c_{ij} = 0$, *then* $c'_{ij} = 0$. *If* $c_{ij} = 1$, *then* $\text{Prob}[c'_{ij} = 1] \geq 1/2$.

As in the algorithm for verifying matrix multiplication, make the random choices to be ϵ-bias and get that the probability of error is at most $1/2 + \epsilon$.

To decrease the probability of failure, we will run the algorithm for $\log(n^2/\delta)$ choices of the matrix A'. Since the matrix C' has n^2 entries, the total probability of error is bounded from above by δ.

7.3. Simultaneous verification of exponentiation. Suppose that for some prime p and integer a we are given n pairs $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ and we want to verify whether the equality $a^{x_i} = y_i \bmod p$ is true for all $1 \leq i \leq n$. Fiat and Naor (personal communication) have suggested the following randomized algorithm:

1. Pick a random vector $r = (r_1, \ldots, r_n)$ where each r_i is chosen uniformly and randomly from $\{0, 1\}$.
2. Compute $t = \sum_{i=1}^{n} r_i \cdot x_i \bmod (p-1)$ and $m = \prod_{i \in S} y_i^{r_i} \bmod p$.
3. Test whether $a^t = m \bmod p$.

As in Freivalds's algorithm, if for any $1 \leq i \leq n$, $a^{x_i} \neq y_i$, then the above algorithm will detect it with probability at least $\frac{1}{2}$. The complexity of this algorithm is n multiplications, instead of $n \log p$ for checking each equality separately.

We can actually phrase the problem as that of finding a distinguisher with respect to a nonzero vector in the integer ring modulo $p-1$. Let z_i be such that $a^{z_i} = y_i$. Then, a distinguisher with respect to the vector w must be found, where $w_i = x_i - z_i \bmod (p-1)$. For that we can use the construction of §3.1 in a similar way to §7.1.

Note that the expected size of each entry in step 4 of the algorithm in §3.1.1 is $O(1)$ and, therefore, the expected number of multiplications remains $O(n)$.

Suppose that a and N are integers such that $(N, a) = 1$. The above procedure can be applied to verify n equalities of the type $x_i^a = y_i \bmod N$. This may be used for instance to check n given RSA equations [46] and thus amortize the cost of verifying signatures.

8. Vector sets for exhaustive testing. A problem that has received much attention in the fault diagnostic literature is that of generating a small set of vectors, $T \subset \{0, 1\}^n$, such that for any k-subset of indices $S = \{i_1, i_2, \ldots i_k\}$, the projection of T on the indices S contains all possible 2^k configurations. See [51] for a bibliography on the problem. Such a set of vectors is called (n, k) universal. The motivation for this problem is that such a test set allows exhaustive testing of a circuit where each component relies on at most k inputs. Alternatively, we can phrase the problem as searching for a collection of n subsets of a ground set which is as small as possible such that the intersection of any k subsets or their complement is nonempty. This was called k-independent by Kleitman and Spencer [35].

The best known bounds for constructing (n, k) universal sets are given in [51] and [3]. The connection between k-wise δ-dependent probability spaces and small (n, k)-universal sets is made in the next proposition.

PROPOSITION 8.1. *If Ω is a k-wise δ-dependent probability space for $\delta \leq 2^{-k}$, then Ω is also a (n, k)-universal set.*

Proof. If for a k-subset $i_1, i_2, \ldots i_k$, there is a $\{0, 1\}^k$ configuration which has probability 0 in Ω, then the distance from the uniform distribution of $x_{i_1}, x_{i_2}, \ldots x_{i_k}$ is at least $2^{-(k-1)} > \delta$. □

Combining the construction suggested in §3.1.2 with Lemma 4.2, we can construct a probability space Ω which is k-wise 2^{-k}-dependent such that the cardinality of Ω is $\log n \cdot 2^{O(k)}$. Thus, our construction for a (n, k)-universal set can be phrased in coding theory terminology. Let G be the generating matrix of a binary $[n', k', d]$ linear code for which

$$\frac{d}{n'} = \frac{1}{2} - \frac{1}{2^{2k}}$$

and let H be the parity check matrix of a binary $[n, n', k]$ linear code. The (n, k)-universal set consists of the rows of the matrix $G \cdot H$. This is better than the best explicit construction given in [51] for $k << n$, and matches the lower bound given there up to constant factors. Alon [3] showed an explicit construction of size $\log n \cdot 2^{O(k^2)}$ which is optimal for constant k. No such constructions were known for larger values of k. For k which is $\Theta(\log n)$, this is the first explicit construction of an (n, k)-universal set of polynomial size.

9. Communication complexity. Suppose player A has a string $x \in \{0, 1\}^n$ and player B has a string $y \in \{0, 1\}^n$, and they wish to decide whether $x = y$ while exchanging as few bits as possible. It is well known that this can be done by exchanging $O(\log n)$ bits and the probability of the protocol arriving at the correct result is at least a constant. We can show how to achieve probability of success which is any $\frac{1}{\text{poly}(n)}$, while maintaining the logarithmic communication complexity.

The algorithm will use the first two stages in constructing the small biased probability spaces. To achieve probability of error ϵ:

1. Player A chooses $O(\log n + \log \frac{1}{\epsilon})$ random bits that define vectors $r_1, r_2, \ldots, r_l$. She sends the random bits to player B and also sends $< r_1, x >, < r_2, x >, \ldots, < r_l, x >$.
2. B computes $< r_1, y >, < r_2, y >, \ldots < r_n, y >$. If for any i, $< r_i, y > \neq < r_i, x >$ he announces it. Otherwise, he concludes that $x = y$.

If $x \neq y$, then with probability at least $1 - \epsilon$, for at least one i, $< r_i, x \oplus y > \neq 0$ and hence $< r_i, x > \neq < r_i, y >$.

Once a distinguisher has been found, detecting an index of an entry on which players A and B differ is easy by exchanging $\log n$ bits.

Another application is for the problem of determining set equality. Suppose that $A, B \subset \{1, \ldots, n\}$ and we wish to decide whether $A = B$. We would like a single pass on the elements of A and B and the amount of memory should be $O(\log n)$. Here again a method that achieves constant probability error is known (see Blum and Kannan [16] for a description).

We will show that it is possible to achieve any ϵ error while using only $O(\log n + \log \frac{1}{\epsilon})$ bits of memory. Let v_A and v_B denote the incidence vectors of A and B. We can think of the problem as deciding whether $v_A \oplus v_B = 0$. Again, we can use the first two stages of the construction of small bias probability spaces.

1. Pick $O(\log n + \log \frac{1}{\epsilon})$ random bits that define $r_1, r_2, \ldots, r_l$. (They are not computed explicitly at this point.) Let $r_i(a)$ denote the ath coordinate of r_i.
2. Initialize bits $d_1, d_2, \ldots d_l$ to 0.
3. For each element $a \in A$ and for each $1 \leq i \leq l, d_i \leftarrow d_i \oplus r_i(a)$.
4. For each element $b \in B$ and for each $1 \leq i \leq l, d_i \leftarrow d_i \oplus r_i(b)$.
5. If all the d_is are 0, decide that $A = B$; otherwise $A \neq B$.

As before, if for at least one i, $< r_i, v_A \oplus v_B > \neq 0$, we will detect the inequality of the sets. The probability that this happens is at least $1 - \epsilon$. For this application, the first method of constructing small bias probability spaces should be used, since we are not interested in the full vector r_i, but in selected locations of it.

This can be looked upon as a family $\mathcal{H}$ of hash functions with the following property. Let $h \in \mathcal{H}$, where $h : \{1 \ldots n\} \rightarrow \{1 \ldots m\}$. The family $\mathcal{H}$ is accessible with $O(\log n + \log m)$ bits and for any subsets A and B of $\{1 \ldots n\}$, the probability that $\sum_{a \in A} h(a) = \sum_{b \in B} h(b)$ is smaller than $\frac{1}{m^\beta}$ for β constant, where addition is bitwise XoR.

10. Further results. Since the preliminary version of this paper appeared in [42], several new applications of small bias probability spaces have been discovered. Alon [4] used them to obtain an NC^1 algorithm for the parallel version of Beck's algorithm for the Lovasz local lemma. Feder, Kushilevitz, and Naor [24] applied them to amortize the communication complexity of equality. Blum et al. [15] used the hash function construction to authenticate memories. Kushilevitz and Mansour [31] used small bias probability spaces to derandomize their decision tree learning algorithm. Alon et al. [7] used the polylogarithmic size construction of $\log\log n$-wise $1/\log n$-bias probablity space in order to derandomize an algorithm for all pairs shortest path whose running time is almost that of matrix multiplication. Blum and Rudich [14] have used the construction of (n, k)-universal sets of §8 to derandomize a k-term DNF polynomial time learning algorithm, for k which is logarithmic in the number of variables.

Azar, Motwani, and Naor [10] defined and constructed small bias probability spaces for nonbinary random variables. Even et al. [23] constructed δ-dependent nonuniform probability spaces based on small bias probability spaces. Alon et al. [8] provided simple and different constructions for small bias probability spaces. Their constructions are based on quadratic characters and linear feedback shift registers. Schulman [48] constructed efficiently probability spaces with known dependencies.

Acknowledgment. We would like to thank Shuki Bruck for his contribution to §3.1.2. We would also like to thank Noga Alon, Yishay Mansour, and Avi Wigderson for helpful discussions. We thank Steven Ponzio and the referees for their careful reading of the manuscript, and we thank Robert Beals for pointing out [22].

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, *Deterministic simulation in* LOGSPACE, in Proceedings of the 19th ACM Annual Symposium on Theory of Computing, 1987, pp. 132–140.
[2] M. Ajtai and A. Wigderson, *Deterministic simulation of probabilistic constant depth circuits*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 11–19.
[3] N. Alon, *Explicit constructions of exponential sized families of k-independent sets*, Discrete Math, 58 (1986), pp. 191–193.
[4] ———, *A parallel algorithmic version of the local lemma*, in Random Structures and Algorithms, 2 (1991), pp. 367–378.

[5] N. Alon, L. Babai, and A. Itai, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[6] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth, *Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs*, IEEE Trans. Inform. Theory, 38 (1992), pp. 509–516.

[7] N. Alon, Z. Galil, O. Margalit, and M. Naor, *Witnesses for matrix multiplication and for shortest paths*, in Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, 1992, pp. 417–426.

[8] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, *Simple constructions for almost k-wise independent random variables*, in Random Structures and Algorithms, 3 (1992), pp. 289–304.

[9] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., New York, 1992.

[10] Y. Azar, R. Motwani, and J. Naor, *Approximating arbitrary distributions using small sample spaces*, 1990, manuscript.

[11] E. Bach, *Realistic analysis of some randomized algorithms*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 453–461.

[12] R. Ben-Nathan, M.Sc. Thesis, Hebrew University, 1990.

[13] B. Berger and J. Rompel, *Simulating* ($\log^c n$)*-wise independence in* NC, J. Assoc. Comput. Mach., 38 (1991), pp. 1026–1046.

[14] A. Blum and S. Rudich, *Fast learning k-term* DNF *formulas with queries*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, to appear.

[15] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, *Checking the correctness of memories*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 90–99.

[16] M. Blum and R. Kannan, *Designing programs that check their work*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1987, pp. 86–97.

[17] B. Chor and O. Goldreich, *On the power of two-point based sampling*, J. Complexity, 5 (1989), pp. 96–106.

[18] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem or t-resilient functions*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 396–407.

[19] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.

[20] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification and weak random sources*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 14–19.

[21] ———, *Multigraph amplification*, survey, 1989.

[22] P. Diaconis, *Group Representations in Probability and Statistics*, IMS Lecture Notes, Monograph Series, Volume 11, Hayward, CA, 1988.

[23] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Velickovic, *Approximation of general independent distributions*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 10–16.

[24] T. Feder, E. Kushilevitz, and M. Naor, *Amortized communication complexity*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 239–248.

[25] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1968.

[26] R. Freivalds, *Fast probabilistic algorithms*, Lecture Notes in Computer Science, Volume 74, Mathematical Foundations of Computer Science, Springer-Verlag, Berlin, New York, 1979, pp. 57–69.

[27] O. Gaber and Z. Galil, *Explicit construction of linear size superconcentrators*, J. Comput. Systems Sci., 22 (1981), pp. 407–420.

[28] R. Impagliazzo and D. Zuckerman, *Recycling random bits*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 248–253.

[29] S. Jimbo and A. Marouka, *Expanders obtained from affine transformations*, in Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 88–97.

[30] J. Justesen, *A class of asymptotically good algebraic codes*, IEEE Trans. Inform. Theory, 18 (1972), pp. 652–656.

[31] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 455–464; SIAM J. Comput., to appear.

[32] R. Karp and N. Pippenger, *A time randomness tradeoff*, AMS Conference on Probabilistic Computation and Complexity, Durham, NC, 1983.

[33] H. Karloff and P. Raghavan, *Randomized algorithms and pseudorandom numbers*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 310–321.

[34] H. Karloff and D. Shmoys, *Efficient parallel algorithms for edge coloring problems*, J. Algorithms, 8 (1987), pp. 39–52.

[35] D. J. Kleitman and J. Spencer, *Families of k-independent sets*, Discrete Math., 6 (1973), pp. 255–262.

[36] R. M. KARP AND A. WIGDERSON, *A fast parallel algorithm for the maximal independent set problem*, J. Assoc. Comput. Mach., 32 (1985), pp. 762–773.

[37] A. LUBOTZKY, R. PHILLIPS, AND P. SARNAK, *Explicit expanders and the Ramanujan conjecture*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 240–246.

[38] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[39] R. MOTWANI, J. NAOR, AND M. NAOR, *The probabilistic method yields deterministic parallel algorithms*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 8–13.

[40] F. J. MACWILLIAMS AND N. J. A. SLOANE, *The Theory of Error Correcting Codes*, North–Holland, Amsterdam, 1977.

[41] M. NAOR, *Constructing Ramsey graphs from small probability spaces*, IBM Res. Report RJ 8810, 1992.

[42] J. NAOR AND M. NAOR, *Small-bias probability spaces: Efficient constructions and applications*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 213–223.

[43] N. NISAN AND A. WIGDERSON, *Hardness vs. randomness*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 2–11.

[44] R. PERALTA, *On the randomness complexity of algorithms*, CS Res. Report TR 90-1, University of Wisconsin, Milwaukee, WI.

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

[46] R. RIVEST, A. SHAMIR, AND L. ADELMAN, *A method for obtaining digital signatures and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

[47] M. SANTHA, *On using deterministic functions to reduce randomness in probabilistic algorithms*, Inform. and Comput., 74 (1987), pp. 241–249.

[48] L. SCHULMAN, *Sample spaces uniform on neighborhoods*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 17–25.

[49] M. SIPSER, *Expanders, randomness, or time versus space*, J. Comput. Systems Sci., 36 (1988), pp. 379–383.

[50] J. SPENCER, *Ten Lectures on the Probabilistic Method*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[51] G. SEROUSSI AND N. BSHOUTI, *Vector sets for exhaustive testing of logic circuits*, IEEE Trans. Inform. Theory, 34 (1988), pp. 513–522.

[52] U. VAZIRANI, *Randomness, adversaries and computation*, Ph.D. Thesis, University of California, Berkeley, CA, 1986.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 857–874, August 1993

DECIDING PROPERTIES OF NONREGULAR PROGRAMS*

DAVID HAREL†‡ AND DANNY RAZ‡

Abstract. Extensions of propositional dynamic logic (PDL) with nonregular programs are considered. Three classes of nonregular languages are defined, and for each of them it is shown that for any language L in the class, PDL, with L added to the set of regular programs as a new program, is decidable. The first class consists of the languages accepted by pushdown automata that act only on the basis of their input symbol, except when determining whether they reject or continue. The second class (which contains even noncontext-free languages) consists of the languages accepted by deterministic stack machines, but which have a unique new symbol prefixing each word. The third class represents a certain delicate combination of these, and, in particular, it serves to prove the 1983 conjecture that PDL with the addition of the language $\{a^ib^ic^i \mid i \geq 0\}$ is decidable.

Key words. propositional dynamic logic (PDL), decidability, stack automata, context-free languages

AMS subject classifications. 03B70, 03D05, 68Q60, 68Q68

1. Introduction.

Propositional dynamic logic (PDL) was introduced by Fischer and Ladner [FL], based upon the first-order version of Pratt [P1]. It is a direct extension of the propositional calculus, in which programs can appear in the formulas. Thus, for example, $P \rightarrow \langle\alpha\rangle Q$ asserts that whenever P holds it is possible to carry out some computation of α, leading to a state in which Q holds, and $\langle\alpha\rangle P \equiv \langle\beta\rangle P$ asserts a certain kind of equivalence of programs α and β. Formulas in PDL can involve many programs and are able to express a wide variety of properties pertaining to their input/output behavior. See [H], [KT] for detailed surveys.

In most versions of PDL, the programs are taken to be regular sets of sequences of basic programs and tests. Whether these are represented by regular expressions or by automata, the validity/satisfiability problem for formulas in the logic has been shown to be logspace complete for exponential time [P2], [FL], [HS]. (It is thus interesting that, as far as we know, the validity problem for this much richer logic might be no worse than that of its fragment, the propositional calculus.)

In the early 1980s the problem of extending PDL with *non*regular programs was raised. In terms of programming languages, moving from regular programs to, say, context-free ones is tantamount to moving from iterative programs to (parameterless) recursive procedures. It would be nice if we could decide the truth of propositional-level properties of recursive programs, too. Ladner observed in 1977 that PDL with context-free programs must be undecidable, since it is not too difficult to see that the formula $\langle\alpha\rangle P \equiv \langle\beta\rangle P$ is valid if and only if the languages represented by the programs α and β are equal. (Here, P is a basic propositional letter.) However, it remained to investigate (a) the level of undecidability of context-free PDL, and (b) the point at which the problem starts becoming undecidable.

The first results were strikingly negative. Denote by PDL_L the logic obtained by allowing the language L as a single new program, in addition to the regular ones. In [HPS] it was shown that the validity problem for both PDL_{L_1} and PDL_{L_2,L_3} (in the second case there are *two* new programs) is highly undecidable, that is, Π_1^1-complete. Here,

$$L_1 = a^\Delta b a^\Delta = \{a^iba^i \mid i \geq 0\},$$

*Received by the editors September 25, 1991; accepted for publication (in revised form) April 9, 1992. A preliminary version of this paper appeared in Proc. 31st IEEE Symposium on Foundations of Computer Science, IEEE Press, New York, 1990. This work was supported in part by grants from the Gutwirth Foundation and the Israel Ministry of Science and Technology.

†This author holds the William Sussman chair in mathematics.

‡Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel (harel@wisdom.weizmann.ac.il and danny@wisdom.weizmann.ac.il).

$$L_2 = a^\Delta b^\Delta = \{a^i b^i \mid i \geq 0\},$$

$$L_3 = b^\Delta a^\Delta = \{b^i a^i \mid i \geq 0\}.$$

This settled issue (a) above since context-free PDL in its entirety can be easily shown to be *in* Π_1^1. It was also shown in [HPS] that there exists a primitive recursive nonregular one-letter language (which must be noncontext free, by Parikh's theorem [P]) that exhibits the same disastrous behavior. Later, in [HP], it was shown that the specific logic PDL_{L_4} is also Π_1^1-complete, where

$$L_4 = \{a^{(2^i)} \mid i \geq 0\}.$$

In contrast to these negative results, a rather surprising positive result was proved in [KP], to the effect that adding L_2 or L_3 alone does *not* result in undecidability. Thus, PDL_{L_2}, for example, is decidable, which is rather curious, given the similarity of L_1 and L_2. The proof in [KP] shows that PDL_{L_2}, which does not enjoy the usual finite model property of [FL], does, in fact, enjoy a finite *pushdown model* property. More specifically, each satisfiable formula has a model of bounded size, along the edges of which appear not only program letters but also various *push* and *pop* instructions. The details of the construction are quite complicated, and strongly depend on the idiosyncrasies of the language L_2.

The feeling we get when studying these results is that the borderline between the classes of languages for which extended PDLs are decidable and those for which they are not has to do with the behavior of accepting pushdown machines. For example, one important difference between L_1 and L_2 is that in accepting the former language a pushdown automaton reading an a may carry out a push or a pop depending upon its location in the input word, whereas in the latter language the a's can be safely pushed and the b's popped, regardless of where we are in the word. When dealing with acceptance in PDL models, a machine seeing an a out of context will not "know" what to do in the first case, but will in the second. If anything like this distinction is the "right" one, it would be particularly interesting to know the answer to a question raised in [H, p. 533]: Is PDL_{L_5} decidable, where

$$L_5 = a^\Delta b^\Delta c^\Delta = \{a^i b^i c^i \mid i \geq 0\}\ ?$$

Harel conjectured in 1983 that the answer is yes. Note that L_5 is not context-free, and thus cannot be accepted with a single pushdown. However, if a machine with two pushdowns is employed, or a stack machine,[1] it enjoys a similar "obliviousness" property; its operations on the stack are determined by the input symbol alone.

In this paper we provide the first results in this area that do not deal with isolated languages, but apply to broad classes of languages in a general way. The approach we use is to deal directly with the automata that accept the languages under consideration. Our first result takes a step towards confirming the intuition outlined above. Specifically, call a pushdown automaton *simple-minded* (an SM-PDA for short) if its behavior is uniquely determined by the input symbol alone, and the internal state and stack symbol may only help determine whether the machine aborts or carries out this (unique) next step. It is noteworthy that SM-PDAs accept a large fragment of the context-free languages, including L_2 and L_3, as well as all manner of parenthesis languages (semi-Dyck sets), and many of their intersections with regular languages. Here we have the following theorem.

[1] A stack machine is a one-way pushdown automaton whose head can travel up and down the stack but in which changes can be made only at its top [Ka].

THEOREM 1. *If L is accepted by an* SM-PDA, *then* PDL_L *is decidable.*

Our second result considers a different class of languages, based on a richer kind of machine, namely, deterministic stack automata (DSA). These accept many kinds of noncontext-free languages, such as L_5 and its generalizations $a_1^{\Delta}a_2^{\Delta}\dots a_n^{\Delta}$ for any n, as well as many variants thereof. Here we have our next theorem.

THEOREM 2. *If L is a language over Σ accepted by a* DSA, *then* $\mathrm{PDL}_{L'}$ *is decidable, where $L' = \{¢ \cdot w \mid w \in L\}$ and $¢$ is a new symbol, $¢ \notin \Sigma$.*

Finally, combining the constructions used in the proofs of both theorems we are able to resolve the L_5 problem with the following theorem.

THEOREM 3. PDL_{L_5} *is decidable.*

The proofs of Theorems 1 and 2 follow the widely applicable ideas of Vardi and Wolper [VW], in which decidability of a satisfiability problem in a logic of programs is established by defining special classes of automata on infinite trees and showing that their emptiness problem is decidable. This construction is typically preceded by proving that every satisfiable formula has a satisfying tree model of an appropriate form.

In our case, we had to go beyond the various kinds of finite-state tree automata considered in [VW] and its follow ups, and consider pushdown and stack automata on infinite trees. The emptiness problem for such automata was not known to be decidable, and the proof that it is turned out to be quite involved and possibly of independent interest. Consequently, we have devoted a separate paper to it [HR]. In §3 of the present paper we describe this result briefly.

In §2 we show that for *any* language L, a satisfiable formula in PDL_L has a special kind of Hintikka tree model, labeled with elements of an appropriate subformula-like closure. Such a model has bounded branching, and is constructed so that the paths satisfying any two "diamond promises" are edge distinct.

We then turn to the main constructions, leading, respectively, to Theorems 1 and 2. Given a formula f in PDL_L, where L is accepted by some SM-PDA, we construct in §4 a tree pushdown automaton A_f that accepts precisely the aforementioned Hintikka models of f. Thus, testing A_f for emptiness amounts to determining the satisfiability of f. The construction and its justification appear to be considerably more complicated than similar kinds of results in the literature. We should add that a naive implementation of the entire decision procedure yields triple-exponential time.

In §5 we carry out a similar construction of a tree *stack* automaton for the case where L is accepted by a DSA M, but here we must require the presence of the new symbol $¢$ preceding the words of the language, due to the richer possibilities in the behavior of M.

In §6 we show how the combination of both constructions (the one of §4 and the one of §5) can be used to prove decidability of an even more general family of languages. In this family, we are able to replace the requirement for the special symbol $¢$ of Theorem 2 with a new category of the stack machines. In addition to some technical requirement on the accepting machines, we require that all words accepted can be partitioned into two parts, coming one after the other, and defined over two disjoint alphabets. In particular, since L_5 is accepted by such a machine, we get a proof for Theorem 3. Section 7 contains a discussion of remaining open problems.

2. The existence of "nice" models.

We will use a variant of PDL called APDL in [HS]. In APDL, regular programs are described by finite automata rather than regular expressions. Notice that in terms of decidability there is no difference between PDL and APDL. We use APDL because dealing with automata is a little more convenient here. Let *Prop* be a set of atomic propositions and let *Prog* be a set of atomic programs. Let L denote a fixed language over *Prog*. APDL_L is defined as follows:

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas, then so are $\neg f_1$ and $f_1 \wedge f_2$.
- If f is a formula, then $f?$ is a test.
- If α is a program and f is a formula, then $\langle\alpha\rangle f$ is a formula.
- If α is a finite automaton over alphabet Σ, where Σ is a finite set of atomic programs from $Prog$, tests (each of which is considered a single symbol of Σ), and the special letter L, then α is a program.

Note that in the syntax L is treated as an atomic program, while in the semantics it will be interpreted as the set of sequences of atomic programs it denotes.

APDL$_L$ formulas are interpreted over structures $M = (W, R, \pi)$, where W is a set of states, $R : Prog \to 2^{W \times W}$ is a transition relation , and $\pi : W \to 2^{Prop}$ assigns truth values to the propositions in $Prop$ for each state in W. We now, inductively, extend R to all programs and define satisfaction of a formula F in a state u of a structure M, denoted $M, u \models f$:

- $R(f?) = \{(u, u) | M, u \models f\}$.
- $R(L) = \{(u, v)|$ there exists a word $w = w_1 \dots w_l$ in the language L, and states $u_0, u_1, \dots, u_l$ of W such that $u = u_0$, $v = u_l$, and for all $1 \leq i \leq l$, $(u_{i-1}, u_i) \in R(w_i)$ $\}$.
- $R(\alpha) = \{(u, v)|$ there exists a word $w = w_1 \dots w_l$ accepted by α, and states $u_0, u_1, \dots, u_l$ of W such that $u = u_0$, $v = u_l$, and for all $1 \leq i \leq l$, $(u_{i-1}, u_i) \in R(w_i)$ $\}$.
- For a proposition $p \in prop$, $M, u \models p$ if and only if $p \in \pi(u)$.
- $M, u \models f_1 \wedge f_2$ if and only if $M, u \models f_1$ and $M, u \models f_2$.
- $M, u \models \neg f_1$ if and only if not $M, u \models f_1$.
- $M, u \models \langle\alpha\rangle f$ if and only if there exists a state v such that $(u, v) \in R(\alpha)$ and $M, v \models f$.

A formula f is *satisfiable* if there is a structure M and a state u such that $M, u \models f$. The *satisfiability problem* is to determine, given a formula f, whether f is satisfiable.

Before describing the decision procedure for the satisfiability problem of APDL$_L$, where L is in some special families of languages, we define the closure of APDL$_L$ formulas, similar to that defined by Fisher and Ladner in [FL]. From now on, we will identify a formula g with $\neg\neg g$. The *closure* of a formula f, denoted $cl(f)$, is defined as the smallest set satisfying:

- $f \in cl(f)$.
- If $g_1 \wedge g_2 \in cl(f)$ then $g_1, g_2 \in cl(f)$.
- If $\neg g \in cl(f)$ then $g \in cl(f)$.
- If $g \in cl(f)$ then $\neg g \in cl(f)$.
- If $\langle\alpha\rangle g \in cl(f)$ then $g \in cl(f)$.
- If $\langle\alpha\rangle g \in cl(f)$, where α is the finite state automaton $\langle\Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha\rangle$, then $h \in cl(f)$ for all $h? \in \Sigma_\alpha$.
- If $\langle\alpha\rangle g \in cl(f)$, where α is the finite state automaton $\langle\Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha\rangle$, then $\langle\alpha_s\rangle g \in cl(f)$ for all $s \in S_\alpha$, where α_s is the automaton α in which the state s replaces $s_{0\alpha}$ as the initial state.

The size of $cl(f)$ can be shown to be linear in the length of f (cf. [FL], [HS]). We first prove that APDL$_L$ has the *tree model property*. Let f be a formula of APDL$_L$, containing n distinct atomic programs including those used in L. A *tree structure* for f in APDL$_L$ is a structure $M = (W, R, \pi)$ such that:

1. $W \subseteq [k]^*$, where $k = nl$ for some integer l, and $W \neq \emptyset$.
2. If $xi \in W$ then $x \in W$.
3. If $(x, y) \in R(a)$ for an atomic program a, then x is the predecessor of y in W, and $(x, y) \notin R(b)$ whenever $b \neq a$.

A tree structure $M = (W, R, \pi)$ is a *tree model* for f if $M, \lambda \models f$. We now show that for any L, if an APDL$_L$ formula f is satisfiable, then it has a tree model. To do this we unravel any model of f into a tree, and then use a subset-like argument to create a structure in which every state has a finite number of successors. More formally we have the following proposition.

PROPOSITION 2.1. *A formula f in* APDL$_L$ *is satisfiable if and only if it has a tree model.*

Proof. The *only if* part is immediate from the fact that a tree structure is a structure. Suppose that $M, u \models f$, for some structure $M = (W, R, \pi)$ and some state $u \in W$, and let $C_1, \dots, C_{2^{|cl(f)|}}$ be an enumeration of the subsets of $cl(f)$ and let $k = nr$ for some integer r. To show that f has a tree model, we first define a partial mapping $\phi : [k]^* \to 2^W$ by induction on the length of words in $[k]^*$. First, let $\phi(\lambda) = \{u\}$. Suppose now that ϕ is known for every $x \in [k]^m$, and let $xl \in [k]^{m+1}$. If $\phi(x)$ is the empty set then so is $\phi(xl)$. If $\phi(x) = S \subseteq W$ is not empty, then for all $1 \le i \le n$ and $1 \le j \le 2^{|cl(f)|}$, we have:

$$\phi(x(i+nj)) = \{t \in W \mid \exists s \in S_j (s,t) \in R(a_i) \ \ and \ C_j = \{g \in cl(f) \mid M, t \models g\}\}$$

We now define a structure $M' = (W', R', \pi')$ as follows. $W' = \{x \mid \phi(x) \text{ is not empty}\}$, $R'(a_j) = \{(x, xi) \mid i = j + nj \text{ for all } 1 \le j \le 2^{|cl(f)|}, \text{ and } xi \in W'\}$, and $\pi'(x) = \pi(t)$ for some $t \in \phi(x)$. (Note that π' is well defined by the definitions of ϕ and π.) It is not difficult to show that M' is a tree structure, and that if $x \in W'$ and $g \in cl(f)$, then $M', x \models g$ if and only if $M, \phi(x) \models g$. In particular, $M', \lambda \models f$. □

We would now like to associate a natural infinite k-ary tree over $2^{cl(f) \cup \{\perp\}}$ with the tree model $M' = (W', R', \pi')$ constructed above: Every node in W' will be labeled by the formulas in $cl(f)$ that are satisfied therein, and all the nodes not in W' are labeled by the special symbol $\perp$. These trees satisfy some special properties, as we will now see.

DEFINITION 2.2. A *Hintikka tree* for an APDL$_L$ formula f with atomic programs $a_1, \dots, a_n$ is a k-ary tree $T : [k]^* \to 2^{cl(f) \cup \{\perp\}}$, such that $f \in T(\lambda)$, $k \ge n$, and for all elements x of $[k]^*$:

1. Either $T(x) = \{\perp\}$ or $\perp \notin T(x)$, and $g \in T(x)$ if and only if $\neg g \notin T(x)$;
2. $g_1 \wedge g_2 \in T(x)$ if and only if $g_1 \in T(x)$ and $g_2 \in T(x)$;
3. If $\langle\alpha\rangle g \in T(x)$, where α is the finite state automaton $\langle \Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha \rangle$, then there exists a word $w = w_1 \dots w_m$ over Σ_α, states $s = s_1, \dots, s_m$ of S_α, and $u = u_0, \dots, u_m$ of $[k]^*$ such that $u_0 = x$, $g \in T(u_m)$, $s_m \in F$, and, for all $1 \le i \le m$:
   (a) $s_i \in \rho_\alpha(s_{i-1}, w_i)$;
   (b) If w_i is $h?$, then $h \in T(u_{i-1})$ and $u_i = u_{i-1}$;
   (c) If w_i is $a_j \in Prog$, then $u_i = u_{i-1}r$, where $r = j + nl \le k$ for some l;
   (d) If w_i is L, then there exists $l \ge 0$, a word $a_{j_1} \dots a_{j_l} \in L$ and $1 \le r_1 \dots r_l \le k$, such that $u_i = u_{i-1}r_1 \dots r_l$, and for all $1 \le m \le l$ we have $r_m = j_m + nl_m$ for some l_m.
4. If $\neg\langle\alpha\rangle g \in T(x)$, where α is the finite state automaton $\langle \Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha \rangle$, then $s_0 \in F_\alpha$ implies $\neg g \in T(x)$, and for all $s \in S_\alpha$ and $w \in \Sigma_\alpha$, such that $s \in \rho_\alpha(s_0, w)$, we have:
   (a) If w is $h?$, then either $\neg h \in T(x)$ or $\neg\langle\alpha_s\rangle g \in T(x)$;
   (b) If w is $a_j \in Prog$, then for each $r = j + nl \le k$ we have $\neg\langle\alpha_s\rangle g \in T(xr)$ or $T(xr) = \{\perp\}$;
   (c) If w is L, then for all $a_{j_1} \dots a_{j_l} \in L$ and for every $r_m = j_m + nl_m \le k$, we have $\neg\langle\alpha_s\rangle g \in T(xr_1 \dots r_l)$ or $T(xr_1 \dots r_l) = \{\perp\}$.

PROPOSITION 2.3. *A formula f in* APDL$_L$ *has a Hintikka tree if and only if it has a tree model.*

Proof. Let f be an APDL$_L$ formula with atomic programs $a_1, \dots, a_n$.

(*if*) Let $M = (W, R, \pi)$ be a tree model of f. We define the Hintikka tree T for f as follows: For an element $x \in [k]^* - W$ take $T(x) = \{\perp\}$, and for an element $x \in W$, take

$T(x) = \{g \in cl(f) \mid M, x \models g\}$. We now must show that T is indeed a Hintikka tree for f. By the definition of a tree model, $M, \lambda \models f$, which implies condition 1. That conditions 2, 3, 4, and 5 hold follows immediately from the definition of satisfaction in APDL$_L$.

(*only if*) Let T be a Hintikka tree for f. We construct a tree model for f as follows. The structure is $M = (W, R, \pi)$, where $W = \{x \in [k]^* \mid T(x) \neq \{\perp\}\}$, $R(a_i) = \{(x, xj) \mid j = i + nl \in [k] \text{ and } xj \in W\}$, and for all $x \in W$, $\pi(x) = \{p \in prop \mid p \in T(x)\}$. It now remains to show that $M, \lambda \models f$. To that end, we show by induction on the structure of the formula that for all $g \in cl(f)$ and $x \in [k]^*$, $g \in T(x)$ if and only if $M, x \models g$. For the base case of $g \in Prop$, this is immediate by the construction. The inductive step for formulas of the form $g_1 \wedge g_2$, $\neg g_1$, $\langle\alpha\rangle g_1$, and $\neg\langle\alpha\rangle g_1$ follows directly from the Hintikka conditions 2, 3, 4, and 5, respectively. □

One more property of models is needed to be able to carry out the construction of the appropriate automata in the next section.

DEFINITION 2.4. A *unique diamond path Hintikka tree* for a formula f, or a UDH for short, is a Hintikka tree for f that satisfies the following additional condition: There exists a mapping $\Phi : [k]^* \to cl(f) \cup \{\perp\}$, such that for all $u \in [k]^*$:

1. $\Phi(u)$ is either a single diamond formula or the special symbol $\perp$.
2. If $\langle\alpha\rangle g \in T(u)$ then in clause 3 of the definition of a Hintikka tree the nodes $u_1, \ldots, u_m$, guaranteed to exist, must have the property that for all $1 \leq j \leq m$, $\Phi(u_j) = \langle\alpha\rangle g$. (Notice that $\Phi(u_j)$ does not have to be in $T(u_j)$.)

The intuition behind this definition is that at each node u, $\Phi(u)$ singles out a (unique) diamond formula, in such a way that each path is associated with at most one such formula.

We now prove that for any given formula f, we can construct a UDH from any Hintikka tree for f. This is done simply by increasing the number of descendants of each node, so that there is a separate branch for each formula when needed. More formally we have the following proposition.

PROPOSITION 2.5. *A formula f in* APDL$_L$ *has a Hintikka tree if and only if it has a* UDH.

Proof. Given a Hintikka tree $T : [k]^* \to 2^{cl(f) \cup \{\perp\}}$ for f, we must show that f has a UDH. Let $p = |cl(f)| + 1$, $k' = p \cdot k$, and $f_1, f_2, \ldots, f_{p-1}$ be a fixed ordering of $cl(f)$. Define $\psi : [k']^* \to [k]^*$ by induction on the length of words in $[k']^*$. First, let $\psi(\lambda) = \lambda$. Suppose that ψ is known for every $x \in [k]^{l'}$; if $xj \in [k]^{l'+1}$, where $j = i + l \cdot p$ for some $1 \leq i \leq n$ and $1 \leq l < p$, then let $\psi(xj) = xi$. We now define $T' : [k']^* \to 2^{cl(f) \cup \{\perp\}}$ by $T'(u) = t(\psi(u))$.

To conclude the construction of the UDH, define $\Phi : [k']^* \to cl(f) \cup \{\perp\}$ by induction on the length of words in $[k']^*$. Let $\Phi(\lambda) = \perp$. Suppose that Φ is known for every $x \in [k]^{l'}$; if $xj \in [k]^{l'+1}$, then let $\Phi(xj) = f_i$ whenever $i \cdot k = j$ and f_i is a diamond formula, and $\perp$ otherwise.

T' can be seen to be a Hintikka tree. Furthermore, straightforward induction on the length of $u = u_0, u_1, \ldots, u_m$ shows that Φ satisfies the conditions for a UDH. □

As in [VW], we now seek a procedure which, for a formula f, will construct an automaton accepting exactly the special tree models whose existence was established above, namely the UDHs for f. The automata used must be sufficiently powerful to be able to run through the trees, making sure that things are as they should be, but sufficiently weak to have a decidable emptiness problem.

3. Deciding emptiness for stack automata on infinite trees. In this section we provide a full definition of stack automata on infinite trees. This definition is needed later on when we prove that such an automaton accepts precisely the UDHs of some formula.

Let $[k] = \{1, 2, \ldots, k\}$. *A k-ary tree* over a set S is a labeling of the set $[k]^*$ by members of S. The empty word λ denotes the root of the tree.

A *stack k-ary ω-tree automaton* (or an STA for short) is a structure

$$M = \langle Q, \Sigma, \Gamma, q_0, z_0, \delta, F \rangle,$$

where Q is a (finite) set of states, Σ is the (finite) input alphabet, Γ is the (finite) stack alphabet, $q_0 \in Q$ is the initial state, $z_0 \in \Gamma$ is the initial stack symbol, and $F \subseteq Q$ is the set of designated accepting states.

The transition function δ is defined as

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\Lambda\}) \to (2^{(Q\times B)^k} \cup 2^{Q\times B}),$$

where $B = \{pop, md, mu, sp\} \cup \{push(w) \mid w \in \Gamma^+\}$. Here, md, mu, and sp stand for "move down," "move up," and "stay put," respectively. The transition function reflects the fact that M works on trees with outdegree k that are labeled by Σ. The number of rules in δ is denoted by $|\delta|$. We will use Λ as the empty symbol of the stack, describing the stack positions beyond the top of the stack. This is useful when we want to determine if the head is at the top of the stack.

A good (informal) way of viewing STAs is to consider a stack machine that travels on an infinite tree of outdegree k. At each node n of the tree, the machine splits into k copies, each copy continuing to travel the tree from the corresponding successor of n. The way the machine splits, and the configurations of the new copies, are determined by the transition function δ and the label of n.

The set of *stack configurations* is $S = z_0\Gamma^*\uparrow\Gamma^* \cup z_0\Gamma^*\Lambda\uparrow$, and the *initial stack configuration* s_0 is $z_0\uparrow$. For a stack configuration $s = z_0\gamma_1\uparrow\gamma_2$, the *depth* of s is $d(s) = |\gamma_1| + |\gamma_2|$ and the *head position* is $h(s) = |\gamma_1|$. A *node configuration* N is a pair $(q, s) \in Q \times S$, and the *initial node configuration* n_0 is (q_0, s_0). The depth of a node configuration is the depth of its stack.

Let $\mathcal{H} : S \to \Gamma \cup \{\Lambda\}$ be given by $\mathcal{H}(z_0\gamma_1 z \uparrow \gamma_2) = z$, where $z \in \Gamma \cup \{\Lambda\}$ and $\mathcal{H}(z_0 \uparrow \gamma) = z_0$. This describes the letter read by the stack head.

In order to capture the effect of δ on stack configurations, we now define the partial function $\mathcal{B} : (S \times B) \to S$, providing the new contents of the stack:

- $\mathcal{B}(z_0\gamma z\uparrow, pop) = z_0\gamma\uparrow$.
- $\mathcal{B}(z_0\gamma\uparrow, push(w)) = z_0\gamma w\uparrow$.
- $\mathcal{B}(z_0\gamma_1 z\uparrow\gamma_2, md) = z_0\gamma_1\uparrow z\gamma_2$.
- $\mathcal{B}(z_0\gamma_1\uparrow z\gamma_2, mu) = z_0\gamma_1 z\uparrow\gamma_2$.
- $\mathcal{B}(z_0\gamma\uparrow, mu) = z_0\gamma\Lambda\uparrow$.
- $\mathcal{B}(z_0\gamma\Lambda\uparrow, md) = z_0\gamma\uparrow$.
- $\mathcal{B}(s, sp) = s$.

An *ϵ-path* from (q_1, s_1) to (q_l, s_l) is a sequence $((q_1, s_1), \ldots, (q_l, s_l))$, such that for all $1 < i \le l$, we have $(q_i, b_i) \in \delta(q_{i-1}, \epsilon, \mathcal{H}(s_{i-1}))$ and $\mathcal{B}(s_{i-1}, b_i) = s_i$. Let N_ϵ be the set of ϵ-paths. An ϵ-path *signals* F if there is $1 \le j \le l$ such that $q_j \in F$ and $d(s_j) = h(s_j)$.

A computation of M on the infinite tree t over Σ is an extension of t, defined as $C : [k]^* \to (\Sigma \times N_\epsilon)$, such that, in addition to a symbol from Σ, each node is labeled by an ϵ-path. The tree C must satisfy the following condition:

- If N is some ϵ-path to (q, s), then for all u in $[k]^*$ such that $C(u) = (a, N)$, there exists $((q_1, b_1), \ldots, (q_k, b_k)) \in \delta(q, a, \mathcal{H}(s))$, such that for all $1 \le j \le k$, $C(uj) = (a_j, N_j)$, where $a_j \in \Sigma$ and N_j is some ϵ-path from $(q_j, \mathcal{B}(b_j, s))$.

Informally, a computation is really an infinite tree labeled both by letters of Σ and by the configurations of the machine. The label of a node in the computation is its label from Σ together with the configuration of the stack machine that visited it.

A computation C is said to be *Büchi accepting*, or just *accepting* for short, if $C(\lambda) = (a, N)$, where $a \in \Sigma$ and N is an ϵ-path from (q_0, s_0), and every path in C contains infinitely many ϵ-paths that signal F. A tree t is *accepted* by M if there exists an accepting computation of M on t.

The *emptiness problem* is, given an automaton M, to determine if M accepts some tree.

An STA that uses only the symbol sp from B is simply a Büchi k-ary ω-tree automaton, as defined in [VW]. An STA that uses only the symbols sp, $push(w)$, and pop from B is a *pushdown k-ary ω-tree automaton* (PTA for short). This definition is similar to that appearing in [S]. Clearly, if $k = 1$, the infinite trees become infinite sequences. The main result needed in the sequel is the following theorem.

THEOREM [HR]. *The emptiness problem for STAs is decidable.*

The proof in [HR] establishes decidability in four-fold exponential time for STAs and in triple-exponential time for PTAs.

4. Simpleminded context-free languages. Let $M = \langle Q, \Sigma, \Gamma, q_o, z_0, \delta\rangle$ be a finite state pushdown automaton that accepts by empty stack. We say that M is *simple minded*, or an SM-PDA for short, if, whenever $\delta(q, \sigma, \gamma) = (p, b)$, then for each q' and γ' either $\delta(q', \sigma, \gamma') = (p, b)$ or $\delta(q', \sigma, \gamma')$ is undefined. In other words, the automaton's action is determined uniquely by the input symbol; the state and stack symbol are only used to help determine whether the machine halts (rejecting the input) or continues. Note that such an automaton is necessarily deterministic. A language L is said to be *simpleminded context-free* (or an sm-cfl, for short) if there exists an SM-PDA that accepts it.

To make this definition clear, accepting by empty stack here means that whenever z_0 is at the top of the stack the machine accepts its input. This way of acceptness is much more convenient when dealing with deterministic machines. In [WW, p. 219] a subclass of CFLs, called *input driven*, is mentioned. This class is accepted by DPDAs in which moves of the pushdown head depend only on the input symbol. Input driven languages can be recognized in logarithmic space [BV]. In our case (sm-cfl), we require that the new state also depends only on the input symbol, hence, sm-cfls can be recognized in logarithmic space, too.

Example 4.1. Let $M = \langle\{q_0, q\}, \Sigma, \Gamma, q_0, z_0, \delta\rangle$ be a PDA, where $\Sigma = \{a, b\}$, $\Gamma = \{z, z_0\}$, and the transition function δ is given by:

$$\delta(q_0, a, z_0) = (q_0, push(z))$$

$$\delta(q_0, a, z) = (q_0, push(z))$$

$$\delta(q_0, b, z) = (q, pop)$$

$$\delta(q, b, z) = (q, pop).$$

Here δ is undefined for all other possibilities. Since M accepts by empty stack, the language accepted is precisely $L_2 = \{a^i b^i \mid i \geq 0\}$. The automaton M is an SM-PDA since it always performs $push(z)$ when the input is a, and pop when the input is b.

Example 4.2. Let $M = \langle\{q\}, \Sigma\cup\Sigma', \Gamma, q, z_0, \delta\rangle$ be a PDA, where $\Sigma = \{[,]\}$, $\Gamma = \{[, z_0\}$, and the transition function δ is given by:

$$\delta(q, [, z) = (q, push([))$$

$$\delta(q, \sigma, [) = (q, sp), \quad \text{for any } \sigma \notin \{[,]\}$$

$$\delta(q,], [) = (q, pop).$$

Here $\sigma \in \Sigma'$ and δ is undefined for all other possibilities. Since the automaton only accepts by empty stack, the language accepted by M is precisely the set of balanced parentheses expressions over Σ'. The automaton M is an SM-PDA since it always performs *push*([ ) when the input is "[", *pop* when the input is "]", and *sp* when the input is a letter from Σ'.

It is easy to see that all semi-Dyck sets [Ha], are sm-cfl, as are many others. Since throughout this chapter we are dealing with pushdown automata, we will assume that they are given in a normal form, in which the automaton never push more than one symbol per move. We use the following notation:

$$\begin{array}{rcl} \delta(q,a,z) = (p,\epsilon) & \text{instead of} & \delta(q,a,z) = (p, pop) \\ \delta(q,a,z) = (p,z) & \text{instead of} & \delta(q,a,z) = (p, sp) \\ \delta(q,a,z) = (p,zz') & \text{instead of} & \delta(q,a,z) = (p, push(z')). \end{array}$$

Let us define $\Omega : \Sigma \times \Gamma \to \Gamma^*$ by: $\Omega(a,z) = w$ if there exists $p, q \in Q$ such that $\delta(p,a,z) = (q,w)$. Notice that for SM-PDAs Ω is a partial function.

Given an sm-cfl L, we now describe the construction, for each f in APDL$_L$, of an STA A_f. This STA will be shown to accept precisely the UDH trees of the formula f. A_f is an appropriate parallel composition of three machines. The first, called A_l, is a tree automaton (with no pushdown) that tests the input tree for standard local consistency properties. The second component of A_f, called $A_\diamond$, is a tree PDA that deals with the diamond formulas of $cl(f)$, and the third component, called $A_\Box$, is a tree PDA that deals with the negations of diamond formulas, i.e., with box formulas. Thus, for the rest of this section, A_f is actually a PTA, and not a full-fledged STA.

Let $M_L = \langle Q, \Sigma, \Gamma, q_{0_L}, z_0, \rho\rangle$ be an SM-PDA that accepts the language L, and let f be a formula in APDL$_L$.

The local automaton for f is

$$A_l = \langle 2^{cl^+(f)}, 2^{cl^+(f)}, N_f, \delta, 2^{cl^+(f)}\rangle$$

where:

- $cl^+(f) = cl(f) \cup \{\bot\}$;
- The starting set N_f consists of all sets s such that $f \in s$;
- $(s_1, \ldots, s_k) \in \delta(s,a)$ if and only if $s = a$ and:
  - either $s = \{\bot\}$, or $\bot \notin s$,
  - $g \in s$ if and only if $\neg g \notin s$,
  - $g_1 \wedge g_2 \in s$ if and only if $g_1 \in s$ and $g_2 \in s$,
  - if $\neg\langle\alpha\rangle g \in s$, where α is the finite state automaton $\langle\Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0_\alpha}, F_\alpha\rangle$, then $s_{0_\alpha} \in F_\alpha$ implies $\neg g \in s$, and for all $q \in S_\alpha$ and $w \in \Sigma_\alpha$ such that $q \in \rho_\alpha(s_{0_\alpha}, w)$:
    1. If w is $h?$, then either $\neg h \in s$ or $\neg\langle\alpha_q\rangle g \in s$, and
    2. If w is $a_j \in Prog$, then for each $i = j + nl \leq k$ we have $\neg\langle\alpha_q\rangle g \in s_i$ or $s_i = \{\bot\}$.

PROPOSITION 4.3. *The automaton A_l accepts precisely the trees that satisfy Hintikka conditions* 1, 2, 4(a), *and* 4(b).

Proof. A computation of an automaton M on an infinite tree $T : [k]^* \to \Sigma$ is an infinite tree $C : [k]^* \to Q'$,where Q' is the set of internal states of M. Clearly if T satisfies Hintikka conditions 1, 2, 4(a), and 4(b) then T is also an accepting computation of A_l on T. On the other hand, if C is an accepting computation of A_l on some tree T, then C itself as an infinite tree over $2^{cl^+(f)}$ that satisfies the desired conditions. By the first rule of A_l, for every node a we have $a = s$, hence $T = C$, and T satisfies Hintikka conditions 1, 2, 4(a), and 4(b). □

The aim of the the next component of A_f is to check satisfaction of Hintikka condition 4(c), which is the condition that deals with box formulas containing the symbol L. Informally (and somewhat imprecisely), the job of this part of the machine is to make sure that if $[L]p$ is written in some node n of the tree, then p is written in every node n', such that $n' = nw$ for some word w in L. The way this automaton works is as follows: At each node it puts all the subformulas that should be checked (i.e., all these p's) on the stack. It then simulates the behavior of M on its path. If at a certain node M would enter an accepting state, $A_\Box$ makes sure that all the subformulas written on the stack are also written at this node. Notice that for this simulation to be carried out it is essential that M is an SM–PDA.

The box automaton for f is

$$A_\Box = \langle Q_\Box, 2^{cl^+(f)}, \Gamma \times 2^{cl^+(f)}, q_0, (z_0, \emptyset), \delta, Q_\Box \rangle$$

where $Q_\Box = Q$, and δ is given by:

- $((q_1, w_1), \ldots, (q_k, w_k)) \in \delta(q, \mathbf{a}, (z, \mathbf{s}))$ if and only if
  1. Either $a = \perp$ or $\mathbf{s} \subseteq \mathbf{a}$, and
  2. For all $1 \leq j \leq n$ and for all $i = j + nl \leq k$ we have:
     (a) If $\rho(q, a_j, z) = (q', \epsilon)$, then $q_i = q'$ and $w_i = \epsilon$,
     (b) if $\rho(q, a_j, z) = (q', z)$, then $q_i = q'$ and $w_i = (z, \mathbf{s} \cup \mathbf{s}')$,
     (c) if $\rho(q, a_j, z) = (q', zz')$, then $q_i = q'$ and $w_i = (z, \mathbf{s} \cup \mathbf{s}')$, $(z', \emptyset)$, and
     (d) if $\rho(q, a_j, z)$ is undefined, then:
        i. If $\rho(q_0, a_j, z_0)$ is undefined, then $q_i = q_0$ and $w_i = (z_0, \emptyset)$;
        ii. if $\rho(q_0, a_j, z_0) = (q', z_0)$, then $q_i = q'$ and $w_i = (z_0, \mathbf{s}')$; and
        iii. if $\rho(q_0, a_j, z_0) = (q', z_0 z)$, then $q_i = q'$ and $w_i = (z_0, \mathbf{s}')$, $(z', \emptyset)$.

     Here, if $\rho(q_0, a_j, z_0)$ is defined, then for all $\neg\langle\alpha\rangle g \in \mathbf{a}$, where α is the finite state automaton $\langle \Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha \rangle$, and for all $s' \in \rho_\alpha(s_{0\alpha}, L)$, we have $\neg\langle\alpha_{s'}\rangle g \in \mathbf{s}'$. If $\rho(q_0, a_j, z_0)$ is undefined, then $\mathbf{s}' = \emptyset$.

In Clause 1 we check whether old box promises that involve the language L are kept, while in Clause 2 we put new such box promises on the stack to be checked later on. (This idea is also present in a somewhat different form in the proof in [KP].) Notice that the stack behavior of $A_\Box$ depends only upon the path in the tree and does not depend at all upon the values of the tree nodes.

LEMMA 4.4. *Let $x \in [k]^*$, $T : [k]^* \to 2^{cl^+(f)}$, and $C_\Box(x) = (q, (z_0, \mathbf{s}_0), \ldots, (z_m, \mathbf{s}_m))$, where $C_\Box$ is a computation of $A_\Box$ over T. Then, for each $w = a_{j_1} \ldots a_{j_l} \in L$ and $r_m = j_m + nl_m \leq k$, the following two conditions hold*:

- $C_\Box(xr_1 \ldots r_l) = (q', (z_0, \mathbf{s}_0), \ldots, (z_{m-1}, \mathbf{s}_{m-1}), (z_m, \mathbf{s}'_m))$.
- $\mathbf{s}'_m$ contains all formulas of the form $\neg\langle\alpha_s\rangle g$, for which $\neg\langle\alpha\rangle g \in T(x)$ and $s \in \rho_\alpha(s_{0\alpha}, L)$.

Proof. Define $\psi_m : Q \times (\Gamma \times 2^{cl^+(f)})^+ \to Q_\Box \times \Gamma^+$ by $\psi(q, (z_0, \mathbf{s}_0) \ldots (z_m, \mathbf{s}_m) \ldots (z_r, \mathbf{s}_r)) = (q, z_m, \ldots, z_r)$. Let $(q_0, \gamma_0) \ldots (q_l, \gamma_l)$ be a computation of M that accepts w. Since w is in L, for all $r_1 = j_1 + nl_1 \leq k$, we have that $\delta(q_0, a_j, z_0)$ is defined; hence, by the definition of $\mathbf{s}'$ in $A_\Box$ we have $C_\Box(xr_1) = (q', (z_0, \mathbf{s}_0), \ldots, (z_m, \mathbf{s}'_m), \gamma')$, where γ' may be empty and $\mathbf{s}'_m$ contains all $\neg\langle\alpha_{s'}\rangle g$ for $s' \in \rho_\alpha(s_{0\alpha}, L)$. We proceed by induction on i, and prove that $\psi_m(C_\Box(xr_1 \ldots r_i)) = (q_i, \gamma_i)$ for all $1 \leq i \leq l$. The base case has just been proven, and the general case follows immediately from the definition of $A_\Box$. This, for $i = l$, proves the lemma. □

PROPOSITION 4.5. *The box automaton $A_\Box$ accepts precisely the trees that satisfy Hintikka condition* 4(c).

Proof. We must show that $A_\Box$ has an accepting computation over some tree T if and only if for all $x \in [k]^*$ we have:

If $\neg\langle\alpha\rangle g \in T(x)$, where α is the finite state automaton $\langle \Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha \rangle$, then for all $s \in \rho_\alpha(s_{0\alpha}, L)$, for all $a_{j_1}, \ldots, a_{j_l} \in L$ and for all $r_m = j_m + nl_m \leq k$, we have $\neg\langle\alpha_s\rangle g \in T(xr_1 \ldots r_l)$ or $T(xr_1 \ldots r_l) = \{\perp\}$.

(*only if*) Suppose, by way of contradiction, that there exist $x_0 \in [k]^*$, $\neg\langle\alpha\rangle g \in T(x_0)$, $s' \in \rho_\alpha(s_{0\alpha}, L)$, and $w = a_{j_1}, \ldots, a_{j_l} \in L$, such that $T(xr_1 \ldots r_l) \neq \{\perp\}$ and $\neg\langle\alpha'_s\rangle g \notin T(xr_1 \ldots r_l)$, for some $r_m = j_m + nl_m \leq k$. Let C be any computation of $A_\Box$. By Lemma 4.4 we know that $C(xr_1 \ldots r_l) = (q', (z_0, \mathbf{s}'_0) \ldots (z_m, \mathbf{s}'_m))$, and $\neg\langle\alpha_s\rangle g \in \mathbf{s}'_m$. However this yields a contradiction to our assumption, since Clause 1 in the definition of $A_\Box$ requires that $\mathbf{s} \subseteq \mathbf{a}$, which implies $\neg\langle\alpha_s\rangle g \in T(xr_1 \ldots r_l)$.

(*if*) If T satisfies the above condition and at each stage of the computation we add to $\mathbf{s}'$ exactly all $\neg\langle\alpha'_s\rangle g$ formulas when $\delta(q_0, a_j, z_0)$ is defined, and add $\emptyset$ otherwise, we get an infinite computation of $A_\Box$ over T. This computation is accepted because $F_\Box = Q_\Box$. □

The third component of A_f deals with diamond formulas. Notice that unlike the box case, all diamond formulas are nonlocal in nature, and thus cannot be handled by the local automaton. The special condition of UDHs is the key for the following construction, since it assures us that each diamond formula is satisfied along a unique path. Hence, all $A_\Diamond$ has to do is guess nondeterministically which successor lies on the appropriate path, and to check that indeed there is a finite path through that successor satisfying the diamond formula.

The diamond automaton for f is

$$A_\Diamond = \langle Q_\Diamond, 2^{cl^+(f)}, \Gamma \times \{0,1\}, (1, \perp, \perp), (z_0, 0), \delta, F_\Diamond\rangle$$

where:

- $Q_\Diamond = \{0,1\} \times cl^+(f) \times (Q \cup \{\perp\})$. The first component is used to indicate acceptance. The second one contains the diamond formula that is being verified, or $\perp$ if no such formula exists. The third component is used to simulate the computation of M_L, and to detect words in the language L;
- $F_\Diamond$ = all triples in $Q_\Diamond$ containing 1 in the first component or $\perp$ in the second;
- define

$$\psi_M(a_j, z) = \begin{cases} \epsilon & \text{if } \Omega(a_j, z) = \epsilon \\ (z, 0) & \text{if } \Omega(a_j, z) = z \\ (z,0)(z',1) & \text{if } \Omega(a_j, z) = zz' \end{cases}$$

and

$$\psi_N(a_j, z) = \begin{cases} \epsilon & \text{if } \Omega(a_j, z) = \epsilon \\ (z, 1) & \text{if } \Omega(a_j, z) = z \\ (z,1)(z',1) & \text{if } \Omega(a_j, z) = zz' \end{cases}$$

$((q_1, w_1), \ldots, (q_k, w_k)) \in \delta((c, \langle\alpha\rangle g, q), \mathbf{a}, (z, b))$ if and only if the following three conditions hold:

1. For all $\langle\beta\rangle h \in \mathbf{a}$ there exist $i = j + nl \leq k$, and a word $v = f_1? \ldots f_m?$ such that $\{f_1, \ldots, f_m\} \subseteq \mathbf{a}$, and one of the following holds:
   (a) $q_i = (c_i, \langle\beta_t\rangle h, \perp)$, $t = \rho_\beta(s_{0\beta}, va_j)$, and $w_i = \psi_N(a_j, z)$;
   (b) $q_i = (c_i, \langle\beta_t\rangle h, p)$, $p = \rho(q_0, a_j, z_0)$, $t = \rho_\beta(s_{0\beta}, v)$, and $w_i = \psi_M(a_j, z)$;
   (c) $t = \rho_\beta(s_{0\beta}, v)$, $t \in F_\beta$ and $h \in \mathbf{a}$.
2. There exists a word $v = f_1? \ldots f_m?$ such that $\{f_1, \ldots, f_m\} \subseteq \mathbf{a}$, and one of the following holds:
   (a) $g \in \mathbf{a}$, and for all $1 \leq j \leq n$ and $i = j + nl \leq k$ we have that the first component of q_i is 1, $w_i = \psi_N(a_j, z)$, and if $\rho_\alpha(s_{0\alpha}, v) \notin F_\alpha$ then $b = 0$, $q \neq \perp$, and $\rho_\alpha(s_{0\alpha}, Lv) \in F_\alpha$;

(b) for all $1 \leq m \leq k$, we have $q_m = (0, q'_m)$, and there exist $i = j + nl \leq k$ such that the following two clauses hold:

i. If $q \neq \perp$ then either $q_i = (c_i, \langle\alpha\rangle g, p)$, $p = \rho(q, a_j, z)$, and $w_i = \psi_N(a_j, z)$, or $b = 0$ and either $q_i = (c_i, \langle\alpha_t\rangle g, \perp)$ where $t = \rho_\alpha(s_{0\alpha}, Lva_j)$ and $w_i = \psi_N(a_j, z)$, or $q_i = (c_i, \langle\alpha_t\rangle g, p)$, where $t = \rho_\alpha(s_{0\alpha}, Lv)$ and $p = \rho(q_0, a_j, z_0)$;

ii. if $q = \perp$ then either $q_i = (c_i, \langle\alpha_t\rangle g, p)$, where $t \in \rho_\alpha(s_{0\alpha}, v)$, $p = \rho(q_0, z_0)$ and $w_i = \psi_M(a_j, z)$, or $q_i = (c_i, \langle\alpha_t\rangle g, \perp)$, where $t \in \rho_\alpha(s_{0\alpha}, va_j)$ and $w_i = \psi_N(a_j, z)$.

3. For all $1 \leq j \leq n$ and $i = j + nl \leq k$, we have $w_i = \psi_N(a_j, z)$ or $w_i = \psi_M(a_j, z)$.

Here again, the idea is much simpler than it appears from looking at the detailed construction. Condition 1 takes care of new diamond formulas. Each such formula is either satisfied in **a** or is written in the machine to be satisfied later. Condition 2 takes care of old promises which are either fulfilled or remain as promises in the machine. Condition 3 deals with the stack. We make sure that all stack operations coincide with those of M_L, and use the extra bit on the stack to indicate the beginning of new simulations of M_L.

LEMMA 4.6. *Let f be an* APDL$_L$ *formula with atomic programs $a_1, \ldots, a_l$, and let $T : [k]^* \to 2^{cl(f)\cup\{\perp\}}$ be a k-ary tree. Then T satisfies Hintikka condition* 3 *if and only if for all $x \in [k]^*$ if $\langle\alpha\rangle g \in T(x)$, where α is the finite state automaton $\langle\Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha\rangle$, then there are nodes $u_0 \ldots u_l$ of $[k]^*$, states $s_0, t_0, \ldots, s_l, t_l$ of S_α, atomic programs $a_{j_{i_1}}, \ldots, a_{j_{l_m}}$, and numbers $r_{i_m} = j_{i_m} + nl_{i_m} \leq k$, such that the following conditions hold*:

1. $u_0 = x$, $s_o = s_{0\alpha}$, $t_l \in F_\alpha$;
2. *either* $s_i \in \rho_\alpha(t_{i-1}, a_{j_{i_1}})$ *and* $u_i = u_{i-1}r_{i_1}$, *or* $s_i \in \rho_\alpha(t_{i-1}, L)$ *and* $u_i = u_{i-1}j_{i_1} \ldots j_{i_{l_i}}$, *where* $a_{r_{i_1}} \ldots a_{r_{i_{l_i}}} \in L$;
3. *for* $0 \leq i \leq l$, *there exists a word* $w = g_1? \ldots g_m?$, $m \geq 0$, *such that* $\{g_1, \ldots g_m\} \subseteq T(u_i)$ *and* $t_i \in \rho_\alpha(s_i, w)$;
4. $g \in T(u_k)$.

Proof. The new condition is nothing but a rephrasing of condition 3 of the definition of a Hintikka tree, where we have separated the tests from the programs in the automaton α. The details of the proof are straightforward and are left to the reader. □

LEMMA 4.7. *Let C be an accepting computation of $A_\diamond$ over some tree T, such that $C(x) = (c_x, \langle\alpha\rangle g, \perp), (z_0, c_0), \ldots, (z_l, c_l)$, and $C(xi) = (c_y, \langle\alpha_t\rangle g, p), (z_0, c_0), \ldots, (z_l, c'_l, \gamma')$, where $p \neq \perp$ and γ' may be empty. Then there exists a word $w = a_{j_1}a_{j_2} \ldots a_{j_m} \in L$ and numbers $r_i = j_i + nl_i \leq k$, such that $C(xir_1 \ldots r_m) = (c', \langle\alpha_t\rangle g, q'), (z_0, c_0), \ldots, (z_l, 0)$.*

Proof. The lemma actually states that $A_\diamond$ simulates the computation of M. The proof is obtained by straightforward induction on the length of w, using the definition of $A_\diamond$, and is left to the reader. □

PROPOSITION 4.8. *The automaton $A_\diamond$ accepts precisely the trees that satisfy both Hintikka condition* 3 *and the special condition of a* UDH.

Proof. Let C be an accepting computation of $A_\diamond$ over some tree T. We must show that T follows condition 3 of Definition 2.2 together with conditions 1 and 2 of Definition 2.4. Let $x \in [k]^*$ and $\langle\alpha\rangle g \in T(x)$, where α is the finite state automaton $\langle\Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha\rangle$. We must show the existence of appropriate nodes $u_0 \ldots u_l$ of $[k]^*$ and states $s_0, t_0, \ldots, s_l, t_l$ of S_α.

Denote by $w_{x,d}$, $t_{x,d}$, $i_{x,d}$, and $j_{x,d}$ the appropriate w, t, i, and j that exist for $\mathbf{a} = T(x)$ and $d = \langle\alpha\rangle g$ by the definition of $A_\diamond$. Let $u_0 = x$, $s_0 = s_{0\alpha}$, $t_0 = \rho_\alpha(s_0, w_{x,d})$, and u_1, and s_1 depend upon 1 in the definition of $A_\diamond$ in the following way: If 1(a) holds, then $u_1 = xi_{x,d}$

and $s_1 = \rho_\alpha(t_0, a_{j_{x,d}})$. If 1(b) holds, then $u_1 = xw$, where w is the word guaranteed to exist by Lemma 4.7, and $s_1 = \rho_\alpha(t_0, L)$. If 1(c) holds, then $l = 0$ and no u_1 or s_1 exist.

For the general case, assume that $\langle\alpha_{s_i}\rangle g = d$ is in the state in $C(u_i)$ and define $t_i = \rho_\alpha(s_i, w_{x,d})$. Both u_{i+1} and s_{i+1} depend upon 2 in the definition of $A_\diamond$ in the following way: If 2(a) holds then $l = i$ and no u_{i+1} or s_{i+1} exist. Otherwise, if 2(b) holds then $u_{i+1} = xi_{u_i,d}$ and $s_{i+1} = \rho_\alpha(t_i, a_{j_{u_i,d}})$; if 2(c) holds then $u_{i+1} = xw$, where w is the word that exists by Lemma 4.7, and $s_{i+1} = \rho_\alpha(t_i, L)$.

Clearly, these $u_0, \ldots, u_l, s_0, t_0, \ldots, s_l, t_l$ satisfy conditions 1–3 of Lemma 4.6. For $l = 0$, the requirement $h \in \mathbf{a}$ of 1(c) in the definition of $A_\diamond$ yields $g \in T(u_l)$, and for $l \geq 1$ the fact that u_l satisfies condition 4 of lemma 4.6 is obtained from the requirement $g \in \mathbf{a}$ of 2(c) in that definition.

It remains to exhibit the existence of an appropriate Φ. Let us define, for $x \in [k]^*$,

$$\Phi(x) = \begin{cases} \langle\alpha\rangle g & \text{if the state in } C(x) \text{ is } \langle\alpha_t\rangle g \text{ for some state } t \\ \bot & \text{if the state in } C(x) \text{ is } \bot. \end{cases}$$

This Φ satisfies the special condition of a UDH, since, for $1 \leq i \leq l$, all the u_i defined above have the same diamond formula in the state of $C(u_i)$.

For the other direction, suppose T follows condition 3 of the definition of a Hintikka tree, as well as the special condition of a UDH. We must show that there exists an accepting computation C of $A_\diamond$ over T. We construct $C(x)$ by induction on $[k]^*$: If $C(x) = (\mathbf{a}, (c_x, \langle\alpha\rangle g, p), ((z_0, c_0), \ldots, (z_l, c_l)))$, then we apply $\delta((c_x, \langle\alpha\rangle g, p), \mathbf{a}, (z_e, c_e)) = ((q_1, w_1), \ldots, (q_k, w_k))$, where, for all $1 \leq i \leq k$, q_i is set by the value of $\Phi(xi)$, and w_i follows 3 in the definition of δ.

Clearly C is indeed a computation of $A_\diamond$ over T. Now, if C were not accepting, by König's infinity lemma there should exist an infinite path containing no accepting state, which contradicts our assumption on T. □

LEMMA 4.9. *There is a pushdown k-ary tree automaton A_f, such that $L(A_f) = L(A_l) \cap L(A_\diamond) \cap L(A_\Box)$, and the size of A_f is at most $|A_l| \cdot |A_\diamond| \cdot |A_\Box|$.*

Proof. Define $A_f = \langle Q_f, 2^{cl^+(f)}, \Gamma_f, q_{0f}, z_{0f}, \delta_f, F_f\rangle$ as follows:

- $Q_f = Q_l \times Q_\Box \times Q_\diamond$;
- $q_{0f} = N_f \times q_{0\Box} \times q_{0\diamond}$;
- $F_f = Q_l \times Q_\Box \times F_\diamond$;
- $\Gamma_f = \Gamma_\Box \times \Gamma_\diamond$;
- $z_{0f} = z_{0\Box} \times z_{0\diamond}$;
- δ_f is the Cartesian product of the appropriate δ functions.

Since all the states of both the local automaton and the box automaton are accepting states, and since we have taken the third component of A_f to be $F_\diamond$, the accepted language is as required. Also, the size bound is immediate. Hence we only have to show that this definition indeed describes a tree pushdown automaton; in other words, we must show that the transition function δ_f is well defined. This is due to the "simple minded" characterization of the language L. More formally, for each $x \in [k]^*$, and for each $i_m = j_m + nl_m \leq k$, the stack operations of $A_\diamond$ are the same as the stack operations of $A_\Box$, since they both depend only upon the letter a_{j_m}. □

The above lemma, together with the preceding results, yields the following proposition.

PROPOSITION 4.10. *Given a formula f in* APDL$_L$, *where L is an* sm-cfl, *we can construct a* PTA *A_f such that f has a model if and only if there is some tree T accepted by A_f.*

Theorem 1 now follows.

Simplemindedness can be applied to stack automata, not just to pushdown automata. These generalized simpleminded languages contain

$$L_5 = \{a^i b^i c^i \mid i \geq 0\}.$$

However, they do not yield to the construction just given. The main difficulty is in the construction of the box automaton, and is due to the fact that some of the simulation of M could start when the stack of the STA is not empty.

For the stack automata case, we have had to resort to a different kind of construction, as we now show.

5. Unique prefix stack languages.

In this section we define a subfamily of the languages accepted by stack automata, called *unique prefix*. Given a formula f in APDL$_L$, where L is a unique prefix language, we will exhibit a construction similar to that of the previous section to define an STA A_f that accepts precisely the trees that are UDH of the formula f.

Let L' be a language accepted by a deterministic finite-state one way stack automaton over some alphabet Σ, and let $\cent \notin \Sigma$. Then $L = \cent L' = \{\cent w \mid w \in L'\}$ is a *deterministic unique-prefix stack language* (a dup-sl, for short), and the special letter $\cent$ is denoted by $\cent_L$. We again refer the reader to [Ka] for a description of stack automata. A more formal definition appears in [WW, pp. 31–33]. Our stack automata accept by final state, but we require that in an accepting state the automaton is at the top of the stack.

EXAMPLE 5.1. Let $M = \langle Q, \Sigma, \Gamma, q_o, z_0, \delta, \{q_0, q_c\}\rangle$ be a deterministic stack automaton, where $Q = \{q_0, q_a, q_b, q_c\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{z, z_0\}$, and the transition function δ is given by:

$$\delta(q_0, a, z_0) = (q_a, push(z))$$

$$\delta(q_a, a, z) = (q_a, push(z))$$

$$\delta(q_a, b, z) = (q_b, md)$$

$$\delta(q_b, b, z) = (q_b, md)$$

$$\delta(q_b, c, z_0) = (q_c, mu)$$

$$\delta(q_c, c, z) = (q_c, mu).$$

δ is undefined elsewhere. Since the automaton only accepts when its head is at the top of the stack, the language accepted by M is $L_5 = \{a^i b^i c^i \mid i \geq 0\}$. Hence, $L = \cent L_5 = \{\cent a^i b^i c^i \mid i \geq 0\}$ is a dup-sl.

Many noncontext-free languages are dup-sl's, such as $\{\cent a_1^i a_2^i \dots a_k^i \mid i \geq 0\}$ for any fixed k and $\{\cent wcw \mid w \in \Sigma^* \textit{ and } c, \cent \notin \Sigma\}$.

Our goal now is to prove the following result, which implies Theorem 2.

PROPOSITION 5.2. *Given a formula f in* APDL$_L$, *where L is a* dup-sl, *one can construct an* STA A_f *such that f has a model if and only if there is some tree T accepted by A_f.*

Let $M_L = \langle Q, \Sigma, \Gamma, q_{0L}, z_0, \rho, F_L\rangle$ be the deterministic stack automaton that accepts the dup-sl L, and let f be a formula in APDL$_L$.

The local automaton for f is A_l of the previous section.

PROPOSITION 5.3. *The automaton A_l accepts precisely the trees that satisfy Hintikka conditions* 1, 2, 4(a), *and* 4(b).

Proof. The proof is similar to the proof of Proposition 4.3. □

The box automaton in this case is much simpler than the one defined in the previous section for sm-cfl, since the unique prefix property guarantees that at any node at most one word of L should be checked. Thus, in this case, the automaton need not keep the formulas to be checked in it stack, and it can store them as part of its internal state.

The box automaton for f is

$$A_\Box = \langle Q \cup \{\perp\} \times 2^{cl(f)}, 2^{cl^+(f)}, \Gamma, \perp, z_0, \delta, Q \cup \{\perp\} \times 2^{cl(f)} \rangle$$

where δ is given by:

- $(((q_1, \mathbf{s}_1), b_1), \ldots, ((q_k, \mathbf{s}_k), b_k)) \in \delta((q, \mathbf{s}), \mathbf{a}, z)$ if and only if for all $1 \leq j \leq n$ the following conditions hold:
  1. if $q \in F_L$, then $\mathbf{s} \subseteq \mathbf{a}$ or $\mathbf{a} = \{\perp\}$, and
  2. if $a_j \neq ¢_L$, then for all $g \in \mathbf{s}$ and for all $i = j + nl \leq k$, we have $g \in \mathbf{s}_i$, if $q \neq \perp$, then $(q_i, b_i) = \rho(q, a_j, z)$, and if $q = \perp$, then $q_i = \perp$ and $b_i = sp$;
  3. if $a_j = ¢_L$, then for all $i = j + nl \leq k$ the following hold:
     (a) For all $\neg\langle\alpha\rangle g \in \mathbf{a}$, where α is the finite state automaton $\langle \Sigma_\alpha, S_\alpha, \rho_\alpha, s_{0\alpha}, F_\alpha \rangle$, and for all $s' \in \rho_\alpha(s_{0\alpha}, L)$, we have $\neg\langle\alpha_{s'}\rangle g \in \mathbf{s}_i$;
     (b) either $b_i = push(z_0 z)$ and $(q_i, push(z)) = \rho(q_0, ¢_L, z_0)$, or $b_i = push(z_0)$ and $(q_i, sp) = \rho(q_0, ¢_L, z_0)$. (Note that, since $a_j = ¢_L$, these are the only valid possibilities.)

In condition 1 we check whether old box promises that involve the language L are kept. In condition 2 we "store" such old box promises in the machine's internal state, to be checked later on. In condition 3 we deal with new promises. Notice that when a new promise appears, we forget about all old promises. This is a crucial point, and is made possible by the unique prefix characteristic of L. Another point that arises here is that this automaton may be required to perform a push when its head is not at the top of the stack. By convention, this instruction is taken to abbreviate a sequence of instructions to get to the top of the stack and then perform a push.

PROPOSITION 5.4. *The automaton $A_\Box$ accepts precisely the trees that satisfy Hintikka condition* 4(c).

Proof. First note that from Clause 2 in the definition of the δ function of $A_\Box$ we have: If $u_0, \ldots, u_l$ are nodes in $[k]^*$ such that for $1 \leq i \leq l$, $u_i = u_{i-1} r_i$ for some $r_i = j_i + nl_i \leq k$, and $a_{j_i}, \ldots, a_{j_l} \neq ¢_L$, then for any computation C of $A_\Box$ and for any formula g, if $g \in C(u_0)$ then $g \in C(u_l)$. We now show that for similar $u_0, \ldots, u_l$ with $a_{j_1} \ldots a_{j_l} \in L$, if q is the state of M_L after the ith input symbol, then for any computation C of $A_\Box$, $C(u_i) = (q, \mathbf{s}, \ldots)$ for some $\mathbf{s} \in 2^{cl^+(f)}$. This is proved by induction on l. For the base case, $l = 0$, the result follows from Clause 3(b) in the definition of δ. (Recall that L is a deterministic unique prefix language, hence $a_{j_1} = ¢_L$.) The inductive step follows from Clause 2 in the definition of δ. The above, together with Clauses 3 and 1, guarantees that any computation C of $A_\Box$ on a tree T is accepted if and only if T satisfies Hintikka condition 4(c). □

The third component in this case is very similar to $A_\Diamond$ of the previous section. The differences are merely technical.

The diamond automaton for f is

$$A_\Diamond = \langle Q_\Diamond, 2^{cl^+(f)}, \Gamma, (1, \perp, \perp, 0), z_0, \delta, F_\Diamond \rangle$$

where:

- $Q_\Diamond = \{0, 1\} \times cl^+(f) \times (Q \cup \{\perp\}) \times \{0, 1\}$. The first component is used to indicate acceptance. The second one contains the diamond formula that is being verified, or $\perp$ if no such formula exists. The third component is used to simulate the computation of M_L, and to detect words in the language L. The last component indicates whether the automaton is verifying a formula that involves L.
- $F_\Diamond$ = all quadruples in $Q_\Diamond$ containing 1 in the first component or $\perp$ in the second;
- $((q_1, b_1), \ldots, (q_k, b_k)) \in \delta((c, \langle\alpha\rangle g, q, d), \mathbf{a}, z)$ if and only if the following three conditions hold:

1. For all $\langle\beta\rangle h \in \mathbf{a}$ there exist $i = j + nl \leq k$, and a word $v = f_1? \dots f_m?$ such that $\{f_1, \dots, f_m\} \subseteq \mathbf{a}$, and one of the following holds:
   (a) $t = \rho_\beta(s_{0\beta}, v)$, $t \in F_\beta$ and $h \in \mathbf{a}$;
   (b) $q_i = (c_i, \langle\beta_t\rangle h, p, 0)$, $t = \rho_\beta(s_{0\beta}, va_j)$ and $p = \rho(q, a_j, z)$;
   (c) $a_j = ¢_L$, $q_i = (c_i, \langle\beta_t\rangle h, p, 1)$, $p = \rho(q_0, a_j, z_0)$ and $t = \rho_\beta(s_{0\beta}, v)$;
2. There exists a word $v = f_1? \dots f_m?$, such that $\{f_1, \dots, f_m\} \subseteq \mathbf{a}$, and one of the following holds:
   (a) $g \in \mathbf{a}$, for all $1 \leq i \leq k$, we have $q_i = (1, q_i')$, and either $\rho_\alpha(s_{0\alpha}, v) \in F_\alpha$ and $d = 0$ or $q \in F_L$, $d = 1$, and $\rho_\alpha(s_{0\alpha}, Lv) \in F_\alpha$;
   (b) for all $1 \leq m \leq k$, we have $q_m = (0, q_m')$ and there exist $i = j + nl \leq k$ such that the following two clauses hold:
      i. if $d = 1$ then one of the following holds:
         A. $q_i = (c_i, \langle\alpha\rangle g, p, 1)$, $p = \rho(q, a_j, z)$, or $q_i = (c_i, \langle\alpha_t\rangle g, p, 0)$, where $t = \rho_\alpha(s_{0\alpha}, Lva_j)$ and $q \in F_L$;
         B. $q_i = (c_i, \langle\alpha_t\rangle g, p, 1)$, where $t = \rho_\alpha(s_{0\alpha}, Lv)$, $p = \rho(q_0, a_j, z)$ and $a_j = ¢_L$;
      ii. if $d = 0$ then one of the following holds:
         A. $q_i = (c_i, \langle\alpha_t\rangle g, p, 1)$, where $t \in \rho_\alpha(s_{0\alpha}, v)$, $p = \rho(q_0, a_j, z_0)$ and $a_j = ¢_L$;
         B. $q_i = (c_i, \langle\alpha_t\rangle g, p, 0)$, where $t \in \rho_\alpha(s_{0\alpha}, va_j)$ and $p = \rho(q, a_j, z)$.
3. For all $1 \leq j \leq n$ and $i = j + nl \leq k$, we have:
   (a) if $a_j \neq ¢_L$, then if $\rho(q, a_j, z) = (p, b)$ then $b_i = b$, and if $\rho(q, a_j, z)$ is undefined then $b_i = sp$;
   (b) if $a_j = ¢_L$, then if $\rho(q_0, a_j, z_0) = (p, sp)$ then $b_i = push(z_0)$, and if $\rho(q_0, a_j, z_0) = (p, push(z))$ then $b_i = push(zz_0)$.

In this definition, $p = \rho(q, a, z)$ is short for:

$$p = \begin{cases} p' & \text{if } q \neq \perp \text{ and } \rho(q, a, z) = (p', b) \text{ for some } b \\ \perp & \text{otherwise.} \end{cases}$$

Condition 1 takes care of new diamond formulas. Each such formula is either satisfied in **a** or is written in the machine to be satisfied later. Condition 2 takes care of old promises which are either fulfilled or remain as promises in the machine. Condition 3 deals with the stack. We make sure that all stack operations coincide with those of M_L. This will allow us to use $A_\diamond$ as a part of A_f. Notice that the same problem of performing a push when the head is not on top of the stack exists here; it is treated in the same way as for $A_\square$.

LEMMA 5.5. *Let C be an accepting computation of $A_\diamond$ over some tree T. Let $C(x) = ((c_x, \langle\alpha\rangle g, p, d), z_0, \dots, z_l)$, and $C(xi) = ((c_y, \langle\alpha_t\rangle g, p, 1), z_0, \dots, z_l\gamma)$, where $p \neq \perp$, γ may be empty, $i = j + nl \leq k$, and $a_j = ¢_L$. Then there exists a word $w = a_j a_{j_2} \dots a_{j_m} \in L$, and numbers $r_i = j_i + nl_i \leq k$, such that $C(xir_1 \dots r_m) = ((c', \langle\alpha_t\rangle g, q', d'), z_0, \dots, z_l, \gamma')$, where $q' \in F_L$.*

Proof. This lemma actually states that $A_\diamond$ simulates the computation for M_L. The proof is obtained using the definition of $A_\diamond$ by straightforward induction on the length of w, and is left to the reader. □

PROPOSITION 5.6. *The automaton $A_\diamond$ accepts precisely the trees that satisfy both Hintikka condition* 3, *and the special condition of a* UDH.

Proof. Since in this case the automaton $A_\diamond$ is almost identical to that defined for the simple minded languages of the previous section, the proof here follows the proof of Proposition 4.8. □

LEMMA 5.7. *There is a stack k-ary tree automaton A_f, such that $L(A_f) = L(A_l) \cap L(A_\diamond) \cap L(A_\Box)$, and the size of A_f is at most $|A_l| \cdot |A_\diamond| \cdot |A_\Box|$.*

Proof. Define $A_f = \langle Q_f, 2^{cl^+(f)}, \Gamma_f, q_{0f}, z_{0f}, \delta_f, F_f \rangle$ exactly as in the proof of Lemma 4.9. Here, too, we only have to show that the transition function δ_f is well defined. Since both $A_\diamond$ and $A_\Box$ simulate the behavior of M_L, they both perform the same operation on the stack. More formally, we show by induction on $x \in [k]^*$ that the state of Q and the stack in $C_\Box(x)$ are identical to those of $C_\diamond(x)$. Here, $C_\Box(x)$ and $C_\diamond(x)$ stand for computations of $A_\Box$ and $A_\diamond$, respectively. This proof is straightforward and shows that the transition function is well defined. □

The above lemma, together with the preceding results, yields Proposition 5.2, from which Theorem 2 follows.

The construction depends strongly upon the special prefix $¢_L$. Without it, the definition of the box automaton cannot be used.

6. A delicate combination. We now combine the two families of automata, constructed in the proofs of Theorems 1 and 2, in order to prove the decidability of PDL_{L_5}. However, rather than deal directly with the language L_5, we prove decidability for a somewhat more general family of languages.

Let L be a language accepted by a deterministic stack automaton M over some alphabet Σ. Assume there exists a partition $\Sigma = \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$, such that for every word $w \in L$, we have $w = w_1 w_2$, with $w_i \in \Sigma_i^*$ for $i = 1, 2$. Let C be an accepting computation of M on w. We call the part of C working on w_1 part 1 of the computation, and the rest of C is called part 2.

PROPOSITION 6.1. *If part* 1 *of all accepting computations of M has no mu (move up) moves, follows the rules of simpleminded machines, and always ends up with the head of the stack at the bottom position, and if, in addition, part 2 of the computation has only mu (move up) or sp (stay put) moves, then* PDL_L *is decidable.*

Sketch of proof. The automaton we use here has two modes of operation: The first one works like the automaton A_f from §4 (cf. Lemma 4.9), and the second mode is similar to the automaton A_f from §5 (cf. Lemma 5.7). It starts in the first mode and changes modes when it starts seeing tree symbols from Σ_2. Since at this point M is guaranteed to be in the bottom position, the tree automaton "knows" what promises have to be kept, and can transpose them to its internal state when switching modes. The fact that in part 1 of all accepting computations of M there are no *mu* moves, and that it is simple minded, guarantees that indeed the right promises are checked. On the other hand, the fact that part 2 of the computation of M has only *mu* or *sp* moves guarantees that the second mode will operate correctly and will not get into forbidden parts of the stack. A careful and tedious construction, similar the those of §§4 and 5, proves that this automaton, which is indeed a TSA, has an accepting computation if and only if the given formula in PDL_L has a model. We omit the details which can be obtained from those provided in the proofs of Theorems 1 and 2. □

In order to complete the proof of Theorem 3, we prove the following.

PROPOSITION 6.2. *There exists a stack automaton that follows the conditions of Proposition* 6.1 *and recognizes the language L_5.*

Proof. Let $\Sigma_1 = \{a, b\}$, and $\Sigma_2 = \{c\}$. M operates as follows: When seeing an "a" in the input it pushes it onto the stack; when seeing a "b" it performs "move down"; when seeing a "c" it performs "move up," and accepts if it is at the top of the stack. Clearly M is a simpleminded stack automaton, that has *no* "move up" moves in part 1 of the computation, and has *only* "move up" moves in part 2 of the computation. Furthermore, it is easy to verify that M accepts precisely the language L_5. □

7. Discussion. Decidability of the validity problem for PDL_L, where L is in some class C of languages, may be viewed as a general problem about C, like the equivalence or emptiness problems. Just as it is of prime interest to characterize those classes C for which these latter problems are decidable, so is it of interest to characterize those for which the PDL *problem*, as we may call it, is decidable. Apart from its interest as a problem concerning reasoning about propositional programs, when phrased in this manner it becomes a decision problem for formal languages. While we feel that the present paper provides a significant step forward, we are far from having a complete solution.

Two specific aspects are still open. First, we do not know enough about the borderline. It would be nice to be able to provide general *negative* results, stating that for any language L *not* accepted by machines of some special form, PDL_L is *un*decidable. Second, our techniques provide no new information regarding one-letter alphabets; Theorem 1 does not help, since PDAs do not accept any nonregular one-letter languages, and our proof of Theorem 2 requires the prefix symbols. In this arena there are only negative results, specifically, that of [HP] about $\{a^{(2^i)} | i \geq 0\}$ and its extension to $\{a^{(k^i)} | i \geq 0\}$ for any fixed k. Is there *any* nonregular (and therefore noncontext-free) one-letter language L for which PDL_L is decidable? Is there any one-letter language L exhibiting *polynomial* growth in the lengths of its words, for which PDL_L is undecidable?

Acknowledgment. We would like to thank the anonymous referees for several very helpful comments which helped to improve the presentation.

REFERENCES

[BV] B. von Braunmühl and R. Verbeek, *Input driven languages are recognized in* $\log n$ *space*, preprint, Bonn, Germany, 1984.

[FL] M. J. Fischer and R. E. Ladner, *Propositional dynamic logic of regular programs*, J. Comput. System Sci., 18 (1979), pp. 194–211.

[H] D. Harel, *Dynamic Logic*, in Handbook of Philosophical Logic Vol. II, D. Gabbay and F. Gunthner, eds., Reidel, Dordrecht, the Netherlands, 1983, pp. 497–603.

[Ha] M. A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[HP] D. Harel and M. S. Paterson, *Undecidability of* PDL *with* $L = \{a^{2^i} | i \geq 0\}$, J. Comput. System Sci., 29 (1984), pp. 359–365.

[HPS] D. Harel, A. Pnueli, and J. Stavi, *Propositional dynamic logic of nonregular programs*, J. Comput. System Sci., 26 (1983), pp. 222–243.

[HR] D. Harel and D. Raz, *Deciding emptiness for stack automata on infinite trees*, Inform. and Comput., to appear.

[HS] D. Harel and R. Sherman, *Propositional dynamic logic of flowcharts*, Inform. Control, 64 (1985), pp. 119–135.

[Ka] R. Y. Kain, *Automata Theory: Machines and Languages*, McGraw-Hill, New York, 1972.

[KP] T. Koren and A. Pnueli, *There exist decidable context-free propositional dynamic logics*, Proc. Symposium on Logics of Programs, Lecture Notes in Computer Science, Vol. 164, Springer-Verlag, New York, 1983, pp. 290–312.

[KT] D. Kozen and J. Tiuryn, *Logics of programs*, in Handbook of Theoretical Computer Science, Vol. B, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 789–840.

[P] R. J. Parikh, *On context-free languages*, J. Assoc. Comput. Mach., 13 (1966), pp. 570–581.

[P1] V. R. Pratt, *Semantical considerations on Floyd–Hoare logic*, 17th IEEE Symposium on Foundation Comput. Sci., 1976, pp. 109–121.

[P2] ———, *A near optimal method for reasoning about action*, J. Comput. System Sci., 20 (1980), pp. 231–254.

[S] A. Saudi, *Pushdown automata on infinite trees and omega-Kleene closure of context-free tree sets*, Proc. Math. Found. of Comput. Sci., Lecture Notes in Computer Science, Vol. 379, Springer-Verlag, New York, 1989, pp. 445–457.

[St] R. S. Streett, *Propositional dynamic logic of looping and converse is elementarily decidable*, Inform. Control, 54 (1982), pp. 121–141.

[VW] M. Vardi and P. Wolper, *Automata-theoretic techniques for modal logics of programs*, J. Comput. System Sci., 32 (1986), pp. 183–221.

[WW] K. Wagmer and G. Wechsung, *Computational Complexity*, D. Reidel, Dordrecht, the Netherlands, 1986.

SIAM J. COMPUT.
Vol. 22, No. 4, pp. 875–887, August 1993

MULTIPLE COMMUNICATION IN MULTIHOP RADIO NETWORKS*

REUVEN BAR-YEHUDA†, AMOS ISRAELI‡, AND ALON ITAI§

Abstract. Two tasks of communication in a multihop synchronous radio network are considered: Point-to-point communication and broadcast (sending a message to all nodes of a network). Efficient protocols for both problems are presented. Even though the protocols are probabilistic, it is shown how to acknowledge messages deterministically.

Let n, D, and Δ be the number of nodes, the diameter and the maximum degree of our network, respectively. Both protocols require a setup phase in which a BFS tree is constructed. This phase takes $O((n + D \log n) \log \Delta)$ time.

After the setup, k point-to-point transmissions require $O((k + D) \log \Delta)$ time on the average. Therefore the network allows a new transmission every $O(\log \Delta)$ time slots. Also, k broadcasts require an average of $O((k + D) \log \Delta \log n)$ time. Hence the average throughput of the network is a broadcast every $O(\log \Delta \log n)$ time slots. Both protocols pipeline the messages along the BFS tree. They are always successful on the graph spanned by the BFS tree. Their probabilistic behavior refers only to the running time.

Using the above protocols the ranking problem is solved in $O(n \log n \log \Delta)$ time. The performance analysis of both protocols constitutes a new application of queueing theory.

Key words. radio networks, broadcast, point-to-point routing, distributed algorithms, average case analysis, queueing theory, randomized algorithms

AMS subject classifications. 05C85, 60K25, 68M10, 68Q22

1. Introduction.

A *radio network* is a network of processors which communicates using radio. An important feature of radio communication is that if a receiver is in the range of two or more transmitting stations, then, due to interferences, some messages might not be received. A radio communication network is *single-hop* if all nodes are in transmission range of each other. Otherwise it is *multihop*. Thus, sending a message between two stations in a multihop network might involve transmissions through intermediate stations.

Most real life radio networks for data communication are quite limited. In fact, most such networks are single-hop and most existing multihop networks resort to the tree topology. This situation looks rather odd considering the ease with which radio networks can be initiated and the flexibility and modularity of their operation.

A new approach for controlling the activity in multihop radio networks was presented in the work of [3], where an efficient broadcast protocol is presented. Their method gives a new way of looking at radio networks. However, they do not provide protocols for many important network tasks.

In the present work we use some of the ideas presented in [3], together with some new ideas to get very efficient protocols for two important and practical tasks. The tasks are k point-to-point transmission and k-broadcast. *Point-to-point transmission* is the task of sending a message from one station to another. *Broadcast* is a task initiated by a single station called the *source,* which transmits a *message* to all stations in the network. A k-point-to-point transmission (k-broadcast) is a task which consists of k point-to-point transmissions (k-broadcasts).

*Received by the editors February 21, 1989; accepted for publication (in revised form) May 15, 1992. A preliminary version of this paper was presented in the Symposium on Principles of Distributed Computing, Edmonton, Canada, 1989.

†Department of Computer Science, Technion–Israel Institute of Technology, Haifa, 32000, Israel. This author was partially supported by Technion V.P.R. Fund–Albert Einstein Research Fund.

‡Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa, 32000, Israel. This author was partially supported by the Technion V.P.R. Fund, New York Metropolitan Fund, and the Japanese TS Research Fund.

§Department of Computer Science, Technion–Israel Institute of Technology, Haifa, 32000, Israel. This author was partially supported by Technion V.P.R. Fund.

Besides the theoretical interest, these tasks constitute a major part of real life multihop radio network.

1.1. Model description. Our model consists of an undirected graph whose nodes represent stations (i.e., processors) and whose edges indicate possible communication, (i.e., an edge between two nodes implies that the corresponding processors are within range and within line of sight of each other). The processors have distinct IDs. Initially, each processor knows its local neighborhood (i.e., the identity of its neighbors), the size of the network, n, and an upper bound, Δ, on the maximum degree of the network. It need not have any additional information of the topology of the network.

The processors may transmit and receive messages of length $O(\log n)$ and communicate in synchronous time slots subject to the following rules. In each time slot, each processor acts either as a *transmitter* or as a *receiver.* A processor acting as a receiver is said to receive a message in time slot t if exactly one of its neighbors transmits in time slot t. The message received is the one sent. Since communication is synchronous the only difficulty in routing messages, in this model, is the possibility of *conflicts*; that is, situations when several neighbors of a processor transmit simultaneously and it receives nothing. More specifically, we assume that there is no conflict detection (see [4]).

Throughout the paper, n denotes the actual number of processors, Δ the maximum degree and D the diameter of the network.

1.2. Main results. Efficient protocols for k-point-to-point communication and k-broadcast are presented. Even though the protocols are randomized, it is shown how to acknowledge messages deterministically. Both protocols require a setup phase in which a BFS tree is constructed. This phase takes $O((n + D\log n)\log\Delta)$ time.

After the setup, k point-to-point transmissions require $O((k + D)\log\Delta)$ time on the average. Therefore the network allows a new transmission every sequence of $O(\log\Delta)$ time slots. Also, k broadcasts require an average of $O((k+D)\log\Delta\log n)$ time. Hence the average throughput of the network is a broadcast every sequence of $O(\log\Delta\log n)$ time slots.

Both protocols pipeline the messages along the edges of a BFS tree. They are always successful on the graph spanned by this tree. Their probabilistic behavior refers only to the running time. The performance analysis of both protocols constitutes a new application of queueing theory.

1.3. Previous work. Chlamtac and Kutten [7] showed that, given a network and a designated source, finding an optimal broadcast schedule (i.e., broadcasting schedule that uses the minimum number of time slots) is NP-hard. They also routed messages through a (not necessarily BFS) tree, and discussed "implicit acknowledgments." Their acknowledgments are conducted in the absence of conflicts, and are achieved at the cost of increasing the time of a single point-to-point communication to $O(D\Delta)$.

Chlamtac and Weinstein [8] presented a polynomial-time (centralized) algorithm for constructing a broadcast schedule which uses $O(D\log^2 n)$ time slots. This centralized algorithm can be implemented in a distributed system assuming the availability of special control channels, but the number of control messages sent may be quadratic in the number of nodes of the network [16].

In a different context, Birk [2] independently discovered an acknowledgment mechanism similar to ours.

Bar-Yehuda, Goldreich, and Itai [3] described a randomized single-source broadcast protocol. To ensure that with probability $1-\varepsilon$ all nodes receive the message, the protocol requires $O((D + \log(n/\varepsilon))\log\Delta)$ time slots. For $D = 2$, they also showed an $\Omega(n)$ lower bound for deterministic protocols. Thus, for this problem there exist randomized protocols that are much more efficient than any deterministic one. For $D = 2$, Alon et al. [1] showed an $\Omega(\log^2 n)$ lower bound, which matches the upper bound of [8] and [3].

In [4] Bar-Yehuda, Goldreich, and Itai discuss several models of radio communication and show how to detect conflicts and simulate a single-hop network. Thus they show how to use protocols designed for the ETHERNET in a multihop network [6], [1].

1.4. Protocol outline. Both protocols depend on the existence of a BFS tree of the graph which is constructed in a setup phase. In the beginning of the setup part, a leader is chosen. Once the leader is chosen, it initiates the construction of a BFS tree whose root is the leader. For this purpose the protocols of [3] and [4] are used. The setup phase is conducted only once, after which any series of point-to-point transmissions or broadcasts might be performed.

The broadcast process is reactive (continuous); it is invoked whenever a source originates a message to broadcast. It consists of two subprotocols: *Collection*—sending the messages from the sources to the root of the BFS tree and *distribution*—sending the messages from the root to all the processors of the network.

The point-to-point transmission is also reactive. It is invoked whenever a processor wishes to send a message. A message from node u to v travels first up the tree. Once the message reaches a common ancestor of u and v it continues downwards towards v. The protocols for both directions are very similar to the collection protocol and are fully described in §5.

Since both protocols are reactive, it is not possible to wait until all the messages have finished traveling upwards, and only then start their journey downwards. Therefore, in both protocols the collection and distribution subprotocols are conducted concurrently, either by using separate channels or by multiplexing: The odd time slots are dedicated to the upward traffic (collection) and the even ones to the downward traffic. We shall not elaborate further and assume separate channels.

All our protocols make use of a basic protocol, *Decay* [3], for passing messages from one layer to the next. In the sequel we use the term *send* whenever *Decay* is used.

```
procedure Decay ($m$);
repeat at most $2 \log \Delta$ times
        transmit $m$ to all neighbors;
        flip $coin \in_R \{0, 1\}$
until $coin = 0$.
```

Decay is a probabilistic protocol, with the following properties:

(1) It lasts $2 \log \Delta$ time slots.

(2) If several neighbors of a node v use *Decay* to send messages then with probability greater than $\frac{1}{2}$ the node v receives one of the messages.

Several of our protocols require that all successfully sent messages be acknowledged. We show that, although there is positive probability that a message is not received, every message that has actually been received is acknowledged with certainty. The overhead of the acknowledgment mechanism is minimal—it slows down the protocol by a factor of 2. As a result the point-to-point transmission is always successful on the graph spanned by the BFS tree.

1.5. Organization. Section 2 describes the setup phase. Since it relies on previous work we only show how to modify it for our needs. All the other results are entirely new. Section 3 describes the acknowledgment mechanism, §4 the collection protocol and its analysis, §5 the point-to-point transmission protocol and §6 the distribution protocol. An application, ranking, is described in §7. Concluding remarks appear in §8.

2. The setup phase. Our protocols require the existence of a basic communication subnetwork. This network consists of a leader which is a root of a BFS tree. Bar-Yehuda, Goldreich, and Itai [4] described how to find a leader in $O((\log\log n) \cdot (D + \log(n/\varepsilon)) \log \Delta)$ time.

In [3] Bar-Yehuda, Goldreich, and Itai describe how to find a BFS tree. Their algorithm assumes that all nodes wake up at time 0. It requires $O(D \log \Delta \log(n/\varepsilon))$ time slots and succeeds with probability $1 - \varepsilon$. Since we have assumed that all the IDs are distinct and n is known to all the nodes, the leader election and BFS can be modified so that they always succeed; only the running time is random.

First, choose $\varepsilon = 1/n$, thus with probability $\geq 1 - n^{-1}$ the leader election and the BFS protocol succeed. To ensure that the protocol always succeeds, when joining the tree each node sends a message to the root using the collection protocol of §4. This protocol only uses already constructed edges of the BFS tree, always succeeds, and requires an average of $O(n \log \Delta)$ time slots to send all these messages. If the root does not receive all the messages by twice the expected time, the algorithm is aborted and the entire setup phase is reinvoked. Note that since all nodes know when the invocation should terminate, different invocations by the same processor cannot exist concurrently.

Since the probability of reinvocation is less than $\frac{1}{2}$, the entire modified setup protocol lasts $O((n + D \log n) \log \Delta)$ time slots on the average.

2.1. Preventing collisions from different levels. An advantage of the BFS tree is that a collision at a node v at level i can occur only by messages sent from levels $i - 1$, i, and $i + 1$. Collisions of messages sent from different levels are prevented by using time multiplexing: We require that a mode at level i transmits a message at time slot t only if $t \equiv i \bmod 3$. This increases the duration of our protocols by a factor of three. Henceforth, we assume that this mechanism has been built into all our protocols.

3. The acknowledgment protocol. The protocols of §§4 and 5 use messages which are each destinated to a single processor. These protocols require that every message be acknowledged. We now show how to conduct acknowledgments deterministically. The odd time slots are dedicated to the original protocol and the even ones to acknowledgments. Namely, every node that receives a message sends an acknowledgment on the next time slot.

The next theorem shows the correctness of this protocol. The theorem depends on the fact that each message has a unique destination and that the destinations of different messages successfully received at the same time slot are distinct.

THEOREM 3.1. *Let v be a node that received a message from node u using the above protocol, then u receives an acknowledgment.*

Proof. Suppose that v received the message from u at time slot t and that u did not receive the acknowledgment. According to the protocol, v sent an acknowledgment at time slot $t + 1$. Since u did not receive the acknowledgment there must have been a conflict at u, i.e., at time slot $t + 1$ another node, v', connected to u also sent an acknowledgment (see Fig. 1). According to the protocol, v' would not send an acknowledgment unless it received a message destinated to it at time slot t.

However, since the message sent by u was destinated to $v \neq v'$ and v' acknowledges only messages destinated to it, v' received its message from a node $u' \neq u$. Therefore, at time slot t both u and u' sent messages, and since v' is connected to both of them, a conflict occurred at v' (at time slot t) and v' did not successfully receive any message. This contradicts the assumption that v' successfully received a message at time slot t. □

4. Collection. The purpose of the collection protocol is to send messages from the sources to the root of the BFS tree. Since no source knows the number of IDs of the other sources, this is done concurrently and independently by all of them.

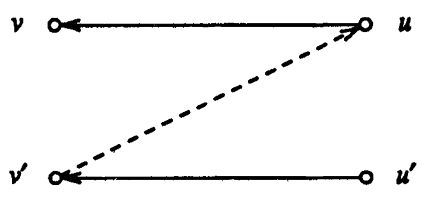


FIG. 1

Messages are sent, using *Decay*, via the BFS tree from BFS children to their parents. To each message we append the ID of the node v which sent the message and the ID of v's BFS parent. This information enables a node to figure out whether the message was sent by its BFS child, by its BFS parent, or by another node. The nodes will make use of this information and we shall omit the details of this.

4.1. The collection protocol. Every node has a buffer of unacknowledged messages. Initially, all buffers except those of the sources are empty. The protocol proceeds in phases. In the odd time slots of each phase every node whose buffer in not empty executes *Decay* to send a message from its buffer to its BFS parent. The even time slots are dedicated to acknowledgments as explained in §3. Every such message is re-sent until an acknowledgment is received. Thereupon it is removed from the sender's buffer. When a message is received it is put on its receiver's buffer. Since the acknowledgment occurs immediately after sending, messages exist in exactly one buffer and proceed from child to parent.

4.2. Analysis of the collection protocol.

4.2.1. Transmission between adjacent levels. We first estimate how fast messages move from level to level.

THEOREM 4.1. *Let $i \geq 1$ be a level containing messages at the beginning of a phase. There is probability $\geq \mu \overset{\text{def}}{=} e^{-1}(1 - e^{-1}) \approx 0.2325$ that during the phase a message from level i is successfully received by its* BFS *parent.*

Before sketching the proof, note the difference between this theorem and property (2) of *Decay*. Suppose node u is sending a message to its neighbor v and node u' to its neighbor v'. If u' is connected also to v and u' to v, then property (2) is satisfied even when v gets the message of u' or v' gets that of u. In contrast, here we insist that each message arrive at its correct destination.

Proof sketch. A phase consists of a single invocation of *Decay*. At any given time, the nodes which still want to transmit are called *live*. At each time slot each live processor first transmits and then with probability $\frac{1}{2}$ *dies*. Therefore, on the average, half of the live nodes die at each time slot.

Let TRY_i be the set of nodes of level i who are live at time $t = 0$. Consider two cases.

Case 1. $|TRY_i| \leq \Delta$: The analysis of *Decay* [3] implies that with probability $\geq \frac{1}{2}$ there exists a time slot with exactly one live processor u of TRY_i. Hence, the BFS parent of u receives u's message.

Case 2. $|TRY_i| > \Delta$: The probability that a node $v \in TRY_i$ is live at time $t_0 \overset{\text{def}}{=} \log \Delta$ is $1/\Delta$. The probability that TRY_i contains a live node at time t_0 is $1 - (1 - (1/\Delta))^{|TRY_i|} \geq 1 - (1 - (1/\Delta))^{\Delta} \geq 1 - e^{-1}$.

Let u be the first such node in some fixed arbitrary order, w its BFS parent, and $S = \{v \in TRY_i | (v, w) \in E\} \backslash \{u\}$ (the transmitting neighbors of w not including u). If at time

t_0, S contains no live nodes, then w receives u's message. The existence of a live node in S is independent of the behavior of u. The probability that at time t_0, S contains no live node is at least

$$\left(1-\frac{1}{\Delta}\right)^{|S|} \geq \left(\left(1-\frac{1}{\Delta}\right)^{\Delta-1}\right)^{|S|/(\Delta-1)}.$$

Since $|S| \leq \Delta - 1$ the above probability is $\geq e^{-1}$. Therefore, the probability of a successful transmission is greater than or equal to

Prob(TRY_i contains a live mode at time t_0) $\times$ Prob (at time t_0 all nodes in S dead)
$\geq (1-e^{-1}) \times e^{-1}$. $\square$

4.2.2. Outline of the analysis. In the remainder of the section we shall show an upper bound on the expected completion time of the collection protocol. We shall go through a series of models and show that when moving from one model to the next the expected completion time can only increase. In this subsection we describe the models; a formal proof follows in subsequent subsections.

The first model is the previously discussed radio network which contains a BFS tree of depth D and k messages arbitrarily placed on the nodes of the tree. In Theorem 4.1 we showed that there is probability $\geq \mu = e^{-1}(1-e^{-1})$ that among all the messages placed in the nodes of level i at least one moves to level $i-1$.

The second model consists of a path of $D+1$ nodes. All the messages in the ith level of the previous model reside in node i of the path. The root of the tree is now node 0. We also stipulate that in a single step at most one message can move from node i to node $i-1$ and that the probability that such a move actually took place is exactly μ.

In the third model the messages are not already present in the system at time = 0, but their arrival is a Bernoulli event with parameter $\lambda < \mu$, i.e., for every time t there is probability λ that a new message appears at node D.

The last model we introduce (model 4) is identical to the third, but we assume that it is already in steady state in the sense of Queueing Theory (see [14]) and we define the expected completion time to be the expected time for k *additional* messages to arrive at node D and then proceed to the root (node 0).

4.2.3. A tandem queue of Bernoulli servers. We now use Queueing Theory to analyze the performance of model 4: The model consists of D servers connected in series, with the output of the ith server being the input to the $i-1$st. We first analyze the behavior of a single level.

A *Bernoulli server with parameter* μ is a discrete server (i.e., it operates in discrete time steps) such that if at any time step the queue of incoming customers is nonempty, then with probability μ during that time step exactly one customer is served (removed from the incoming queue and placed on the outgoing queue). The *arrival rate* λ is the probability that a new customer (i.e., message) appears in the incoming queue during a phase. The *departure process* is the process by which customers are served by the server. Following Burke [5], Hsu and Burke [12] analyzed the behavior of the departure process when $\lambda < \mu$.

THEOREM 4.2 [12]. *Consider the departure process of the above server* (*with* $\lambda < \mu$), *i.e., let* $\delta(t) = 1$ *if at time* t *a customer was processed, and* $\delta(t) = 0$ *otherwise. Then* δ *converges to a Bernoulli process with parameter* λ.

Hsu and Burke also showed that the probability that at time t the length of the queue is j approaches a limit p_j and

$$(*) \qquad p_0 = 1 - \frac{\lambda}{\mu}, \quad p_1 = \frac{\lambda}{(1-\lambda)\mu} p_0, \quad p_j = \left(\frac{\lambda(1-\mu)}{\mu(1-\lambda)} \right)^{j-1} p_1.$$

Thus the expected queue length is $\bar{N} = \sum_{j \geq 0} j p_j = (\lambda(1-\lambda)/\mu - \lambda)$. By Little's result [14], in steady state the average time in the queue is, $E(T) = (\bar{N}/\lambda) = (1-\lambda)/(\mu - \lambda)$.

We now return to model 4, a steady-state network of D Bernoulli processors connected in series each with parameter μ, and the arrival rate to the Dth server is $\lambda < \mu$. The major observation is that since the output of the ith server is the input to the $i-1$st, the input to all servers is Bernoulli with parameter λ. Using this observation we get the following theorem.

THEOREM 4.3. *The expected completion time of model* 4 *is* $(k/\lambda) + (1-\lambda)/(\mu - \lambda)D$.

Proof. Suppose that at time t_0 the queueing system is in steady state and a message m_0 arrives. Let T denote the time the message spends in the queues. Consider the k messages $m_1, \ldots, m_k$ preceding message m_0. Let X_i denote the interarrival time between message m_i and the next message, m_{i-1}. Define

$$Q_k = T + X_1 + X_2 + \cdots + X_k.$$

Q_k is the time for k messages to pass through the queueing system of model 4. The expected time is

$$E(Q_k) = E(T) + E(X_1) + E(X_2) + \cdots + E(X_k).$$

The theorem follows from the fact that $E(X_i) = (1/\lambda)$ and from the previous discussion which showed that

$$E(T) = \frac{1-\lambda}{\mu - \lambda} D. \qquad \square$$

In Theorem 4.15 we shall show the expected completion time of model 1 is less than or equal to that of model 4. Thus the performance of model 4 constitutes an upper bound for the radio network. Substituting $\lambda = 1 - \sqrt{1-\mu}$ satisfies $\lambda < \mu$ and yields that the expected number of phases required for k messages to reach the root is at most $2(1+\sqrt{1-\mu})\mu^{-1}(k+D)$. Since each phase lasts twice the time of *Decay* and $\mu = e^{-1}(1-e^{-1})$, we get the following theorem.

THEOREM 4.4. *The expected number of time slots for k messages to reach the root is bounded by* $32.27(k+D)\log\Delta$.

This constant can be improved using the techniques of [11].

4.2.4. The expected completion time of model 3 is not greater than model 4. The difference between model 3 and 4 is that model 3 stipulates that initially all queues are empty, whereas in model 4 the queue in each node has reached steady state; in particular there is nonnegative probability that the queues are not empty. In this subsection we prove the intuitively clear point that adding messages to the queues can only increase the expected completion time.

Consider partitions of messages between the levels, i.e., $(D+1)$-vectors $\mathbf{a} = (a_1, \ldots, a_{D+1})$ such that $a_i \geq 0$. A *move vector* is a $(D+1)$-vector of nonnegative integers, $\mathbf{m} = (m_1, \ldots, m_{D+1})$. Partition $\mathbf{a}' = Move(\mathbf{a}, \mathbf{m})$ is obtained by moving m_i messages from level i to level $i-1$ (if $m_i > a_i$ then only a_i messages are moved). Formally, the number of messages moved from level i is $\delta_i = \min(a_i, m_i)$, $i = 1, \ldots, D$, $\delta_{D+1} = m_{D+1}$. Therefore, $a'_i = a_i - \delta_i + \delta_{i+1}$.

A *move sequence* is an infinite series $\mathbf{M} = (\mathbf{m}^1, \mathbf{m}^2, \cdots)$ of move vectors. $Move^*(\mathbf{a}, \mathbf{M}, t)$ is the result of making t moves according to $\mathbf{M}$, i.e.,

$$Move^*(\mathbf{a}, \mathbf{M}, 0) = \mathbf{a}$$

$$Move^*(\mathbf{a}, \mathbf{M}, t+1) = Move(Move^*(\mathbf{a}, \mathbf{M}, t), \mathbf{m}^{t+1}).$$

Define a partial order $\leq$ between partitions, such that $\mathbf{a} \leq \mathbf{b}$ if and only if there exists a move sequence $\mathbf{M}$ and an integer t such that $\mathbf{a} = Move^*(\mathbf{b}, \mathbf{M}, t)$. (Also, $\mathbf{a} < \mathbf{b}$ if $\mathbf{a} \leq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$.)

A move vector $\mathbf{m}$ is a *singleton* if exactly one of its components is 1 and all the other components are zero; the singleton whose ith component is 1 is denoted $\mathbf{e}_i$. The following lemma shows that we can simulate any move vector by a move sequence of lexicographically nonincreasing singletons.

LEMMA 4.5. *For every move vector* $\mathbf{m}$ *there is a singleton move sequence* $\mathbf{E}_\mathbf{m}$ *such that for every partition* $\mathbf{a}$, $Move(\mathbf{a}, \mathbf{m}) = Move^*(\mathbf{a}, \mathbf{E}_\mathbf{m}, \sum_{i=1}^{D} m_i)$.

Proof. Let $\mathbf{E}_\mathbf{m} = (\mathbf{e}^1_\mathbf{m}, \mathbf{e}^2_\mathbf{m}, \cdots)$ such that $\mathbf{e}^t{}_\mathbf{m} = \mathbf{e}_j$, where j is the first nonzero component of $\mathbf{m} - \sum_{i=1}^{t-1} \mathbf{e}^{t-1}_\mathbf{m}$. □

COROLLARY 4.6. $\mathbf{a} \leq \mathbf{b}$ *if and only if there exist an integer* t *and a move sequence* $\mathbf{E}$ *consisting only of singletons such that* $\mathbf{a} = Move^*(\mathbf{b}, \mathbf{E}, t)$.

LEMMA 4.7. *If* $\mathbf{a} \leq \mathbf{b}$ *then for any move vector* $\mathbf{m}$, $Move(\mathbf{a}, \mathbf{m}) \leq Move(\mathbf{b}, \mathbf{m})$.

Proof. Lemma 4.5 and Corollary 4.6 imply that it suffices to prove the lemma for $\mathbf{m} = \mathbf{e}_j$ and $\mathbf{a} = Move(\mathbf{b}, \mathbf{e}_i)$. If $i \neq j+1$ then $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_i), \mathbf{e}_j) = Move((\mathbf{b}, \mathbf{e}_j), \mathbf{e}_i)$, implying $Move(\mathbf{a}, \mathbf{e}_j) \leq Move(\mathbf{b}, \mathbf{e}_j)$.

Also, if $b_i = 0$ then $\mathbf{a} = \mathbf{b}$. Thus we assume that $i = j+1$ and $b_i > 0$. If $b_j = 0$ then $\mathbf{b} = Move(\mathbf{b}, \mathbf{e}_j)$, and $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j)$. From the definition of $\leq$, $Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j) \leq \mathbf{b} = Move(\mathbf{b}, \mathbf{e}_j)$. Otherwise ($b_j \neq 0$), $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_j), \mathbf{e}_{j+1}) \leq Move(\mathbf{b}, \mathbf{e}_j)$. □

The *completion time* of a partition $\mathbf{a}$ with respect to a move sequence $\mathbf{M}$ is $T(\mathbf{a}, \mathbf{M}) = \min\{t : Move(\mathbf{a}, \mathbf{M}, t) = (0, 0, \ldots, 0)\}$. (For some $\mathbf{M}$'s the completion time may be infinite.)

LEMMA 4.8. *If* $\mathbf{a} \leq \mathbf{b}$ *then for all* $\mathbf{M}$, $T(\mathbf{a}, \mathbf{M}) \leq T(\mathbf{b}, \mathbf{M})$.

Proof. Let $\mathbf{b}$ be the least partition for which there exists a move sequence $\mathbf{M}$ and a partition $\mathbf{a}$ such that $\mathbf{a} < \mathbf{b}$ while $T(\mathbf{a}, \mathbf{M}) > T(\mathbf{b}, \mathbf{M})$. Let $\mathbf{M}'$ satisfy

$$T(\mathbf{b}, \mathbf{M}') = \text{Min}\{T(\mathbf{b}, \mathbf{M}) : T(\mathbf{b}, \mathbf{M}) < T(\mathbf{b}, \mathbf{M})\}.$$

Let $\mathbf{M}' = (\mathbf{m}^1, \mathbf{m}^2, \cdots)$. The minimality of $\mathbf{M}'$ implies that $Move(\mathbf{b}, \mathbf{m}^1) < \mathbf{b}$. Define $\mathbf{M}'' = (\mathbf{m}^1, \mathbf{m}^2, \cdots)$. By Lemma 4.7 $Move(\mathbf{a}, \mathbf{m}^1) \leq Move(\mathbf{b}, \mathbf{m}^1)$, thus by the minimality of $\mathbf{b}$, $T(Move(\mathbf{a}, \mathbf{m}^1), \mathbf{M}'') \leq T(Move(\mathbf{b}, \mathbf{m}^1), \mathbf{M}'')$. Hence,

$$T(\mathbf{a}, \mathbf{M}') \leq 1 + T(Move(\mathbf{a}, \mathbf{m}^1), \mathbf{M}'') \leq 1 + T(Move(\mathbf{b}, \mathbf{m}^1), \mathbf{M}'') = T(\mathbf{b}\mathbf{M}'). \qquad \square$$

Consider an arbitrary probability distribution on the move sequences. In a tandem queue of Bernoulli servers with parameter μ and arrival rate λ, $P(m^t_i = 1) = \mu, i = 1, \ldots, D$, and $P(m^t{}_{D+1} = 1) = \lambda$.

Let $p_t(\mathbf{a})$ be the probability that $T(\mathbf{a}, \mathbf{M}) = t$. The *average completion time* is $E(T(\mathbf{a})) = \sum_{t=1}^{\infty} t p_t(\mathbf{a}) = \sum_{t=1}^{\infty} \sum_{j=t}^{\infty} p_j(\mathbf{a})$.

LEMMA 4.9. $\mathbf{a} \leq \mathbf{b}$ *implies that* $E(T(\mathbf{a})) \leq E(T(\mathbf{b}))$.

Proof. Let $\mathbf{a} \leq \mathbf{b}$. Thus by Lemma 4.8, $\{\mathbf{M} : T(\mathbf{a}, \mathbf{M}) \leq t\} \supseteq \{\mathbf{M} : T(\mathbf{b}, \mathbf{M}) \leq t\}$. Taking probabilities, $P(\{\mathbf{M} : T(\mathbf{a}, \mathbf{M}) \leq t\}) \geq P(\{\mathbf{M} : T(\mathbf{b}, \mathbf{M}) \leq t\})$. In other words, $\sum_{j=0}^{t} p_j(\mathbf{a}) \geq \sum_{j=0}^{t} p_j(\mathbf{b})$. The last condition defines that $T(\mathbf{a})$ is *stochastically* *greater than* $T(\mathbf{b})$ [15]. In this case, $E(T(\mathbf{a})) = \sum_{t=1}^{\infty} \sum_{j=t}^{\infty} p_j(\mathbf{a}) \leq \sum_{t=1}^{\infty} \sum_{j=t}^{\infty} p_j(\mathbf{b}) = E(T(\mathbf{b}))$. □

LEMMA 4.10. *The expected completion time of model* 3 *is less than or equal to that of model* 4.

Proof. In model 4 the expected completion time depends both on the distribution of the initial partition and on the move sequences.

Let $\mathbf{a} = (a_1, \ldots, a_D, a_{D+1})$ be an initial partition. $a_1, \ldots, a_D$ are the lengths of the queues at the nodes, while $a_{D+1} = k$ since the completion time of model 4 is the time k additional messages appear at node D and reach the root.

In model 3 the initial partition is $\mathbf{k} = (0, 0, \ldots, 0, k)$. $\mathbf{k} \leq \mathbf{a}$ since for $M = ((a_1, \ldots, a_D, 0), (a_2, \ldots, a_D, 0, 0), \ldots, (a_D, 0, \ldots, 0), (0, \ldots, 0), \ldots)$, $\mathbf{k} = Move^*(\mathbf{a}, M, D)$. By Lemma 4.9, $E(T(\mathbf{k})) \leq E(T(\mathbf{a}))$. Our result follows since this holds for every initial partition of model 4. □

4.2.5. The expected completion time of model 2 is not greater than model 3.

LEMMA 4.11. *The expected completion time of model* 2 *is less than or equal to that of model* 3.

Proof. In model 3 the initial partition is $\mathbf{k} = (0, 0, \ldots, 0, k)$, while in model 2 it is $\mathbf{b} = (b_1, \ldots, b_D, 0)$, such that $k = \sum b_i$. Obviously, $\mathbf{b} \leq \mathbf{k}$, so the result follows as before from Lemma 4.9. □

4.2.6. The expected completion time of model 2 is not smaller than model 1. The difference between models 1 and 2 is in the move vectors. In model 2, $m_i^t \in \{0, 1\}$ and $P(m_i^t = 0) = 1 - \mu$, while in model 1, m_i^t can assume any nonnegative integer value and $P(m_i^t = 0) = 1 - \mu$. The actual movements of the messages depend on the topology. To prove that the expected completion time of model 2 is not greater than that of model 1 we need to consider the effect of changing the move vectors.

A move vector $\mathbf{m} = (m_1, m_2, \ldots, m_{D+1})$ *dominates* the move vector $\tilde{\mathbf{m}} = (\tilde{m}_1, \tilde{m}_2, \ldots, \tilde{m}_{D+1})$ if for all i, $m_i \geq \tilde{m}_i$.

LEMMA 4.12. *If* $\mathbf{m}$ *dominates* $\tilde{\mathbf{m}}$ *and* $\mathbf{a} \leq \mathbf{b}$ *then* $Move(\mathbf{a}, \mathbf{m}) \leq Move(\mathbf{b}, \tilde{\mathbf{m}})$.

Proof. Let $\mathbf{E_m}$ and $\mathbf{E_{\tilde{m}}}$ be the singleton move sequences corresponding to $\mathbf{m}$ and $\tilde{\mathbf{m}}$ (Lemma 4.5). Let t and $\tilde{t}$ be the respective lengths. Since $\mathbf{E_{\tilde{m}}}$ is a subsequence of $\mathbf{E_m}$, by repeated use of Lemma 4.7 we can show that $Move^*(\mathbf{a}, \mathbf{E_m}, t) \leq Move^*(\mathbf{a}, \mathbf{E_{\tilde{m}}}, \tilde{t})$. Thus, by Lemma 4.5,

$$Move(\mathbf{a}, \mathbf{m}) = Move^*(\mathbf{a}, \mathbf{E_m}, t) \leq Move^*(\mathbf{a}, \mathbf{E_{\tilde{m}}}, \tilde{t}).$$

By Lemma 4.7,

$$\leq Move^*(\mathbf{b}, \mathbf{E_{\tilde{m}}}, \tilde{t}) = Move(\mathbf{b}, \tilde{\mathbf{m}}). \quad \square$$

A move sequence $\mathbf{M}(\mathbf{m}^1, \mathbf{m}^2, \cdots)$ *dominates* $\tilde{\mathbf{M}}(\tilde{\mathbf{m}}^1, \tilde{\mathbf{m}}^2, \cdots)$ if for all j, $\mathbf{m}^j$ dominates $\tilde{\mathbf{m}}^j$.

LEMMA 4.13. *If* $\mathbf{M}$ *dominates* $\tilde{\mathbf{M}}$ *and* $\mathbf{a} \leq \mathbf{b}$ *then* $T(\mathbf{a}, \mathbf{M}) \leq T(\mathbf{b}, \tilde{\mathbf{M}})$.

Proof. The proof is by induction on $T(\mathbf{a}, \mathbf{M})$ and using Lemma 4.12. □

LEMMA 4.14. *The expected completion time of model* 1 *is less than or equal to that of model* 2.

Proof. Consider an instance $\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^T$ of model 1, i.e., the completion time is T and a_i^t is the number of messages at level i at time $t \leq T$. Define the move sequence $\mathbf{M} = (\mathbf{m}^1, \mathbf{m}^2, \cdots)$ as follows: m_i^t is the number of messages that moved from level i to level $i - 1$ at time t. When $a_i^t > 0$ then by Theorem 4.1, $P(m_i^t \geq 1) \geq \mu$. However, when $a_i^t = 0$ then $m_i^t = 0$, so obviously, $P(m_i^t \geq 1) = 0$, violating the probabilistic assumptions of model

2. We therefore define the move sequence $\bar{\mathbf{M}} = (\bar{\mathbf{m}}^1, \bar{\mathbf{m}}^2, \cdots)$: if $a_i^t > 0$ then $\bar{m}_i^t = m_i^t$, otherwise $(a_i^t = 0)$, $\bar{m}_i^t = 1$. By Theorem 4.1 and the construction $P(\bar{m}_i^t \geq 1) \geq \mu$. Since $\mathbf{M}$ and $\bar{\mathbf{M}}$ differ only when $a_i^t = 0$, for every $t \leq T$, $Move^*(\mathbf{a}, \mathbf{M}, t) = Move^*(\mathbf{a}, \bar{\mathbf{M}}, t)$. Thus the completion times $T(\mathbf{a}, \bar{\mathbf{M}}) = T(\mathbf{a}, \mathbf{M}) = T$.

Construct a move sequence $\tilde{\mathbf{M}} = (\tilde{\mathbf{m}}^t, \ldots, \tilde{\mathbf{m}}^T, \cdots)$: $(\tilde{m}_i^t \in \{0, 1\})$. If $\bar{m}_i^t \geq 1$ then $\tilde{m}_i^t = 1$ with probability $((P(m_i^t \geq 1) - \mu)/P(m_i^t \geq 1))$; otherwise $\tilde{m}_i^t = 0$. Since $\bar{\mathbf{M}}$ dominates $\tilde{\mathbf{M}}$, by Lemma 4.13, the completion time of $\bar{\mathbf{M}}$ is less than or equal to that of $\tilde{\mathbf{M}}$.

By the construction of $\tilde{\mathbf{M}}$,

$$\begin{aligned} P(\tilde{m}_i^t = 0) &= P(\bar{m}_i^t = 0) + P(\bar{m}_i^t > 0 \text{ and } \tilde{m}_i^t = 0) \\ &= P(\bar{m}_i^t = 0) + P(\bar{m}_i^t > 0)\frac{P(\bar{m}_i^t > 0) - \mu}{P(\bar{m}_i^t > 0)} \\ &= (1 - P(\bar{m}_i^t > 0)) + (P(\bar{m}_i^t > 0) - \mu) = 1 - \mu. \end{aligned}$$

Thus, the distribution of $\tilde{M}$ is identical to the distribution of the move sequences of model 2. The required result follows by taking expectations. □

We summarize the above reductions with the following theorem.

THEOREM 4.15. *The expected completion time of model* 1 *is less than or equal to that of model* 4.

5. Point-to-point transmission. As mentioned before, this protocol consists of two subprotocols: The upward direction subprotocol from the initiator of the message u to a common ancestor w of u and the destination v and the downward direction subprotocol from w to v. However, in order to conduct these protocols the network should first execute a preparation protocol. This protocol is executed only once.

5.1. Preparation. This protocol is executed during the set-up phase (§2) before starting any point-to-point transmission. In §2 we described how the BFS tree is constructed; here we describe how additional information the BFS parent, and the BFS descendants (and on which subtree each descendant belongs), are conveyed to each node.

The BFS protocol of §2 enables each node to know the ID of its BFS parent and its depth (distance from the root). The descendant information can be found once the BFS tree is constructed: As soon as a node joins the BFS tree, it sends its ID to the root via its parent. During the BFS protocol, each node can record its parent; thus, to send a message to the root it is sufficient to send it to that parent and ask it to send the message further. Whenever a node receives such a message from one of its children, it adds the ID of the originator of the message to the list of IDs of descendants. Conveying all the descendant information requires the collection of $n - 1$ messages, i.e., $O((n + 1 + D) \log \Delta) = O(n \log n)$ time. To record all this information each node must have sufficient storage to keep $O(n)$ IDs.

To save space (and time) we propose the following scheme [13]: After the BFS tree is completed, a depth first search (DFS) is conducted on the BFS tree. Henceforth, each node uses its DFS number as its address. Since the DFS numbers of the descendants of a node constitute a consecutive range, it suffices that each node remember the DFS number of each of its children and the maximum DFS number of all the descendants. Thus, each node v needs only $O(deg(v) \log n)$ bits of local memory.

Using a token, DFS can be implemented in $O(n)$ time: First the token conducts a DFS of the graph, each node sends the token to the largest neighbor not yet in the DFS tree, and when all the neighbors are exhausted it is sent to the parent. Whenever a node sends the token, it broadcasts its own ID together with the ID of its BFS parent. Thus all the neighbors of a node know when the node joins the DFS tree, and the token is not sent to nodes already in the tree (except when the DFS backtracks from a child to its parent). There are no conflicts, since only the node holding the token can transmit; also the entire traversal requires $2n - 2$ time, since the token traverses once in each direction of each tree edge.

After the first DFS is completed, each node knows the parents of all its neighbors; in particular it knows which of its neighbors are its BFS children. Thus we can conduct a second DFS traversal this time on the BFS tree. This traversal also costs $O(n)$ and after it is completed each node knows the DFS number of all its BFS children and its maximum descendant. Note that we required that each node knows the IDs of all its neighbors only in order to conduct the first DFS traversal.

5.2. The upward subprotocol. This protocol is essentially identical to the collection protocol, except that messages do not go all the way to the root but only to the least common ancestor of the originator and the destination which are included in the message. When the message reaches a BFS tree ancestor of the destination, the downward protocol is invoked.

5.3. The downward subprotocol. This protocol is also very similar to the collection protocol. The messages are prepended with the ID of their final destination. Every node sends messages destinated to its BFS children and keeps a buffer of unacknowledged downgoing messages. Here we also use *Decay*. On each phase a message is sent, according to that protocol. Messages are resent until an acknowledgment is received. A node w receiving a message destinated to u processes it only if u is a BFS-tree descendant of w. To process a message, w acknowledges it, and if $w \neq u$ it is put on w's downgoing buffer.

5.4. Performance analysis. The setup phase requires $O(n + D \log \Delta)$ time after which passing k messages requires an average of $O((k + D) \log \Delta$ time. When $k \to \infty$ the average time per message is $O(\log \Delta)$.

6. Broadcast. To broadcast a message a node first sends the message to the root using the collection subprotocol of §4. Then the message is sent to all the nodes of the network using the distribution subprotocol to be described.

In the distribution protocol every message has several destinations, therefore, the acknowledgment mechanism of §3 can no longer be used. In principle the message can be sent using the BFS protocol. However, each message would require $2D \log \Delta \log n$ time to reach all the nodes with probability $1 - \varepsilon$. A better idea is to use pipelining: Send the $i + 1$st message before the ith one reaches its destination.

The protocol consists of *superphases* each consisting of $4 \log \Delta \log n$ time slots (we allow an error of $\varepsilon = 1/n^2$). At superphase t the root sends the tth message and all the nodes of level i repeatedly send the t–ith message (using the *Decay* protocol $2 \log n$ times).

Let v be a node of level i and t a superphase in which the nodes of level $i - 1$ send the message m. By property (2) of *Decay*, in each invocation of *Decay*, there is probability $\geq \frac{1}{2}$ that v receives a message. Since there can be no interference by messages sent from different levels (§2.2), if v receives any message it must be from level $i - 1$ and since all the nodes of that level send the same message, with probability $\geq \frac{1}{2}$ v receives the message m. Since a superphase consists of $2 \log n$ invocations of *Decay* there is probability $\geq 1 - 1/n^2$ that v receives m during the superphase. The probability that m is passed successfully to all the nodes of the network is $\geq 1 - 1/n$.

For each message the above protocol may fail with finite probability. If the number of messages is unbounded then eventually the protocol will fail. This failure can be prevented by changing the protocol as follows:

> The root appends consecutive numbers to the messages. Every node v examines these numbers and when v encounters a gap it realizes that it did

not receive a message. Thereupon, v sends a message to the root requesting it to resend the missing message.

Since, on the average, no more than $1/n$ of the messages will be resent, the extra load on the network is a factor $\sum_{i=0}^{\infty}(1/n^i) = n/(n-1)$. Moreover, the time spent in each layer is $O(\log\Delta\log n)$; thus the effective rate in which messages leave the root is $O((n/(n-1))\log\Delta\log n) = O(\log\Delta\log n)$. Also, each message requires an average of $O(D\log\Delta\log n)$ time slots to reach all the nodes.

The previous change causes another problem: The message numbers are unbounded. An additional change can correct this problem. The messages are numbered mod $3n^2$. After message number n^2 is received each node sends an acknowledgment to the root. The expected time that all these messages reach the root is $O((n+D)\log\Delta)$. Thus the expected time that all the acknowledgments reach the root is $O((D\log n + n)\log\Delta)$. Let c be the implied constant in the above expression. If the root does not receive acknowledgments from all the nodes by $2c((D\log n + n)\log\Delta)$ time slots after it sent the n^2th message it resends the last n^2 messages. It can be shown that the probability that the n^2th message has to be resent is less than $\frac{1}{2}$, thus this last correction increases the load of the system by at most a factor of 2.

7. Ranking. Our protocols can be used for additional problems, such as ranking in expected time $O(n\log n\log\Delta)$:

The problem:

Given n processors with distinct IDs $id_1, \ldots, id_n$, renumber the processors, $id'_1, \ldots, id'_n$ such that $1 \le id'_i \le n$ and $id'_i < id'_j$ if and only if $id_i < id_j$.

The protocol:

Use point-to-point communication to send all the IDs to the root. It calculates the destination of each of the new IDs and sends them to the nodes.

There is a total of $2n-2$ messages, which require $O(n\log\Delta)$ time (not including the setup costs of §2).

8. Remarks and open problems. (1) If n is not known but only an upper bound N, we can still find a BFS tree with probability $1-\varepsilon$ in expected time $O(D\log(N/\varepsilon)\log\Delta)$. This setup time is sufficient for k-broadcast. However, point-to-point transmissions still require $O(n + D\log(N/\varepsilon)\log\Delta)$ time to acquire the descendant information.

(2) If there are no IDs then the processors can randomly choose sufficiently long IDs such that with probability $1-\varepsilon$ all the IDs are distinct.

(3) Suppose that we change the model such that in case of a conflict the receiver may get one of the messages. In this model our deterministic acknowledgment mechanism is no longer valid. A more complicated, less reliable and slower protocol also exists for this case.

(4) In some "real life" situations processors can detect that a conflict occurred. We have not postulated this ability since we do not know how to use it.

(5) Our protocols route messages through a spanning tree causing congestion at the root. Are there efficient communication protocols that avoid this problem?

Acknowledgments. It is a pleasure to thank Shay Kutten and Moshe Sidi for helpful discussions and an anonymous referee for suggesting using DFS numbers to improve the memory requirements.

REFERENCES

[1] N. ALON, A. BAR-NOY, N. LINIAL, AND D. PELEG, *A lower bound for radio broadcast*, J. Comput. System Sci., 43 (1991), pp. 188–210; also in STOC 1989.

[2] Y. Birk, *Concurrent communication among multi-transceiver stations over shared media*, Ph.D. thesis, Tech. Report CSL-TR-87-321, Stanford University, Stanford, CA.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time-complexity of broadcast in radio networks: An exponential gap between determinism and randomization*, J. Comput. System Sci., 45 (1992), pp. 104–126; also in PODC 1987.

[4] ———, *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distributed Computing, 5 (1991), pp. 67–71.

[5] P. J. Burke, *The output of a queuing system*, Oper. Res., 4 (1956), pp. 699–704.

[6] J. Capetanakis, *Generalized* TDMA: *The multi-accessing tree protocol*, IEEE Trans. Comm., COM-27 (1979), pp. 1479–1484.

[7] I. Chlamtac and S. Kutten, *On broadcasting in radio networks–problem analysis and protocol design*, IEEE Trans. Comm., COM-33 (1985).

[8] L. Chlamtac and O. Weinstein, *The wave expansion approach to broadcasting in multi-hop radio networks*, INFOROM (1987), pp. 874–881.

[9] Digital–Intel–Xerox, *The Ethernet data link layer and physical layer specification 1.0*, Sept., 1980.

[10] R. Gallager, *A perspective on multiaccess channels*, IEEE Trans. Inform. Theory, IT-31 (1985), pp. 124–142.

[11] M. Hofri, *A feedback-less distributed broadcast algorithm for multihop radio networks with time-varying structure*, 2nd ACM Intr. MCPR Workshop, Rome, May, 1987. Also available as TR-451, Computer Science Dept., Technion, Haifa, Israel, March, 1987.

[12] Hsu and P. J. Burke, *Behavior of tandem buffers with geometric input and Markovian output*, IEEE Trans. Comm., COM-24 (1976), pp. 359–361.

[13] A. Itai and M. Rodeh, *The multi-tree approach to reliability in distributed systems*, Proc. 25 Symposium on Foundations of Computer Science, Oct. 1984, pp. 137–147. Also in Inform. Comput., 79 (1988), pp. 43–59.

[14] L. Kleinrock, *Queueing Systems, Volume* 1: *Theory*, John Wiley & Sons, New York, 1975.

[15] D. Stoyan, *Comparison Method for Queues and Other Stochastic Models*, John Wiley & Sons, Inc., New York, 1983.

[16] O. Weinstein, *The wave expansion approach to broadcasting in multihop radio networks*, M. Sc. thesis, Computer Science Dept., Technion, Haifa, Israel, 1987.

[17] D. E. Willard, *Log-logarithmic selection resolution protocols in a multiple access channel*, SIAM J. Comput., 15 (1986), pp. 468–477.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 889–912, October 1993

FINDING A SMALLEST AUGMENTATION TO BICONNECT A GRAPH*

TSAN-SHENG HSU† AND VIJAYA RAMACHANDRAN†

Abstract. The problem of finding a minimum number of edges whose addition biconnects an undirected graph is considered. This problem has been studied by several other researchers, two of whom presented a linear-time algorithm for this problem in an earlier volume of this journal. However, that algorithm contains an error that is exposed in this paper. A corrected linear-time algorithm for this problem, as well as a new efficient parallel algorithm, are presented. The parallel algorithm runs in $O(\log^2 n)$ time with a linear number of processors on an EREW PRAM, where n is the number of vertices in the input graph.

Key words. algorithm, linear time, graph augmentation, biconnected graph, parallel computation, poly-log time, EREW PRAM

AMS subject classifications. 68Q20, 68R10, 94C15, 05C40

1. Introduction.

The problem of augmenting a graph to reach a certain connectivity requirement by adding edges has important applications in network reliability [6], [12], [21] and in fault-tolerant computing. One version of the augmentation problem is to augment the input graph to reach a given connectivity requirement by adding a smallest set of edges. We refer to this problem as the *smallest augmentation* problem.

The following results on solving the smallest augmentation problem on an undirected graph are known to satisfy a vertex connectivity requirement. Eswaran and Tarjan [4] gave a lower bound on the smallest number of edges for biconnectivity augmentation and proved that the lower bound can be achieved. Rosenthal and Goldner [18] developed a linear-time sequential algorithm for finding a smallest augmentation to biconnect a graph. Watanabe and Nakamura [26], [28] gave an $O(n(n+m)^2)$-time sequential algorithm for finding a smallest augmentation to triconnect a graph with n vertices and m edges. Hsu and Ramachandran [11] developed a linear-time algorithm for this problem. There is no polynomial-time algorithm known for finding a smallest augmentation to k-vertex-connect a general graph for $k > 3$. There is also no efficient parallel algorithm known to find a smallest augmentation to k-vertex-connect a graph for $k \geq 2$.

For the problem of finding a smallest augmentation for a graph to reach a given edge connectivity property, several polynomial-time algorithms on undirected graphs, directed graphs, and mixed graphs are known. These results can be found in Cai and Sun [1], Eswaran and Tarjan [4], Frank [5], Gusfield [8], Kajitani and Ueno [13], Naor, Gusfield, and Martel [15], Ueno, Kajitani, and Wada [24], Watanabe [25], and Watanabe and Nakamura [27]. Efficient parallel algorithms for finding smallest augmentations for 2-edge connectivity, strong connectivity, and making a mixed graph strongly orientable can be found in Soroker [20].

Another version of the problem is to augment a graph, with a weight assigned to each edge, to meet a connectivity requirement by using a set of edges with a minimum total cost. Several related problems have been proved to be **NP**-complete. These results can be found in Eswaran and Tarjan [4], Frank [5], Frederickson and Ja'Ja' [7], Watanabe and Nakamura [26], and Watanabe, Narita, and Nakamura [29].

In this paper we present an efficient parallel algorithm for finding a smallest augmentation to biconnect an undirected graph. In addition, we have discovered an error in the sequential

*Received by the editors March 4, 1991; accepted for publication (in revised form) April 15, 1992. This work was supported in part by National Science Foundation grant CCR-89-10707. A preliminary version of this paper appears in Proceedings of the Second Annual International Symposium on Algorithms, Lecture Notes in Computer Science 557, Springer-Verlag, Berlin, 1991, pp. 326–335.

†Department of Computer Sciences, University of Texas, Austin, Texas 78712.

algorithm of Rosenthal and Goldner [18]. We first give a corrected linear-time sequential algorithm for the problem. Our efficient parallel algorithm is based on this corrected sequential algorithm. However, we have to use several insights into the problem in order to derive the parallel algorithm. The algorithm runs in $O(\log^2 n)$ time with a linear number of processors on an EREW PRAM, where n is the number of vertices in the input graph. (For more on PRAM models and PRAM algorithms see Karp and Ramachandran [14].)

The algorithmic notation used is from Tarjan [22] and Ramachandran [17]. We enclose comments between "{*" and "*}." We use the following **pfor** statement for executing a loop in parallel.

```
pfor iterator → statement list rofp
```

The effect of this statement is to perform the statement list in parallel for each value of the iterator. We use the following form for an **if** statement.

```
if condition$_1$ → statement list$_1$
| condition$_2$ → statement list$_2$
        ⋮
| condition$_n$ → statement list$_n$
fi
```

The effect of this statement is to perform the first statement list whose corresponding condition is true. If no condition is true, none of the statement lists is evaluated. Parameters are called by value unless they are declared with the key word **modifies**, in which case they are called by value and result.

2. Definitions.

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. Let $\{E_i \mid 1 \leq i \leq k\}$ be a partition of E into a set of k disjoint subsets such that two edges e_1 and e_2 are in the same partition if and only if there is a simple cycle in G containing e_1 and e_2 or e_1 is equal to e_2. A vertex is called an *isolated* vertex if it is not adjacent to any other vertex. Let q be the number of isolated vertices in G. Let $\{V_i \mid 1 \leq i \leq k+q\}$ be a collection of sets of vertices, where V_i is the set of vertices in E_i for each i, $1 \leq i \leq k$, and V_{i+k} contains only the ith isolated vertex for each i, $1 \leq i \leq q$. A vertex v is a *cutpoint* of a graph G if v appears in more than one vertex set V_i. G is *biconnected* if it has at least three vertices and contains no cutpoint or isolated vertex. The subgraph $G_i = (V_i, E_i)$ $\forall i$, $1 \leq i \leq k$, is a *biconnected component* of G if V_i contains more than two vertices. Note that $E_i = \emptyset$ $\forall i$, $k < i \leq k+q$, since V_i contains an isolated vertex. The subgraph $G_i = (V_i, E_i)$ $\forall i$, $1 \leq i \leq k+q$, is called a *block* of G. Given an undirected graph G, we can define its *block graph* $\text{blk}(G)$ as follows. Each block and each cutpoint of G is represented by a vertex of $\text{blk}(G)$. The vertices of $\text{blk}(G)$ that represent blocks are called *b-vertices* and those representing cutpoints are called *c-vertices*. Two vertices u and v of $\text{blk}(G)$ are adjacent if and only if u is a c-vertex, v is a b-vertex, and the corresponding cutpoint of u is contained in the corresponding block of v or vice versa. It is well known that $\text{blk}(G)$ is a forest and that if G is connected, $\text{blk}(G)$ is a tree. If $\text{blk}(G)$ is a tree, it is also called a *block tree*.

Let n_c be the number of c-vertices in $\text{blk}(G)$. A vertex v_i represents a c-vertex of $\text{blk}(G)$ and d_i is the degree of v_i. We assume that $d_i \geq d_{i+1}$ $\forall i$, $1 \leq i < n_c$, throughout the discussion. For convenience we define $a_i = d_i - 1$. If $\text{blk}(G)$ is a tree, let T be the rooted tree obtained from $\text{blk}(G)$ by rooting $\text{blk}(G)$ at the b-vertex that connects to v_1 and is on the path from v_1 to v_2. We use T_i to represent the subtree of T rooted at v_i for each i, $1 \leq i \leq n_c$, and we use T' to represent the subtree of T after deleting T_1. Let l_i be the number of leaves of T_i

$\forall i$, $1 \le i \le n_c$. We also use T_v to represent the subtree rooted at a vertex v of blk(G). The subgraph of T induced by deleting the vertex v is denoted by $T - v$.

In a forest a vertex with degree 1 is a *leaf*. Let l be the number of leaves in blk(G). For a graph G' we use l' to denote the number of degree-1 vertices in blk(G'). Let $d(v)$ be the degree of the vertex v in blk(G), and let d be the largest degree of all c-vertices in blk(G).

In figures we use a rectangle to represent a b-vertex and a circle to represent a c-vertex. A line denotes an edge. A path in the block graph is represented by a thick dashed line, and a collection of subtrees is represented by a polygon. These notations are shown in Fig. 1.

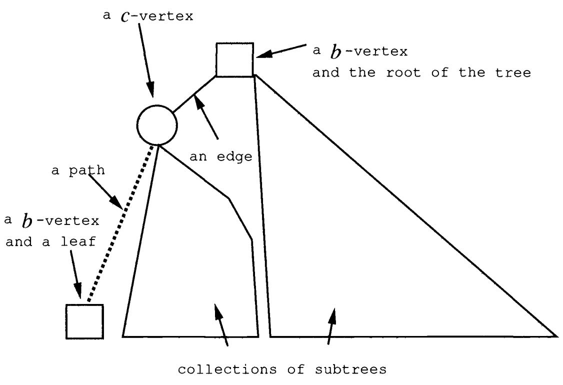


FIG. 1. *Notations for figures.*

We also need the following definitions. Part of Definition 2.4 is from [18].

DEFINITION 2.1. A vertex v of blk(G) is called *massive* if and only if v is a c-vertex with $d(v) - 1 > \lceil l/2 \rceil$. A vertex v of blk(G) is *critical* if and only if v is a c-vertex with $d(v) - 1 = \lceil l/2 \rceil$. The graph blk($G$) is *critical* if and only if there exists a critical c-vertex in blk(G).

DEFINITION 2.2. A block graph blk(G) is *balanced* if and only if G is connected and without any massive c-vertex. (Note that blk(G) could have a critical c-vertex.) A graph G is balanced if and only if blk(G) is balanced.

DEFINITION 2.3. (leaf-connecting condition). Two leaves u_1 and u_2 of blk(G) satisfy the leaf-connecting condition if and only if u_1 and u_2 are in the same tree of blk(G) and the path P from u_1 to u_2 in blk(G) contains either (1) two vertices of degree more than 2 or (2) one b-vertex of degree more than 3.

DEFINITION 2.4. Let v be a c-vertex of blk(G). We call those components of blk(G) $- v$ that contain only one vertex of degree 1 in blk(G) *v-chains* [18]. A degree-1 vertex of blk(G) in a v-chain is called a *v-chain leaf.*

3. Main lemmas.

In this section we present results that will be crucial in the development of our efficient parallel algorithm.

LEMMA 3.1. *If* blk(G) *has more than two c-vertices, then* $a_1 + a_2 + a_3 - 1 \le l$.

Proof. Note that v_1 is a c-vertex with the largest degree. Vertex v_2 is a c-vertex with the largest degree among all c-vertices other than v_1. Vertex v_3 is a c-vertex with the largest degree among all c-vertices other than v_1 and v_2. Recall that if blk(G) is a tree, we root blk(G) at the b-vertex b that connects to v_1 and is on the path from v_1 to v_2. Let the rooted tree be T. Recall that T_i is the subtree of T rooted at v_i and l_i is the number of leaves in T_i. T' is the subtree obtained from T by removing T_1. Let l_x be the number of leaves in T'.

Case 1. If v_3 is in T_1, then $l_1 \geq a_1 - 1 + a_3$ and $l_x \geq a_2$. This implies $l = l_1 + l_x \geq a_1 + a_2 + a_3 - 1$.

Case 2. If v_3 is in T' but not in T_2, then $l_1 \geq a_1$ and $l_x \geq a_2 + a_3$. Thus $l = l_1 + l_x \geq a_1 + a_2 + a_3$.

Case 3. If v_3 is in T_2, then $l_1 \geq a_1$ and $l_x \geq l_2 \geq a_2 - 1 + a_3$. This implies $l = l_1 + l_x \geq a_1 + a_2 + a_3 - 1$.

Suppose that blk(G) is a forest and v_1, v_2, and v_3 are in different trees T_1, T_2, and T_3, respectively. If v_i is the only c-vertex in T_i, then $a_i = l_i - 1$. Otherwise, $a_i \leq l_i$. Thus $a_1 + a_2 + a_3 \leq l_1 + l_2 + l_3 \leq l$. It is easy to prove the lemma for the case in which blk(G) is a forest and any two of v_1, v_2, and v_3 are in the same tree. □

COROLLARY 3.2. *If* blk(G) *has more than two c-vertices, then* $a_3 \leq \frac{l+1}{3}$.

Proof. From the definition we know that $a_1 \geq a_2 \geq a_3$. If $a_3 > \frac{l+1}{3}$, then $a_1 \geq a_2 \geq a_3 > \frac{l+1}{3}$, which implies $a_1 + a_2 + a_3 > \frac{l+1}{3} * 3 = l + 1$. This is a contradiction to Lemma 3.1. □

COROLLARY 3.3. *There can be at most one massive vertex in* blk(G).

Proof. The corollary is obviously true if there are fewer than two c-vertices in blk(G). If blk(G) has only two c-vertices, v_1 and v_2, there is a b-vertex b^* in blk(G) that connects to both v_1 and v_2. We root blk(G) at b^*. Since there are only two c-vertices, the children of v_1 and v_2 are all leaves. We know that a_1 and a_2 are equal to the number of children of v_1 and v_2, respectively; thus $a_1 + a_2 = l$. Suppose v_1 is massive; then $a_1 > \frac{l}{2}$. Thus $a_2 < \frac{l}{2}$. If blk(G) has more than two c-vertices and v_1 and v_2 are massive, then $a_1 + a_2 > l$. Since $a_3 \geq 1$, we have derived a contradiction to Lemma 3.1. □

COROLLARY 3.4. *If there is a massive vertex in* blk(G), *then there is no critical vertex in* blk(G).

Proof. The proof of Corollary 3.3 also applies here. □

COROLLARY 3.5. *There can be at most two critical vertices in* blk(G) *if* $l > 2$.

Proof. The corollary is obviously true if $blk(G)$ has only one or two c-vertices. Assume that $blk(G)$ has more than two c-vertices. From Corollary 3.2, we know that $a_3 \leq \frac{l+1}{3}$. Since $\lceil \frac{l}{2} \rceil \geq \frac{l}{2} > \frac{l+1}{3}$ if $l > 2$, we know that v_3 cannot be critical if $l > 2$. □

Before introducing the next lemma, we have to study properties for updating the block tree. The following fact for obtaining blk(G') from blk(G) is given in Rosenthal and Goldner [18].

FACT 3.6. *Given a graph G and its block tree* blk(G), *adding an edge between two leaves u and v of* blk(G) *creates a cycle C. Let G′ be the graph obtained by adding an edge between u′ and v′ in G, where u′ and v′ are non-cutpoint vertices in the blocks represented by u and v, respectively. The following relations hold between* blk(G) *and* blk(G').

(1) *Vertices and edges of* blk(G) *that are not in the cycle C remain the same in* blk(G').

(2) *All b-vertices in* blk(G) *that are in the cycle C contract to a single b-vertex b′ in* blk(G').

(3) *Any c-vertex in C with degree equal to* 2 *is eliminated.*

(4) *A c-vertex x in C with degree greater than* 2 *remains in* blk(G') *with edges incident on vertices not in the cycle. The vertex x also attaches to the b-vertex b′ in* blk(G').

An example of forming blk(G') from blk(G) is illustrated in Fig. 2.

LEMMA 3.7. *Let* u_1 *and* u_2 *be two leaves of* blk(G) *satisfying the leaf-connecting condition* (*Definition* 2.3). *Let* α *and* β *be noncutpoint vertices in blocks of G represented by* u_1 *and* u_2, *respectively. Let G′ be the graph obtained from G by adding an edge between* α *and* β, *and let P represent the path between* u_1 *and* u_2 *in* blk(G). *The following three conditions are true.*

(1) $l' = l - 2$.

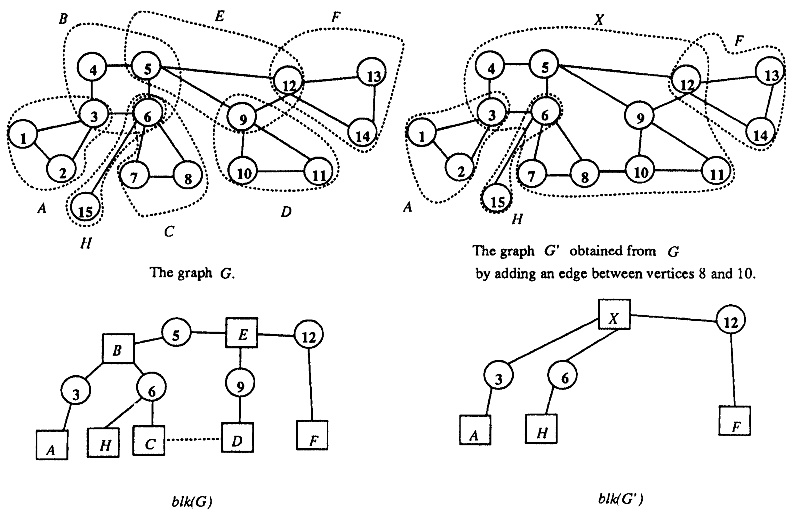

The graph G.
The graph G' obtained from G by adding an edge between vertices 8 and 10.
blk(G)
blk(G')

FIG. 2. *An example of obtaining* blk(G') *from* blk(G). *Vertices of G and G' circled with a dotted line are in the same block. For example, vertices* 1, 2, *and* 3 *of G are in block A. A vertex that appears in more than one block is a cutpoint. For example, vertex* 3 *appears in block A and B, and thus it is a cutpoint. Vertices B, C, D, and E in* blk(G) *are in a cycle if we add an edge between C and D. The cycle contracts into a new b-vertex X in* blk(G'). *The degree of a c-vertex in the cycle decreases by* 1 *in* blk(G') *if the original degree is more than* 2. *A degree-*2 *c-vertex in the cycle is eliminated in* blk(G').

(2) *If v is a cutpoint in P with degree greater than* 2 *in* blk(G), *then the degree of v decreases by* 1 *in* blk(G').

(3) *If v is a cutpoint in P with degree equal to* 2, *then v is eliminated in* blk(G').

Proof. Parts (2) and (3) of the lemma follow from parts (3) and (4) of Fact 3.6. We now prove part (1) of the lemma.

From part (2) of Fact 3.6 we know that every vertex of G that is in a component represented by a b-vertex in P is in a biconnected component Q of G'. Let Q be represented by a b-vertex b in blk(G').

Case 1. Suppose that part (1) of the leaf-connecting condition (Definition 2.3) holds. Let w and y be two vertices of blk(G) having degree more than 2 in blk(G), and let blk(G') be rooted at b. In blk(G) let w' be a vertex adjacent to w and let y' be a vertex adjacent to y, with neither w' nor y' in P. The vertex b has at least two children, w' and y', in blk(G') and hence cannot be a leaf. Since leaves u_1 and u_2 are eliminated in blk(G') and no new leaf is created, $l' = l - 2$.

Case 2. Suppose that part (2) of the leaf-connecting condition (Definition 2.3) holds. Let w be a b-vertex of degree more than 3. We can find at least two c-vertices, y' and z', that are connected to w but that are not in P. The same reasoning used in case 1 can be followed to prove this case. □

4. The algorithm. The original linear-time sequential algorithm in Rosenthal and Goldner [18] consists of three stages. However, we have discovered an error in stage 3 of the algorithm in [18]. We present a corrected version of that stage of the algorithm. Our parallel algorithm follows the structure of the corrected sequential algorithm. The first two stages are easy to parallelize, and we describe them in §§4.1 and 4.2. However, stage 3 is highly sequential. Most of our discussion is on a corrected algorithm for stage 3 and its parallelization (§4.3).

We first state a lower bound on the number of edges needed to augment a graph to reach biconnectivity.

THEOREM 4.1 (Lower bound on the augmentation number (Eswaran and Tarjan [4])). *Let G be an undirected graph with h connected components, and let q be the number of isolated vertices in* $\text{blk}(G)$. *Then at least* $\max\{d + h - 2, \lceil \frac{l}{2} \rceil + q\}$ *edges are needed to biconnect G if $q + l > 1$.*

4.1. Stage 1.

THEOREM 4.2 (Rosenthal and Goldner [18]). *Let G be an undirected graph with h connected components. We can connect G by adding $h - 1$ edges, which we may choose to be incident on noncutpoint vertices in blocks corresponding to leaves or isolated vertices in* $\text{blk}(G)$.

Given $\text{blk}(G)$, stage 1 is easy to optimally parallelize in time $O(\log n)$ on an EREW PRAM by using the Euler tour technique described in Tarjan and Vishkin [23]. The block graph can be updated by creating a new b-vertex b and two new c-vertices c_v and c_w for each new edge (v, w). We create edges from b to c_v and b to c_w. Let b_v and b_w be the two b-vertices in the block graph whose corresponding blocks contain v and w, respectively. We create edges from c_v to b_v and from c_w to b_w.

4.2. Stage 2.

THEOREM 4.3 (Rosenthal and Goldner [18]). *Let G be connected and let v^* be a massive vertex in G. Let $\delta = d - 1 - \lceil \frac{l}{2} \rceil$. Then we can find at least $2\delta + 2$ v^*-chains. Let Q be the set of v-chain leaves. By adding $2k$, $k \leq \delta$, edges to connect $2k + 1$ vertices of Q, we can reduce both the degree of the massive vertex and the number of leaves in the block tree by k.*

COROLLARY 4.4 (Rosenthal and Goldner [18]). *Let G be connected and let v^* be a massive vertex in G. Let $\delta = d - 1 - \lceil \frac{l}{2} \rceil$, and let Q be the set of v^*-chain leaves. By adding 2δ edges to connect $2\delta + 1$ vertices of Q, we can obtain a balanced block tree.*

In stage 2, v^*-chain leaves can be found by first finding the number of leaves in each subtree rooted at a child of v^*. A leaf is in a v^*-chain if and only if it is in a one-leaf subtree rooted at a child of v^*. Let Q be the set of vertices (excluding v^*) on cycles created by adding edges. The new block graph can be updated by merging vertices in Q into a single b-vertex b. Vertices b and v^* are connected by a new edge. These procedures can be done optimally in time $O(\log n)$ on an EREW PRAM.

4.3. Stage 3. In this stage we have to deal with a graph G for which $\text{blk}(G)$ is balanced. The idea is to add an edge between two leaves y and z under the conditions that the path P between y and z passes through all critical vertices and the new block tree has two fewer leaves if $\text{blk}(G)$ has more than three leaves. Thus the degree of any critical vertex decreases by 1, and the tree remains balanced.

In Rosenthal and Goldner [18] $\text{blk}(G)$ is rooted at a b-vertex b^*. A path P is found that contains two leaves y and z such that if $\text{blk}(G)$ contains two critical vertices v and w, P contains both of them. If $\text{blk}(G)$ contains fewer than two critical vertices, P contains b^* and a c-vertex with degree d (recall that d is the maximum degree of any c-vertex). It is possible that in the case in which $\text{blk}(G)$ is balanced with more than three leaves and less than two critical vertices, P contains only one vertex of degree more than 2. If we add an edge between the two end points of P, it is possible that the new block tree has only one fewer leaf. An example of this is shown in Fig. 3. Thus the lower bound cannot be achieved by this method.

We now give a corrected version of stage 3 that runs in linear time. Our method is based on the proof of the tight bound given in Eswaran and Tarjan [4], but we add an additional step to handle the case $d = 2$ (that is, $a_1 = 1$); the analysis of this case is omitted in [4]. We present our revised version of stage 3 below.

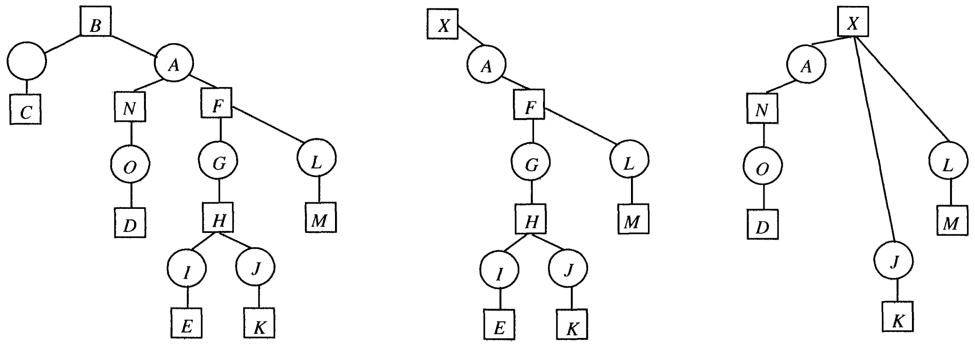


FIG. 3. *A counterexample for the linear-time sequential algorithm given by Rosenthal and Goldner* [18]. *The left tree is* blk(G) *rooted at* B. *Vertex A is the c-vertex with the largest degree. The middle tree is the new block tree after two noncutpoint vertices of G in the corresponding blocks represented by C and D are connected. The number of leaves decreases by* 1. *The right tree is the new block tree after two noncutpoint vertices of G in the corresponding blocks represented by C and E are connected. The number of leaves decreases by* 2. *The pair C and D can be chosen by the algorithm given by* [18], *and the pair C and E can be chosen to reduce the number of leaves by* 2.

```
graph function seq_bca(graph $G$);
{* $G$ has at least three vertices, and blk($G$) is balanced; $l$ is the number of degree-1 vertices
in blk($G$); $a_1 + 1$ is the largest degree of all $c$-vertices in blk($G$). *}
    tree $T$; vertex $v, w, y, z, \alpha, \beta$;
    let $T$ be blk($G$) rooted at an arbitrary $b$-vertex;
    do $l \geq 2$ $\rightarrow$
        if $a_1 = 1 \rightarrow$ if $l = 2 \rightarrow$ let $v$ be any $c$-vertex in $T$; $w := v$
                         | $l > 2 \rightarrow$
                 1.         let $v$ be a $b$-vertex with degree greater than 2; {* Such a vertex
                            must exist if $l > 2$ and $a_1 = 1$. *}
                            $w := v$; {* This is the default value for $w$. *}
                            if $\exists$ a $b$-vertex in $T - v$ with degree greater than 2 $\rightarrow$
                 2.             let $w$ be a $b$-vertex in $T - v$ with degree greater than 2
                            fi
                         fi
        | $a_1 > 1 \rightarrow$
                 3.      let $v$ be a $c$-vertex with the largest degree in $T$;
                         if the largest degree for $c$-vertices in $T - v$ is greater than 2 $\rightarrow$
                 4.         let $w$ be a $c$-vertex in $T - v$ with the largest degree
                         | the largest degree for $c$-vertices in $T - v$ is less than 3
                         or there is no $c$-vertex in $T - v$ $\rightarrow$
                            $w := v$; {* This is the default value for $w$. *}
                            if $\exists$ a $b$-vertex in $T$ with degree greater than 2 $\rightarrow$
                 5.             let $w$ be a $b$-vertex in $T$ with degree greater than 2
                            fi
                         fi
        fi;
    6.  find two leaves $y$ and $z$ such that the path between them passes through $v$ and $w$;
        find a noncutpoint vertex $\alpha$ in the corresponding block of $G$ represented by $y$;
        find a noncutpoint vertex $\beta$ in the corresponding block of $G$ represented by $z$;
        add an edge between $\alpha$ and $\beta$; update the block graph $T$
    od;
    return G
end seq_bca;
```

CLAIM 4.5. *If* blk(G) *is balanced, we can biconnect* G *by adding* $\lceil \frac{l}{2} \rceil$ *edges by using algorithm* seq_bca.

Proof. We first discuss the case for which blk(G) has more than three leaves. In this case a critical vertex must have degree more than 2.

Case 1. If blk(G) has two critical vertices v and w, then algorithm seq_bca finds them in steps 1 and 2 or in steps 3 and 4, respectively.

Case 2. If blk(G) has only one critical vertex v, algorithm seq_bca finds it in step 1 or step 3. Because blk(G) is balanced and $l > 3$, there must exist another vertex w with degree more than 2. Otherwise, v is massive. Algorithm seq_bca finds w in one of the steps 1, 2, 4, or 5.

Case 3. If blk(G) has no critical vertex, then either there is only one vertex (which must be a b-vertex) with degree more than 3 or there are two vertices with degree more than 2. If there is only one vertex v with degree more than 3, algorithm seq_bca finds v in step 1. Suppose there are two vertices v and w with degree more than 2 in the block tree. If v and w are both c-vertices, algorithm seq_bca finds them in steps 3 and 4, respectively. If v and w are both b-vertices, algorithm seq_bca finds them in steps 1 and 2, respectively. If one of v and w is a c-vertex and the other one is a b-vertex, algorithm seq_bca finds the c-vertex in step 3 and the b-vertex in step 5.

In all three cases we can find two vertices of degree more than 2 or a b-vertex of degree more than 3. Thus by Lemma 3.7 the number of leaves in the new block tree reduces by two. Because v and w are the possible critical vertices, we reduce the value of d by 1. Thus the block tree remains balanced. Hence we can achieve the lower bound in Eswaran and Tarjan [4] by the algorithm.

For the case of $l = 3$ we can reduce blk(G) to a new block tree with two leaves by picking any pair of leaves in blk(G) and connecting them. We know that we can reduce a block tree of two leaves into a single vertex by connecting the two leaves. Thus the claim is true. □

CLAIM 4.6. *Algorithm* seq_bca *runs in* $O(n + m)$ *time.*

Proof. The block tree can be built in $O(n + m)$ time. The total number of vertices in the block tree is $O(n)$. A linear-time bucket-sort routine is used to sort degrees of c-vertices and b-vertices. The data structure in Rosenthal and Goldner [18] can be used to keep track of current degrees of vertices in blk(G). Vertices in blk(G) with the same degree are kept in a linked list. An array is used to store the first element of each linked list. A vertex of the largest degree can be found in constant time, and the position, in the array of the linked list, of a vertex in the path found in step 6 can also be updated in constant time. To implement step 6, algorithms in Harel and Tarjan [10] and Schieber and Vishkin [19] are used to find the path P between two vertices v and w in $O(|P|)$ time. By Fact 3.6 the number of times a vertex is visited is no more than its degree. Since the summation of degrees of all vertices in a tree with n vertices is $O(n)$, the claim is true. □

In the rest of this section we describe an efficient parallel algorithm for stage 3. Recall that the sequential algorithm adds one edge at a time and keeps adding edges until the block tree becomes a single vertex. In our parallel algorithm, however, we will find several pairs of leaves such that the path between any such pair of leaves passes through all critical c-vertices, if any. Thus the degrees of critical vertices in the new block tree decrease by the number of edges added to the original block tree. These pairs also satisfy the leaf-connecting condition (Definition 2.3), which guarantees that the number of leaves in the new block tree decreases by twice the number of edges added. The following lemma tells us that the addition of several edges in parallel as outlined above is a valid strategy.

LEMMA 4.7. *Let G be a graph whose block graph is balanced, and let G' be the graph obtained from G by adding a set of k edges $\mathcal{A} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$. For each i, $0 \le i \le k$, let G_i be the graph obtained from G by adding the set of edges $\{(s_1, t_1), \ldots, (s_i, t_i)\}$. Let s'_i and t'_i, $0 \le i \le k$, be the b-vertices in* blk(G_i) *whose corresponding blocks contain s_i and t_i, respectively. If the path between s'_{i+1} and t'_{i+1} $\forall i$, $1 \le i < k$, in* blk(G_i) *passes through all critical vertices in* blk(G_i) *and if s'_{i+1} and t'_{i+1} satisfy the leaf-connecting condition in G_i, then* blk(G') *remains balanced and the value of the lower bound given in Theorem* 4.1 *applied to G' is k less than the same lower bound applied to G.*

Proof. We always obtain the same block graph for G_k, no matter in what sequence we choose to add these k edges, since there is a unique block graph for each graph G. Thus blk(G') = blk(G_k). Since blk(G_i) $\forall i$, $1 \le i \le k$, is balanced, blk(G') is balanced. We know that the value of the lower bound given in Theorem 4.1 applied to G_i is 1 less than the value of the same lower bound applied to G_{i-1} $\forall i$, $1 \le i \le k$, where $G_0 = G$. Hence the value of the lower bound given in Theorem 4.1 applied to G' is k less than the value of the same lower bound applied to G. □

From Theorem 4.1 and Claim 4.5 we know that exactly $\lceil \frac{l}{2} \rceil$ edges must be added to biconnect G if blk(G) is balanced. That is, we have to eliminate l leaves during the computation. Our parallel algorithm runs in stages with at least $\frac{1}{4}$ of the current leaves eliminated in parallel time $O(\log n)$ with a linear number of processors during each stage. We call this subroutine $O(\log n)$ times to complete the augmentation.

Recall that $a_i + 1$ is equal to the degree of the ith c-vertex v_i and $a_i \ge a_{i+1}$. T' is the subtree obtained from T by deleting the subtree rooted at v_1. Let $U_i = \{u \mid u$ is the leftmost leaf of T_y, where y is a child of $v_i\}$. For example, the leaves in U_1 are illustrated as shaded rectangles in Fig. 4.

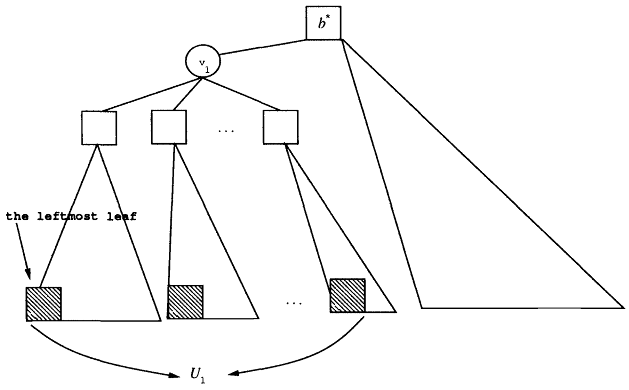


FIG. 4. *Each shaded rectangle represents the leftmost leaf in a subtree rooted at a child of v_1. Leaves in U_1 consist of leftmost leaves in every subtree rooted at a child of v_1.*

Depending on the degree distribution of vertices in the block tree, the parallel algorithm for stage 3 is divided into two cases. In case 1, $a_1 > \frac{l}{4}$. We have a c-vertex with a high degree. We pick the first $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ leaves in U_1 and call them W_1. Leaves in W_1 are matched with the first $\min\{|W_1|, |U_2| - 1\}$ leaves in U_2. Unmatched leaves in W_1, if any, are matched with all remaining leaves but one in T' and, finally, are properly matched within themselves, if necessary. In case 2, $a_1 \le \frac{l}{4}$. There is no c-vertex with a large degree. We show that we can find a vertex u^* with approximately the same number of leaves in each subtree rooted at a child of u^*. If u^* is a b-vertex, a suitable number of leaves between subtrees rooted at children of u^* are matched. Otherwise, u^* is a c-vertex and a suitable number of subtrees rooted at children of u^* are first merged into a single subtree rooted at u^*. Then leaves in the merged subtree are matched with leaves outside.

The algorithm first finds the matched pairs of leaves in each case. Then we add edges between matched pairs of leaves and update the block tree at the end of each case. The block tree and the sequence of cutpoints $v_1, \ldots, v_{n_c}$ will not be changed during the execution of each case.

We now describe the two cases in detail.

4.3.1. Case 1 ($a_1 > \frac{l}{4}$). We root the block tree at the b-vertex b^*, which is adjacent to v_1 and is on the path from v_1 to v_2. Let v_1 be the leftmost child of b^*. We permute the children of v_1 in nonincreasing order (from left to right) of the number of leaves in subtrees rooted at them. We call this procedure *tree normalization*, and we call the resulting tree T. Figure 5 illustrates a normalized tree.

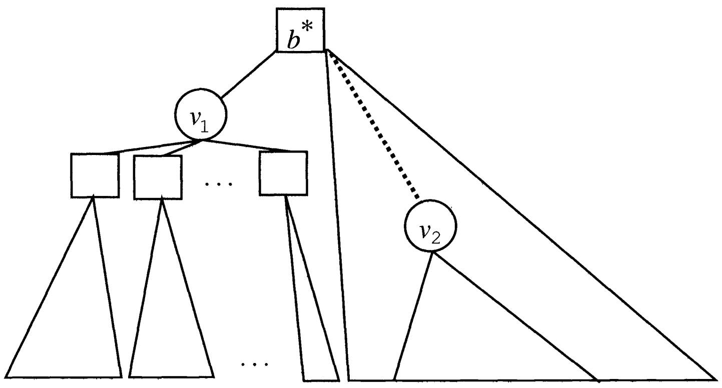


FIG. 5. *A normalized tree. Vertex v_1 is a c-vertex with the largest degree. Vertex v_2 is a c-vertex with a degree greater than or equal to any other c-vertices in $T - v_1$. We permute the children of v_1 in nonincreasing order (from left to right) of the number of leaves in subtrees rooted at them.*

Recall that U_1 is the set of leftmost leaves in subtrees rooted at children of v_1. We select the first (from left to right) $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ leaves from U_1 and call the set W_1. The order of the leaves as specified in the original tree is preserved. There are four phases for this case. In phases 1 and 2, leaves in W_1 are matched with leaves not in T_1. In phase 3, leaves in W_1 are matched with leaves in T_1 excluding those in W_1. In phase 4, the remaining leaves in W_1 are matched between themselves. The algorithm executes each phase in turn once until there is no leaf in W_1 left to be matched.

We now describe the four phases in detail. After the description we give the overall parallel algorithm for case 1 and prove that it eliminates a constant fraction of the leaves while maintaining the lower bound described in Theorem 4.1.

Phase 1. All leaves but the rightmost one in U_2 are matched with the rightmost $a_2 - 1$ leaves of W_1. The matched leaves are removed from W_1. An example of the pairs of leaves matched in phase 1 is given in Fig. 6.

```
set of pairs of vertices function phase1(modifies set of vertices $W_1$, $U_2$);
    set of pairs of vertices $L$; vertex $u$, $v$;
    $L$ := {}; {* $L$ is the set of matched pairs. *}
    number leaves in $W_1$ from right to left starting from 1;
```

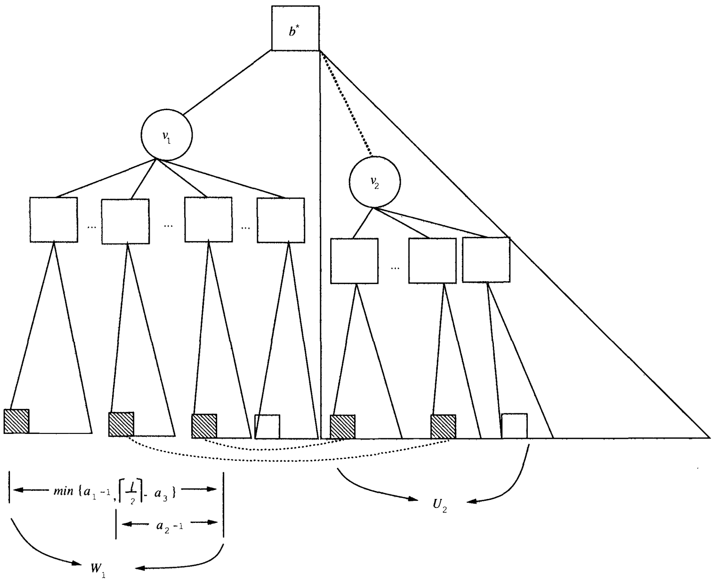


FIG. 6. *Pairs of matched leaves found in phase* 1 *of case* 1 *are connected by dotted lines.* W_1 *consists of the leftmost leaves from the first* $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ *subtrees rooted at children of* v_1. *All of the leftmost leaves in subtrees rooted at children of* v_2 *are in the set* U_2, *and all except the rightmost leaf in* U_2 *are matched* (*if possible*).

```
    number leaves in $U_2$ from left to right starting from 1;
    $k := \min\{|U_2| - 1, |W_1|\}$;
    **pfor** $i = 1 \mathrel{..} k \rightarrow$
        $u$ := the $i$th leaf in $W_1$; remove $u$ from $W_1$;
        $v$ := the $i$th leaf in $U_2$; remove $v$ from $U_2$;
        $L := L \cup \{(u, v)\}$
    **rofp**;
    **return** $L$
**end** phase1;
```

Phase 2. All remaining leaves except one in T' are matched with the rightmost leaves of W_1, and matched leaves are removed from W_1. An example of the pairs of leaves matched in phase 2 is given in Fig. 7.

```
**set of pairs of vertices function** phase2(**modifies set of vertices** $W_1$, **tree** $T'$);
    **set of pairs of vertices** $L$; **vertex** $u$, $v$;
    $L := \{\}$; {* $L$ is the set of matched pairs. *}
    number leaves in $W_1$ from right to left starting from 1;
    number leaves in $T'$ from left to right starting from 1;
    $k$ := min{the number of leaves in $T'$ minus 1, $|W_1|$};
    **pfor** $i = 1 \mathrel{..} k \rightarrow$
        $u$ := the $i$th leaf in $W_1$; remove $u$ from $W_1$; $v$ := the $i$th leaf in $T'$;
        $L := L \cup \{(u, v)\}$
    **rofp**;
    **return** $L$
**end** phase2;
```

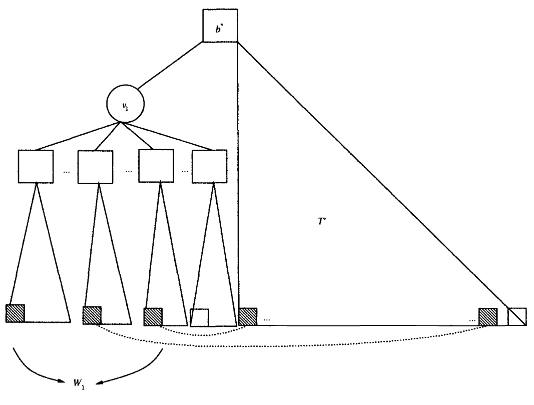


FIG. 7. *Pairs of matched leaves found in phase 2 of case 1 are connected by dotted lines. Recall that T_1 is the subtree of T rooted at v_1. T' is the subtree of T obtained by deleting T_1. W_1 consists of the leftmost leaves in the first* $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\} - a_2 + 1$ *subtrees rooted at children of v_1. Leaves in W_1 are matched with all but the rightmost leaf in T'.*

Phase 3. Recall that T is the original block tree before phase 1, l is the number of leaves in T, v_1 is a c-vertex with the largest degree in T, T_1 is the subtree of T rooted at v_1, l_1 is the number of leaves in T_1, T' is the tree obtained from T by removing T_1, and $U_1 = \{u \mid u$ is the leftmost leaf of T_y, where y is a child of $v_1\}$. Note that there are $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\} - (l - l_1 - 1)$ leaves remaining in W_1. Leaves in W_1 come from the first $|W_1|$ members (from left to right) of U_1. Let the set of v_1-chain leaves in W_1 be Q_1. We denote by Q_2 the set of leaves other than the rightmost one of each subtree rooted at a child of v_1. (Note that $Q_1 \cap Q_2 = \emptyset$.) In this phase we match all leaves in Q_1 (i.e., all v_1-chain leaves in W_1) with an equal number of leaves in Q_2. Leaves in W_1 that are matched in phase 3 (Q_1 and $W_1 \cap Q_2$) are removed from W_1.

Claim 4.8 below shows that we can always find enough leaves in Q_2 to match all leaves in Q_1.

CLAIM 4.8. $|Q_2| \geq |Q_1|$ *if* $l > 3$.

Proof. If $|Q_1| = 0$, the claim is true. Let $|Q_1| > 0$. Recall that there is only one unmatched leaf s left in T' after phase 2. Let T^* be the block tree obtained from T by adding edges between matched pairs of leaves found in phase 1 and phase 2. We root T^* at the b-vertex b^* that is adjacent to v_1 and is on the path from v_1 to s. Let r be the number of subtrees rooted at a child of v_1 in T^* with more than one leaf. Let y be the number of v_1-chain leaves not in Q_1. The notations used in this proof are shown in Fig. 8.

The total number of leaves in T^* is equal to $|Q_1| + |Q_2| + r + y + 1$ if $|Q_1| > 0$. The degree of v_1 in T^* is equal to $|Q_1| + r + y + 1$. Since T^* is balanced (for a proof, see Claim 4.10 at the end of this section), v_1 is not massive and hence

$$|Q_1| + r + y \leq \left\lceil \frac{|Q_1| + |Q_2| + r + y + 1}{2} \right\rceil.$$

Thus

$$2|Q_1| + 2r + 2y \leq |Q_1| + |Q_2| + r + y + 2 \Rightarrow |Q_1| + r + y - 2 \leq |Q_2|.$$

We know that $r \geq 1$; otherwise v_1 is massive if $l > 3$. It is also true that $y \geq 1$ if $|Q_1| > 0$. Thus $|Q_1| \leq |Q_2|$. □

The procedure for phase 3 is described below.

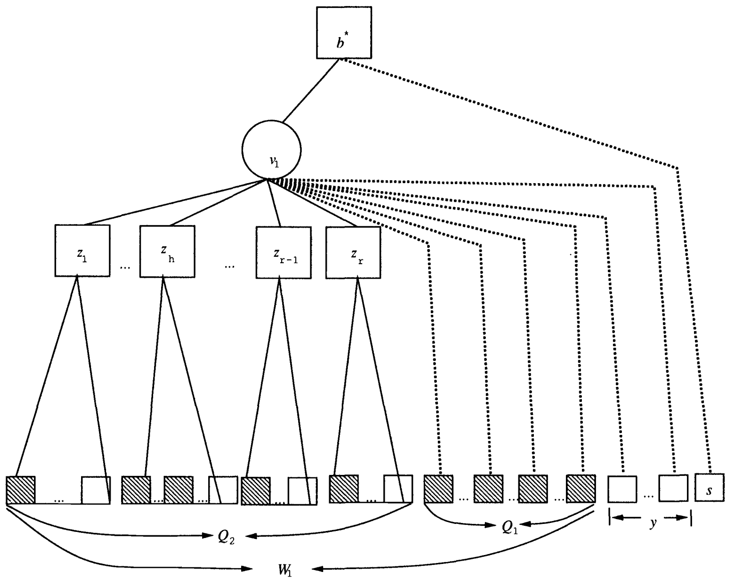


FIG. 8. *Notations used in the proof of a claim used in phase* 3 *of case* 1. *The tree shown is* T^*, *the updated block tree obtained by adding edges between pairs of matched leaves found in phase* 1 *and phase* 2. Q_2 *consists of all but the rightmost leaf in each subtree rooted at a child of* v_1. Q_1 *consists of* v_1*-chain leaves in* W_1 *after phase* 2. *The number of subtrees rooted at a child of* v_1 *with more than one leaf is* r. *The number of* v_1*-chain leaves not in* Q_1 *is* y.

```
set of pairs of vertices function phase3(modifies set of vertices $Q_1$, $Q_2$);
    set of pairs of vertices $L$; vertex $u$, $v$;
    $L$ := {}; {* $L$ is the set of matched pairs. *}
    number leaves in $Q_2$ from right to left starting from 1;
    number leaves in $Q_1$ from 1 to $|Q_1|$ in arbitrary order;
    $k := |Q_1|$;
    pfor $i = 1 .. k \rightarrow$
        $u$ := the $i$th leaf in $Q_2$; remove $u$ from $Q_2$;
        $v$ := the $i$th leaf in $Q_1$; remove $v$ from $Q_1$;
        $L := L \cup \{(u, v)\}$
    rofp;
    return $L$
end phase3;
```

Phase 4. The remaining leaves of W_1 that are not matched during phase 3 are matched within themselves. If the number of remaining leaves in W_1 is odd, we match one of them with the rightmost leaf in the subtree rooted at v_1. An example of the pairs of leaves matched in phase 4 is given in Fig. 9.

```
set of pairs of vertices function phase4(modifies set of vertices $W_1$, tree $T$);
    set of pairs of vertices $L$; vertex $u$, $v$;
    $L$ := {}; {* $L$ is the set of matched pairs. *}
    number leaves in $W_1$ in arbitrary order from 1 to $|W_1|$;
    $k := \lceil \frac{|W_1|}{2} \rceil$;
    pfor $i = 1 .. k \rightarrow$
        $u$ := the $(2 * i - 1)$th leaf in $W_1$; remove $u$ from $W_1$;
```

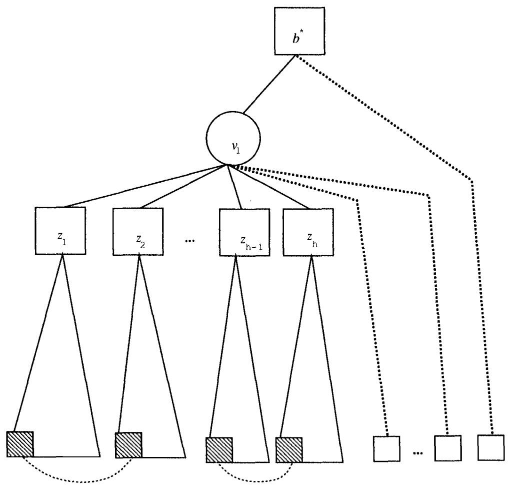


FIG. 9. *Illustration of phase 4 of case 1. The remaining leaves in W_1 are matched within themselves.*

```
        if $2 * i \leq |W_1| \rightarrow v :=$ the $(2 * i)$th leaf in $W_1$; remove $v$ from $W_1$
        | $2 * i > |W_1| \rightarrow v :=$ the rightmost leaf in the subtree rooted at $v_1$
        fi;
        $L := L \cup \{(u, v)\}$
    rofp;
    return $L$
end phase4;
```

We now describe our algorithm for case 1.

```
set of pairs of vertices function case1(tree $T$);
    set of pairs of vertices $L$; set of vertices $W_1$, $Q_1$, $Q_2$; vertex $b^*$; tree $T'$;
    root $T$ at the $b$-vertex $b^*$ that is adjacent to $v_1$ and is on the path from $v_1$ to $v_2$;
    let $v_1$ be the leftmost child of $b^*$ in $T$;
    permute the children of $v_1$ in nonincreasing order (from left to right)
    of the number of leaves in subtrees rooted at them;
    $W_1 :=$ the first (from left to right) $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ leaves of $U_1$;
    $L :=$ phase1($W_1$, $U_2$); {* $L$ is the set of matched pairs. *}
    if $W_1 \neq \{\} \rightarrow L := L \cup$ phase2($W_1$, $T'$) fi;
    $T_1' :=$ the subtree of $T_1$ with the first $|W_1|$ subtrees rooted at children of $v_1$;
    $Q_1 :=$ the set of $v_1$-chain leaves in $T_{v_1}$;
    $Q_2 := \{u | u$ is a nonleftmost leaf of $T_y$, where $T_y$ has more than 1 leaf
    and $y$ is a child of $v_1$ in $T_1'\}$;
    if $W_1 \neq \{\} \rightarrow L := L \cup$ phase3($Q_1$, $Q_2$) fi; $W_1 := W_1 \cap Q_2$;
    if $W_1 \neq \{\} \rightarrow L := L \cup$ phase4($W_1$, $T$) fi;
    return $L$
end case1;
```

CLAIM 4.9. *The number of matched pairs k in case 1 satisfies $\lceil \frac{l}{2} \rceil - a_3 \geq k \geq \frac{l}{8}$, if $l > 3$.*

Proof. Let $z = \min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$. If the procedure does not execute phases 3 and 4, we match z pairs. Because $a_1 - 1 \geq \frac{l}{4}$ and $\lfloor \lceil \frac{l}{2} \rceil - a_3 \rfloor \geq \lfloor \frac{l}{8} \rfloor$ for $l > 3$ (Corollary 3.2), we know that $z \geq \frac{l}{8}$ if $l > 3$. Otherwise, in the worst case we match only $a_2 - 1$ pairs during phase 1 and phase 2. A pair of vertices matched during phase 3 or 4 might both be members of W_1. Thus

$$k \geq a_2 - 1 + \left\lceil \frac{z - a_2 + 1}{2} \right\rceil \geq \frac{z + a_2 + 1}{2}.$$

If $z = \lceil \frac{l}{2} \rceil - a_3$, then $k \geq \frac{\lceil \frac{l}{2} \rceil - 1}{2}$, which is greater than or equal to $\frac{l}{8}$ if $l > 3$ and k is an integer. If $z = a_1 - 1$, then $k \geq \lceil \frac{a_1 - 1}{2} \rceil$. Because $a_1 > \frac{l}{4}$, $k \geq \frac{l}{8}$. □

CLAIM 4.10. (1) *Each pair of matched vertices found in function* case1 *satisfies the leaf-connecting condition* (*Definition* 2.3).

(2) *Let us place an edge between each matched pair found in function* case1 *sequentially and update the block graph each time we add an edge. Critical vertices, if any, of the block graph are on the path between the end points of each edge placed.*

Proof. From part (4) of Fact 3.6, degrees of v_1 and v_2 decrease only by 1 if an edge is added between a pair of vertices matched. Let us consider paths between pairs of vertices matched in each phase. We show that we can find at least two vertices with degree more than 2 in each path.

Phase 1. The path between each pair of matched vertices passes through v_1 and v_2, whose degrees are at least 3.

Phase 2. The path between each pair of matched vertices passes through v_1 and the root b^*, whose degrees are at least 3.

Phases 3 *and* 4. The path P between each pair of matched vertices passes through v_1, whose degree is at least 3. P also passes through a child u of v_1, where the subtree rooted at u has more than one leaf. Thus the degree of u is greater than 2.
Thus the leaf-connecting condition (Definition 2.3) holds for each pair of matched vertices.

Because we add only $\min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ edges during case 1, v_3 and thus v_i $\forall i$ such that $i \geq 4$, do not become critical. From the previous discussion, the path between each pair of matched vertices passes through v_1 and v_2, the only two possible critical vertices, during phase 1. We reduce degrees of possible critical vertices by 1 by adding one new edge between each pair of matched vertices. If we match any pair of vertices after phase 1, the degree of v_2 is at most 2 and the degree of v_1 is at least 3. Thus v_1 is the only possible critical vertex. The path between each pair of matched vertices passes through v_1 after phase 1. We reduce the degree of the possible critical vertex v_1 by one by adding one new edge between any pair of matched vertices. Thus the claim is true. □

COROLLARY 4.11. *Let k be the number of matched pairs found in function* case1. *Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem* 4.1 *applied to G' is k less than the value of the same lower bound applied to G, and* blk(G') *remains balanced. Let l be the number of leaves in* blk(G). *The number of leaves in* blk(G') *is at most $\frac{3l}{4}$ if $l > 3$.*

Proof. From part (1) in Claim 4.10, the number of leaves in blk(G') is $2k$ fewer than the number of leaves in blk(G). Since $k \geq \frac{l}{8}$ if $l > 3$ (Claim 4.9), the number of leaves in blk(G') is at most $\frac{3l}{4}$ if $l > 3$. From part (2) in Claim 4.10, the block graph of each intermediate graph remains balanced even if we place a new edge between each matched pair of leaves found in function case1 sequentially. By Lemma 4.7 we know that the value of the lower bound given in Theorem 4.1 applied to G' is k less than the value of the same lower bound applied to G, and blk(G') remains balanced. □

4.3.2. Case 2 ($a_1 \leq \frac{l}{4}$). In this case we take advantage of the fact that no c-vertex has a large degree. Because there is no critical c-vertex, the algorithm can add at least $\lceil \frac{l}{2} \rceil - a_1$ edges between leaves that satisfy the leaf-connecting condition (Definition 2.3) without regard to whether the path between them passes through a critical c-vertex. This gives a certain degree of freedom for us to choose the matched pairs. We first root the block tree such that no subtree other than the one rooted at the root has more than half of the total number of leaves.

Given any rooted tree T, we use l_v to denote the number of leaves in the subtree rooted at a vertex v. The following lemma shows that we can reroot T at a vertex u^* such that no subtree rooted at a child of u^* has more than half of the total number of leaves.

LEMMA 4.12. *Given a rooted tree T, there exists a vertex u^* in T such that $l_{u^*} > \frac{l}{2}$, but none of the subtrees rooted at children of u^* has more than $\frac{l}{2}$ leaves.*

Proof. We permute children of each nonleaf vertex v from left to right in nonincreasing order of the number of leaves in the subtrees rooted at them. Let us consider the leftmost path P of the tree T. It is obvious that there exists such a vertex u^* in P. □

We root the block tree at u^* and permute children of u^* from left to right in nonincreasing order of the number of leaves in subtrees rooted at them. Let the rooted tree be T. Let u_i $\forall i$, $1 \le i \le r$, be the children (from left to right) of u^*, and let x_i be the number of leaves in the subtree rooted at u_i. Note that $x_i \le \frac{l}{2}$ for each i. There are two subcases, depending on whether u^* is a b-vertex or a c-vertex. We describe the two subcases in detail in the following paragraphs.

Subcase 2.1 (u^* is a b-vertex). We show that we can partition subtrees rooted at children of the root into two sets evenly such that we can match leaves between the two partitions. We first give a claim to show how to perform the partition.

CLAIM 4.13. *There exists p such that $1 \le p < r$ and $\frac{l}{2} \ge \sum_{i=1}^{p} x_i > \frac{l}{4}$.*

Proof. We know that $x_i \ge x_{i+1}$ $\forall i$, $1 \le i < r$, and that $x_i \le \frac{l}{2}$ $\forall i$, $1 \le i \le r$. Thus $\exists p$, $1 \le p < r$, such that

$$\sum_{i=1}^{p} x_i \le \frac{l}{2} \quad \text{and} \quad \sum_{i=1}^{p+1} x_i > \frac{l}{2}.$$

Because $x_i \ge x_{i+1}$ $\forall i$, $1 \le i < r$, we know that $\sum_{i=1}^{p} x_i > \frac{1}{2}(\frac{l}{2})$. □

The notations used for this subcase are illustrated in Fig. 10.

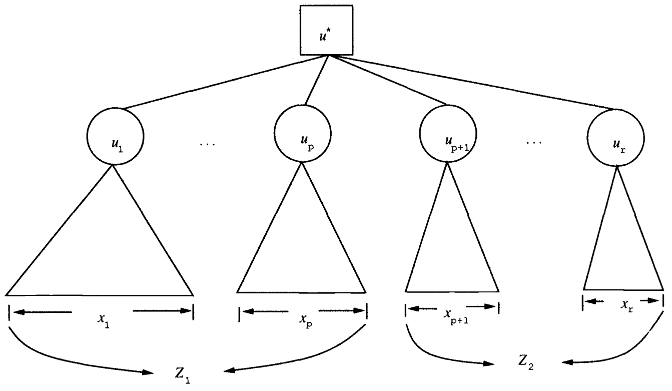


FIG. 10. *Notations used in subcase 2.1. The number of leaves in the subtree rooted at u_i is x_i. We find the largest p such that the total number of leaves in the first p subtrees rooted at children of the root is greater than $\frac{l}{4}$ but less than or equal to $\frac{l}{2}$. Leaves in the first p subtrees rooted at children of b^* are in Z_1. Z_2 consists of the rest of the leaves in the tree.*

COROLLARY 4.14. $\sum_{i=1}^{p} x_i \le l - \sum_{i=1}^{p} x_i$.

We match $\min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\}$ leaves in subtrees T_{u_i} $\forall i, 1 \leq i \leq p$, with leaves outside them. From Claim 4.13 and Corollary 4.14 we know that the matching can be done.

COROLLARY 4.15. *The number of matched pairs k in subcase* 2.1 *satisfies* $\lceil \frac{l}{2} \rceil - a_1 \geq k > \frac{l}{4}$ *if* $l > 3$.

```
set of pairs of vertices function case2_1(tree T);
{* $l$ is the number of leaves in $T$. *}
    integer $p$; set of pairs of vertices $L$; set of vertices $Z_1, Z_2$; vertex $u, v$;
    let $u_i$ be the $i$th (from left to right) child of the root;
    let $x_i$ be the number of leaves in the subtree rooted at $u_i$;
    find the largest integer $p$ such that $\sum_{i=1}^{p} x_i \leq \frac{l}{2}$ but $\sum_{i=1}^{p+1} x_i > \frac{l}{2}$;
    $L := \{\}$; {* $L$ is the set of matched pairs. *}
    $Z_1$ := the set of leaves in the subtrees rooted at $u_i$ $\forall i, 1 \leq i \leq p$;
    $Z_2$ := the set of leaves in the subtrees rooted at $u_i, i > p$;
    number leaves in $Z_1$ in arbitrary order from 1 to $|Z_1|$;
    number leaves in $Z_2$ in arbitrary order from 1 to $|Z_2|$;
    $k := \min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\}$;
    pfor $i = 1 .. k \rightarrow$
        $u, v$ := the $i$th vertex in $Z_1$ and $Z_2$, respectively;
        $L := L \cup \{(u, v)\}$
    rofp;
    return $L$
end case2_1;
```

CLAIM 4.16. *Any matched pair found in function* case2_1 *satisfies the leaf-connecting condition* (*Definition* 2.3) *if* $l > 3$.

Proof. Consider the path P between a pair of matched leaves u and v. Let u be a leaf in a subtree rooted at a u_x, $1 \leq x \leq p$, and let v be a leaf in a subtree rooted at a u_y, $p < y \leq r$. Since we match $\min\{|Z_1| - 1, \lceil \frac{l}{2} \rceil - a_1\}$ leaves in Z_1 with an equal number of leaves in Z_2 and $|Z_1| \leq |Z_2|$ (Corollary 4.14), there is at least one leaf in a subtree rooted at a u_i, $1 \leq i \leq p$, that is not matched and there is also another leaf in a subtree rooted at a u_j, $p < j \leq r$, that is not matched if $l > 3$. The path P contains the root. If the degree of the root is at least 4, u and v satisfy the leaf-connecting condition (Definition 2.3). If the degree of the root is 3, P contains either u_i or u_j, whose degree is at least 3. Otherwise, P contains both u_i and u_j, whose degrees are at least 3. Thus u and v satisfy the leaf-connecting condition (Definition 2.3). □

COROLLARY 4.17. *Let k be the number of matched pairs found in function* case2_1. *Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem* 4.1 *applied to G' is k less than the value of the same lower bound applied to G, and* blk(G') *remains balanced. Let l be the number of leaves in* blk(G). *The number of leaves in* blk(G') *is at most* $\frac{l}{2}$ *if* $l > 3$.

Proof. From Claim 4.16 the number of leaves in blk(G') is $2k$ less than the number of leaves in blk(G). Since $k \geq \frac{l}{4}$ if $l > 3$ (Corollary 4.15), the number of leaves in blk(G') is at most $\frac{l}{2}$. From Corollary 4.15 we add at most $\lceil \frac{l}{2} \rceil - a_1$ edges; thus no c-vertex in blk(G') becomes massive. By Lemma 4.7 we know that the value of the lower bound given in Theorem 4.1 applied to G' is k less than the value of the same lower bound applied to G, and blk(G') remains balanced. □

Subcase 2.2 (u^* is a c-vertex). Recall that the u_i $\forall i, 1 \leq i \leq r$, are the children (from left to right) of u^* (the root). Let x_i be the number of leaves in the subtree rooted at u_i. We know that $\frac{l}{2} \geq x_i$ $\forall i, 1 \leq i \leq r$, and that $x_i \geq x_{i+1}$ $\forall i, 1 \leq i < r$.

We partition the set of subtrees rooted at children of the root into two sets such that we can match leaves between two sets. We first give a claim to show how to partition the set of subtrees.

CLAIM 4.18. *Let q be the largest integer with $x_q \geq 2$. There exists an integer p such that $1 \leq p \leq q$ and $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{l}{8} + (p-1)$.*

Proof. If $x_1 > \frac{l}{8}$, then $p = 1$. If $x_1 \leq \frac{l}{8}$, we can find an integer p such that $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{3l}{8}$ by using an argument similar to the one given in the proof of Claim 4.13. By definition we know that $x_p \geq 2$ because otherwise the root (a c-vertex) is massive. Thus $p \leq \frac{l}{4}$. Hence $(\sum_{i=1}^{p} x_i) - (p-1) > \frac{l}{8}$. □

Let T_{u_i} be the subtree rooted at u_i. We define the *merge* operation for the collection of subtrees T_{u_i} $\forall i$, $1 \leq i \leq p$, as follows. We first connect the rightmost leaf of T_{u_i} and the leftmost leaf of $T_{u_{i+1}}$ $\forall i$, $1 \leq i < p$. This can be done by the fact that each T_{u_i} $\forall i$, $1 \leq i \leq p$, has at least two leaves.

CLAIM 4.19. *Let T^* be the block tree obtained from T by collapsing b-vertices that are in the same fundamental cycle created by the addition of new edges introduced by the merge operation.*

(1) *The merge operation creates only one b-vertex b^*.*

(2) *Vertex b^* is a child of the root, and b^* is the root of the subtree that contains the updated portion of the block tree.*

Proof. Let C_i $\forall i$, $1 \leq i < p$, be the fundamental cycle created by connecting the rightmost leaf of T_{u_i} and the leftmost leaf of $T_{u_{i+1}}$. The cycles C_i and C_{i+1} $\forall i$, $1 \leq i < p-1$, share the b-vertex u_i. From part (2) in Fact 3.6 we know that all b-vertices in cycles C_i $\forall i$, $1 \leq i < p$, shrink into a single b-vertex in the new block tree. Let this new b-vertex be b^*. Thus part (1) of the claim is true. Part (2) of the claim follows from part (4) in Fact 3.6. □

Note that if we root the updated block tree T^* given in Claim 4.19 at the b-vertex b^*, the situation is similar to that in case 2.1. Thus we can match an additional $\min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$ pairs of vertices by pairing up unmatched leaves in subtrees T_{u_i} $\forall i$, $1 \leq i \leq p$, and leaves in subtrees in subtrees T_{u_i} $\forall i$, $p < i \leq r$. This procedure is given below in case2_2. The notations used are shown in Fig. 11.

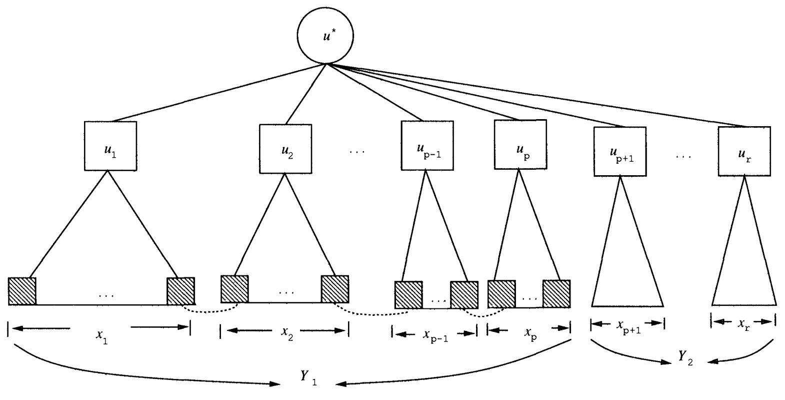


FIG. 11. *Notations used in subcase 2.2. The number of leaves in the subtree rooted at u_i is x_i. We find the largest p such that the total number of leaves in the first p subtrees rooted at children of the root is greater than $\frac{l}{8} + (p-1)$ but at most $\frac{l}{2}$. We first merge subtrees rooted at u_i $\forall i$, $1 \leq i \leq p$, by connecting the rightmost leaf in the subtree rooted at u_i and the leftmost leaf in the subtree rooted at u_{i+1} $\forall i$, $1 \leq i < p$. Leaves in the first p subtrees rooted at children of u^* are in Y_1. Y_2 consists of the rest of leaves in the tree. We then match* $\min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$ *leaves in Y_1 with leaves in Y_2.*

COROLLARY 4.20. *The number of matched pairs k in subcase* 2.2 *satisfies* $\lceil \frac{l}{2} \rceil - a_1 \geq k \geq \frac{l}{8}$ *if* $l > 3$.

```
set of pairs of vertices function case2_2(tree T);
    vertex $u, v$; integer $p$; set of vertices $Y_1, Y_2$; set of pairs of vertices $L$;
    let $u_i$ $\forall i, 1 \leq i \leq r$, be the children of the root $u^*$;
    let $T_{u_i}$ be the subtree rooted at $u_i$; let $x_i$ be the number of leaves in $T_{u_i}$;
    find the largest integer $p$ such that $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{l}{8} + (p-1)$;
    $Y_1$ := the set of leaves in the subtrees rooted at $u_i$ $\forall i, 1 \leq i \leq p$;
    $Y_2$ := the set of leaves in the subtrees rooted at $u_i, i > p$;
    $L := \{\}$; {* $L$ is the set of matched pairs. *}
    pfor $i = 1 .. p-1 \rightarrow$
        let $u$ be the leftmost leaf of $T_{u_i}$; let $v$ be the rightmost leaf of $T_{u_{i+1}}$;
        $L := L \cup \{(u, v)\}$; remove $u$ and $v$ from $Y_1$
    rofp;
    number the leaves in $Y_1$ in arbitrary order from 1 to $|Y_1|$;
    number the leaves in $Y_2$ in arbitrary order from 1 to $|Y_2|$;
    $k := \min\{\sum_{i=1}^{p} x_i, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$;
    pfor $i = 1 .. k \rightarrow$
        $u, v$ := the $i$th vertex in $Y_1$ and $Y_2$, respectively;
        $L := L \cup \{(u, v)\}$
    rofp;
    return $L$
end case2_2;
```

CLAIM 4.21. *Each pair of vertices matched in function* case2_2 *satisfies the leaf-connecting condition* (*Definition* 2.3) *if* $l > 3$.

Proof. The proof is by Claim 4.19 and similar arguments given in the proof of Claim 4.16. □

COROLLARY 4.22. *Let k be the number of matched pairs found in function* case2_2. *Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem* 4.1 *applied to G' is k less than the value of the same lower bound applied to G, and* blk(G') *remains balanced. Let l be the number of leaves in* blk(G). *The number of leaves in* blk(G') *is at most* $\frac{3l}{4}$ *if* $l > 3$.

Proof. From Claim 4.21 the number of leaves in blk(G') is $2k$ less than the number of leaves in blk(G). Since $k \geq \frac{l}{8}$ if $l > 3$ (Corollary 4.20), the number of leaves in blk(G') is at most $\frac{3l}{4}$. From Corollary 4.20 we add at most $\lceil \frac{l}{2} \rceil - a_1$ edges; thus no c-vertex in blk(G') becomes massive. By Lemma 4.7 we know that the value of the lower bound given in Theorem 4.1 applied to G' is k less than the value of the same lower bound applied to G, and blk(G') remains balanced. □

The complete procedure for case 2 is shown below.

```
set of pairs of vertices function case2(tree $T$);
{* $l$ is the number of leaves in $T$; $a_1 + 1$ is the largest degree of all $c$-vertices in $T$. *}
    vertex $u^*$;
    root $T$ at an arbitrary vertex;
    find a vertex $u^*$ such that there are more than $\frac{l}{2}$ leaves in the subtree rooted at
    $u^*$ but none of the subtrees rooted at a child of $u^*$ has more than $\frac{l}{2}$ leaves;
    root $T$ at $u^*$;
    permute children of $u^*$ (from left to right) in nonincreasing order of
    the number of leaves in subtrees rooted at them;
    if $u^*$ is a $b$-vertex $\rightarrow$ return case2_1($T$) | $u^*$ is a $c$-vertex $\rightarrow$ return case2_2($T$)
fi
end case2;
```

The correctness of this algorithm was shown earlier in the two subcases (Corollary 4.17 and Corollary 4.22).

5. The complete parallel algorithm and its implementation.

We first describe the overall parallel algorithm and then an efficient parallel implementation on an EREW PRAM.

5.1. The complete parallel algorithm.

We are ready to present the complete parallel algorithm for the biconnectivity augmentation problem.

```
graph function par_bca(graph $G$);
{* The input graph $G$ has at least three vertices; $l$ is the number of leaves in the block graph $T$. *}
    set of pairs of vertices $L$; tree $T$; vertex $u$, $v$, $\alpha$, $\beta$; set of edges $S$;
    $T$ := blk($G$);
    if $T$ is a forest $\rightarrow$ perform the procedure specified in Section 4.1 fi;
    if $T$ is not balanced $\rightarrow$ perform the procedure specified in Section 4.2 fi;
    do $l \geq 2 \rightarrow$
        if $l > 3 \rightarrow$ if $a_1 > \frac{l}{4} \rightarrow L$ := case1($T$) | $a_1 \leq \frac{l}{4} \rightarrow L$ := case2($T$) fi
        | $l \leq 3 \rightarrow$ let $u$ and $v$ be two leaves in $T$; $L := \{(u, v)\}$
        fi;
        $S := \{\}$;
        pfor each $(u, v) \in L \rightarrow$
            find a noncutpoint vertex $\alpha$ in the corresponding block of $G$ represented by $u$;
            find a noncutpoint vertex $\beta$ in the corresponding block of $G$ represented by $v$;
            add an edge between $\alpha$ and $\beta$; $S := S \cup \{(u, v)\}$
        rofp;
1.      $T$ := par_update($T$, $S$) {* The procedure par_update returns the updated block
        tree after adding the set of edges in $S$. *}
    od;
    return $G$
end par_bca;
```

The correctness of algorithm par_bca follows from the correctness of the various cases (Corollary 4.11, Corollary 4.17, and Corollary 4.22) we established earlier.

In the previous sections we showed details of each step in algorithm par_bca except step 1. We now describe an algorithm for updating the block tree, given the original block tree T and the set of edges S added to it (step 1 in algorithm par_bca).

To describe the parallel algorithm for updating the block graph T after adding a set of edges S, we define the following equivalence relation $\mathcal{R}$ on the set of b-vertices B, where $B = \{v \mid v \text{ is a } b\text{-vertex in } T \text{ and } v \text{ is in a cycle created by adding the edges in } S\}$. A pair (x, y) is in $\mathcal{R}$ if and only if $x \in B$, $y \in B$ and vertices in blocks represented by x and y are in the same block after the edges in S are added. It is obvious that $\mathcal{R}$ is reflexive, symmetric, and transitive. Since $\mathcal{R}$ is an equivalence relation, we can partition B into k disjoint subsets B_i, $1 \leq i \leq k$, such that for each i, $x, y \in B_i$ implies $(x, y) \in \mathcal{R}$ and for any $(x, y) \in \mathcal{R}$, x and y both belong to the same B_i.

The following claim can easily be verified by using Fact 3.6 and the above definition for the equivalence relation on the set of b-vertices.

CLAIM 5.1. *Two b-vertices b_1 and b_2 are in the same equivalence class if and only if there exists a set of fundamental cycles $\{C_0, \ldots, C_q\}$ such that $b_1 \in C_0$, $b_2 \in C_q$, and C_i and C_{i+1} share a common b-vertex for* $0 \le i < q$. □

Notice that fundamental cycles in the block tree created by adding edges between pairs of leaves found in phase 1 and phase 2 of case 1 and subcase 2.1 share a common *b*-vertex (the root). Any pair of fundamental cycles created by adding edges between pairs of leaves found in phase 3 of case 1 either share a child of v_1 (a *b*-vertex) or do not share any *b*-vertex at all. Fundamental cycles created by adding edges between pairs of leaves found in phase 4 of case 1 do not share any *b*-vertex with any other fundamental cycle. Any pair of fundamental cycles created by adding edges between pairs of leaves found in subcase 2.2 share either the root (a *b*-vertex) or a *b*-vertex created by the merge operation (Claim 4.19).

From the above discussion we know that *b*-vertices in fundamental cycles formed by adding edges due to phase 1 and phase 2 of case 1 shrink into a single *b*-vertex in the new block tree. The *b*-vertices in fundamental cycles formed by adding edges due to phase 3 of case 1 that share a common child of v_1 shrink into a single *b*-vertex. The *b*-vertices in a fundamental cycle formed by adding edges due to phase 4 of case 1 shrink into a single *b*-vertex. The *b*-vertices in all fundamental cycles formed by adding edges due to subcase 2.1 or subcase 2.2 shrink into a single *b*-vertex. Thus we know how to compute the equivalence classes of $\mathcal{R}$.

We now describe the algorithm for updating the block tree given the original block tree T and the set of edges S added to it.

```
tree function par_update(tree $T$, set of edges $S$);
    vertex $w$; integer $k$; set of edges $S_1$, $S_2$, $S_3$, $S_4$;
    let $B$ be the set of $b$-vertices in a cycle in $T \cup S$;
    {* The partition $\{B_i \mid 1 \le i \le k\}$ of $B$ is computed such that two $b$-vertices $b_1$ and
    $b_2$ are in the same set if and only if there exists a set of fundamental cycles
    $\{C_0, \ldots, C_q\}$ in $T \cup S$ with $b_1 \in C_0$, $b_2 \in C_q$, and $C_i$ and $C_{i+1}$ share a common $b$-vertex
    $\forall i, 0 \le i < q$. *}
    if $S$ is constructed from pairs found in case 1 $\rightarrow$
        let $S_i$ $\forall i$, $1 \le i \le 4$, be the edges in $S$ corresponding to the pairs found in phase $i$;
        let $B_1$ be the set of $b$-vertices in fundamental cycles in $T \cup S_1 \cup S_2$;
        pfor the $i$th child $z_i$ of $v_1$ $\rightarrow$
            let $B_{i+1}$ be the set of $b$-vertices in fundamental cycles in $T \cup S_3$ that contain $z_i$
        rofp;
        $k$ := 1 + the number of children of $v_1$ in $T$;
        pfor the $i$th edge $e_i$ in $S_4$ $\rightarrow$
            let $B_{i+k}$ be the set of $b$-vertices in the fundamental cycle in $T \cup \{e_i\}$
        rofp;
        $k := k + |S_4|$
    | $S$ is constructed from the pairs found in case 2 $\rightarrow$
        $B_1 := B$; $k := 1$
    fi;
    pfor $i = 1 .. k$ $\rightarrow$
        collapse all $b$-vertices in $B_i$ into a single $b$-vertex
    rofp;
    eliminate parallel edges created by collapsing $b$-vertices;
    let $T'$ be this graph;
    pfor each $c$-vertex $w$ in $T'$ $\rightarrow$ if degree($w$) = 1 $\rightarrow$ eliminate $w$ fi rofp;
    return $T'$
end par_update;
```

CLAIM 5.2. *Function* par_update *returns the updated block tree.*

Proof. The proof is from Claim 5.1 and parts (3) and (4) in Fact 3.6. □

Note that we can get the updated block tree by using an algorithm for finding biconnected components. We will, however, show in §5.2 that the time needed on an EREW PRAM for updating the block tree by using function par_update is less than that needed to compute connected components using with a linear number of processors. Hence we do not want to use the straightforward algorithm for finding connected components to implement function par_update.

5.2. The parallel implementation. We now describe an efficient parallel implementation for algorithm par_bca.

Given an undirected graph, we can find its block graph in time $O(\log^2 n)$ with a linear number of processors on an EREW PRAM by using the parallel algorithm in Tarjan and Vishkin [23] for finding biconnected components and by using some procedures in Nath and Maheshwari [16].

The parallel versions of stage 1 and stage 2 are described in §4.1 and §4.2, respectively. In stage 3 the children-permutation procedure can be accomplished in time $O(\log n)$ with a linear number of processors on an EREW PRAM by calling the parallel merge sort routine in Cole [2] and by using the Euler tour technique in Tarjan and Vishkin [23] to restructure and normalize the tree. To perform functions case1, case2, and par_update we need the following procedures.

- A procedure that numbers leaves in the tree from left to right or from right to left.
- For each vertex v in a tree, a procedure that finds the number and the set of leaves in the subtree rooted at v.
- For a vertex v in a tree, a procedure that finds the leftmost leaf of each subtree rooted at a child of v.
- For a tree T with a set of edges S added between leaves in T, a procedure that computes
  - the number of cycles that pass through a vertex in $T \cup S$,
  - the set of vertices in a cycle in $T \cup S$.

All of these procedures can be done in $O(\log n)$ time using a linear number of processors on an EREW PRAM by using the Euler technique in Tarjan and Vishkin [23] and procedures in Schieber and Vishkin [19].

From Corollary 4.11, Corollary 4.17, and Corollary 4.22 we know that algorithm par_bca removes at least a quarter of the leaves in the current block graph during each execution of the **do** loop. Initially, the number of leaves is at most n. Hence the main **do** loop in algorithm par_bca is executed $O(\log n)$ times. Each iteration takes $O(\log n)$ time with a linear number of processors since the parallel sorting routine used in permuting children needs $O(n)$ processors. This establishes the following claim.

CLAIM 5.3. *The biconnectivity augmentation problem on an undirected graph can be solved in time $O(\log^2 n)$ with a linear number of processors on an* EREW PRAM, *where n is the number vertices in the input graph.*

6. Conclusion. In this paper we have presented a linear time sequential algorithm and an efficient parallel algorithm to find a smallest augmentation to biconnect a graph. Our sequential algorithm corrects an error in an earlier algorithm proposed for this problem in Rosenthal and Goldner [18]. Our parallel algorithm is new, and it runs in $O(\log^2 n)$ time by using a linear number of processors on an EREW PRAM. Although the parallel algorithm follows the overall structure of our sequential algorithm, the parallelization of some of the steps required new insights into the problem. Our parallel algorithm can be made to run within the same time bound by using a sublinear number of processors through the use of the algorithm for finding connected components in [3] and the algorithm for integer sorting in [9].

REFERENCES

[1] G.-R. Cai and Y.-G. Sun, *The minimum augmentation of any graph to a k-edge-connected graph*, Networks, 19 (1989), pp. 151–172.

[2] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.

[3] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1986, pp. 478–491.

[4] K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653–665.

[5] A. Frank, *Augmenting graphs to meet edge-connectivity requirements*, in Proc. 31th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1990, pp. 708–718.

[6] H. Frank and W. Chou, *Connectivity considerations in the design of survivable networks*, IEEE Trans. Circuit Theory, CT-17 (1970), pp. 486–490.

[7] G. N. Frederickson and J. Ja'Ja', *Approximation algorithms for several graph augmentation problems*, SIAM J. Comput., 10 (1981), pp. 270–283.

[8] D. Gusfield, *Optimal mixed graph augmentation*, SIAM J. Comput., 16 (1987), pp. 599–612.

[9] T. Hagerup, *Towards optimal parallel bucket sorting*, Inform. and Comput., 75 (1987), pp. 39–51.

[10] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[11] T.-S. Hsu and V. Ramachandran, *A linear time algorithm for triconnectivity augmentation*, in Proc. 32th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1991, pp. 548–559.

[12] S. P. Jain and K. Gopal, *On network augmentation*, IEEE Trans. Reliability, R-35 (1986), pp. 541–543.

[13] Y. Kajitani and S. Ueno, *The minimum augmentation of a directed tree to a k-edge-connected directed graph*, Networks, 16 (1986), pp. 181–197.

[14] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 869–941.

[15] D. Naor, D. Gusfield, and C. Martel, *A fast algorithm for optimally increasing the edge-connectivity*, in Proc. 31th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1990, pp. 698–707.

[16] D. Nath and N. Maheshwari, *Parallel algorithms for the connected components and minimal spanning tree problems*, Inform. Process. Lett., 14 (1982), pp. 7–11.

[17] V. Ramachandran, *Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity*, in Synthesis of Parallel Algorithms, J. H. Reif, ed., Morgan-Kaufmann, San Mateo, CA, 1993, pp. 275–340.

[18] A. Rosenthal and A. Goldner, *Smallest augmentations to biconnect a graph*, SIAM J. Comput., 6 (1977), pp. 55–66.

[19] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, in Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science 319, Springer-Verlag, Berlin, 1988, pp. 111–123.

[20] D. Soroker, *Fast parallel strong orientation of mixed graphs and related augmentation problems*, J. Algorithms, 9 (1988), pp. 205–223.

[21] K. Steiglitz, P. Weiner, and D. J. Kleitman, *The design of minimum-cost survivable networks*, IEEE Trans. Circuit Theory, CT-16 (1969), pp. 455–460.

[22] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

[23] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.

[24] S. Ueno, Y. Kajitani, and H. Wada, *Minimum augmentation of a tree to a k-edge-connected graph*, Networks, 18 (1988), pp. 19–25.

[25] T. Watanabe, *An efficient way for edge-connectivity augmentation*, Tech. Report ACT-76-UILU-ENG-87-2221, Coordinated Science Laboratory, University of Illinois, Urbana, IL, 1987.

[26] T. WATANABE AND A. NAKAMURA, *On a smallest augmentation to triconnect a graph*, Tech. Report C-18, Department of Applied Mathematics, Faculty of Engineering, Hiroshima University, Higashi-Hiroshima, Japan, 1983; revised 1987.

[27] ———, *Edge-connectivity augmentation problems*, J. Comput. System Sci., 35 (1987), pp. 96–144.

[28] ———, 3-*connectivity augmentation problems*, in Proc. 1988 IEEE International Symposium on Circuits and Systems, Institute of Electrical and Electronics Engineers, New York, 1988, pp. 1847–1850.

[29] T. WATANABE, T. NARITA, AND A. NAKAMURA, 3-*edge-connectivity augmentation problems*, in Proc. 1989 IEEE International Symposium on Circuits and Systems, Institute of Electrical and Electronics Engineers, New York, 1989, pp. 335–338.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 913–934, October 1993

DIFFERENT MODES OF COMMUNICATION*

BERND HALSTENBERG† AND RÜDIGER REISCHUK†

Abstract. This paper compares the communication complexity of discrete functions under different modes of computation, unifying and extending several known models. Protocols can be deterministic, nondeterministic, or probabilistic. Furthermore, in the last case the error probability may vary. On the other hand, communication can be one-way, two-way, or as an intermediate stage can consist of a fixed number $k > 1$ of rounds.

The following main results are obtained. A square gap between deterministic and nondeterministic communication complexity is shown for a specific function, which is the maximum possible. This improves the results of K. Mehlhorn and E. M. Schmidt in [*Proc.* 14th *Annual ACM Symposium on Theory of Computing*, 1982, pp. 330–337] and of A. V. Aho, J. D. Ullman, and M. Yannakakis in [*Proc.* 15th *Annual ACM Symposium on Theory of Computing*, 1983, pp. 133–139]. For probabilistic one-way and two-way protocols linear lower bounds are proved for functions that satisfy certain independence conditions, extending the results of A. C. Yao in [*Proc.* 11th *Annual ACM Symposium on Theory of Computing*, 1979, pp. 209–213], and in [*Proc.* 24th *Annual IEEE Symposium on Foundations of Computer Science*, 1983, pp. 420–428]. Further, with more technical effort an exponential gap between deterministic k-round and probabilistic $(k-1)$-round communication with fixed error probability is obtained. This generalizes the main result of P. Duris, Z. Galil, and G. Schnitger [*Inform. and Comput.*, 73 (1987), pp. 1–22]. In contrast, for arbitrary error probabilities less than 1/2 there is no difference between the complexity of one-way and two-way protocols, which extends the results of R. Paturi and J. Simon [*J. Comput. System Sci.*, 33 (1986), pp. 106–123]. Finally, communication with fixed message length and uniform probability distributions is considered, and simulations of arbitrary protocols by such uniform distributions with little overhead are provided.

Key words. communication complexity, k-round protocols, probabilistic protocols

AMS subject classifications. 68Q10, 68Q15, 68Q22, 68Q30, 94A05

1. Introduction.

In this paper we compare different ways in which a discrete function $f : X_0 \times X_1 \to Y$ can be computed in a distributive way. The model was originally defined by Yao [Y79]. Two processors, P_0 and P_1, get inputs $x_0 \in X_0$ and $x_1 \in X_1$, respectively. In order to compute f, they alternate in exchanging messages according to some (deterministic, nondeterministic, or probabilistic) algorithm A; finally, one of them computes the result depending only on its input and the messages that have been exchanged.

Without loss of generality we require the messages exchanged during a computation of A to be strings $m_1, \dots, m_r \in \{0,1\}^*$, where r is the number of *rounds* of that computation. The messages m_i with odd (even) i are those sent by $P_0\,(P_1)$. We require the algorithm A to generate self-delimiting messages, i.e., for any two computations $(m_1, \dots, m_r)$ and $(m'_1, \dots, m'_r)$ (possibly padded with empty strings to have the same number of rounds) and every $i \in \{1, \dots, r\}$, the concatenation $m_1 \cdots m_i$ must not be a proper prefix of $m'_1 \cdots m'_i$.

A deterministic algorithm, also called a *protocol*, can be specified by two *transmission functions* $\phi_i : X_i \times \{0,1\}^* \to \{0,1\}$ and the partial *output functions* $a_i : X_i \times \{0,1\}^* \to Y$, $i \in \{0,1\}$. For $\hat{k} := k \bmod 2$, $\phi_{\hat{k}}(x_{\hat{k}}, w_1 \cdots w_k)$ is the message sent by processor $P_{\hat{k}}$ if the messages $w_1, \dots, w_k$ have already been exchanged. If, for some $k \geq 0$, $a_{\hat{k}}(x_{\hat{k}}, w_1 \cdots w_k)$ is defined, then this value specifies the result of the computation. To guarantee prefix freeness, in some cases one additional round may be needed to inform $P_{\widehat{k+1}}$ about termination.

For a deterministic protocol A let $l(A)$ denote its *length*, or *communication complexity*, that is, the maximal number of bits exchanged during a computation of A for any pair of inputs. A computes the function f if for all $(x_0, x_1) \in X_0 \times X_1$ the computation of A on (x_0, x_1) is

*Received by the editors November 28, 1988; accepted for publication (in revised form) May 27, 1992. A preliminary version of these results was presented at the 20th Annual ACM Symposium on Theory of Computing (see [HR 88]). This research was supported by Deutsche Forschungsgemeinschaft grant RE 672/1-1.

†Institut für Theoretische Informatik, Technische Hochschule Darmstadt, Alexanderstraße 10, 6100 Darmstadt, Germany (reischuk@iti.informatik.th-darmstadt.de).

finite and the result equals $f(x_0, x_1)$. Then the *deterministic communication complexity* of f is defined as

$$\mathrm{C}_{\mathrm{det}}(f) := \min\{l(A) \mid A \text{ computes } f\}.$$

To define probabilistic protocols Yao used transmission functions $\phi_i : X_i \times \{0,1\}^* \times \{0,1\}^* \to [0,1]$ and output functions $a_i : X_i \times \{0,1\}^* \times Y \to [0,1]$ denoting probability distributions on the set of possible messages, resp., the set of possible decision values [Y79], [Y83]. $\phi_{\hat{k}}(x_{\hat{k}}, w_1 \cdots w_k, w)$ is the probability that message w will be sent by $P_{\hat{k}}$ if the messages $w_1, \cdots, w_k$ have already been exchanged. $a_{\hat{k}}(x_{\hat{k}}, w_1 \ldots w_k, y)$ (if defined) is the probability that the computation will stop with result y after exchanging messages $w_1, \ldots, w_k$. Then the probability that the result on input (x_0, x_1) equals y is

$$\sum_{k\in\mathrm{N}} \sum_{\substack{w_1, \ldots, w_k \in \{0,1\}^* : \\ a_{\hat{k}}(x_{\hat{k}}, w_1 \cdots w_k, y) \text{ is defined}}} a_{\hat{k}}(x_{\hat{k}}, w_1 \cdots w_k, y) \cdot \prod_{j=0}^{k-1} \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}).$$

Note that, unlike the usual assumption for probabilistic computations in the sequential case, the alternatives for continuing a protocol are not necessarily given by a uniform distribution. In this model a powerful random source is added for free. We will discuss this property in more detail in §6.

For a probabilistic algorithm A let $l(A)$ and $\bar{l}(A)$ denote the maximal (respectively, expected) length of the computations of A on input (x_0, x_1), maximized over all inputs $(x_0, x_1) \in X_0 \times X_1$. $\epsilon(A, f)$ (or $\epsilon(A)$ if f is fixed) denotes the error probability of A with respect to f for the worst input, that is,

$$\epsilon(A, f) := \max_{(x_0, x_1)} \Pr\big[A \text{ on } (x_0, x_1) \text{ yields some } y \neq f(x_0, x_1)\big].$$

Then, for $0 < \epsilon < \frac{1}{2}$ we call

$$\mathrm{C}_\epsilon(f) := \min\{l(A) | A \text{ is a probabilistic algorithm with } \epsilon(A, f) < \epsilon\},$$
$$\bar{\mathrm{C}}_\epsilon(f) := \inf\{\bar{l}(A) | A \text{ is a probabilistic algorithm with } \epsilon(A, f) < \epsilon\}$$

the *worst case*, respectively, *expected probabilistic communication complexity of f for error probability less than ϵ.*

Nondeterministic protocols for f also may have different alternatives for continuing a computation. A computation path does not have to generate a result, but for any input (x_0, x_1) at least one computation path must exist that ends with a result. If this happens, the result must equal $f(x_0, x_1)$. For a nondeterministic algorithm A let $l'(A)$ be the minimum length of all computations with output maximized over all inputs $(x_0, x_1) \in X_0 \times X_1$. The *nondeterministic communication complexity of f* is defined as

$$\mathrm{C}_{\mathrm{ndet}}(f) := \min\{l'(A) | A \text{ is a nondeterministic algorithm for } f\}.$$

For Boolean-valued functions f we also consider nondeterministic algorithms A accepting the language defined by f, $L(f) := f^{-1}(\{1\})$. There exists an accepting computation (output 1) of A on input (x_0, x_1) if and only if $(x_0, x_1) \in L(f)$. Let $l''(A)$ denote the minimum length of such an accepting computation, maximized over all inputs $(x_0, x_1) \in L(f)$. We call

$$\mathrm{C}_{\mathrm{L\,ndet}}(f) := \min\{l''(A) | A \text{ accepts } L(f)\}$$

the *nondeterministic communication complexity of the language defined by f*. The complexity of its complement is denoted by

$$\mathrm{C}_{\mathrm{coL\,ndet}}(f) := \mathrm{C}_{\mathrm{L\,ndet}}(\bar{f}),$$

where $\bar{f} := 1 - f$ is the complement function of f.

For all the models defined above we also consider the *k-round communication complexities*, i.e., the complexity of algorithms that use at most k rounds of communication. In this case the notation $\mathrm{C}_{k,\mathrm{det}}$, $\mathrm{C}_{k,\epsilon}$, etc., will be used

As for ordinary sequential computations with probabilistic choices, one can show that for fixed error $\epsilon < \frac{1}{2}$ worst-case and expected communication complexities are of the same order. Therefore, we restrict the discussion to the worst-case probabilistic complexity.

LEMMA 1. *For all $k \in \mathbf{N}$ and fixed $\epsilon \in (0, \frac{1}{2})$, $\mathrm{C}_{k,\epsilon}(f) = \Theta(\bar{\mathrm{C}}_{k,\epsilon}(f))$.*

Proof. Clearly, $\bar{\mathrm{C}}_{k,\epsilon}(f) \leq \mathrm{C}_{k,\epsilon}(f)$. On the other hand, let A be a k-round algorithm that computes f with error probability less than ϵ. Construct an algorithm A' by executing three independent computations of A in parallel and by deciding on a most frequent result. Then $\bar{l}(A') \leq 3 \cdot (\bar{l}(A) + \lceil \log |Y| \rceil)$ and $\epsilon(A') < 3\epsilon^2 - 2\epsilon^3 =: \epsilon' < \epsilon$ if $\epsilon < \frac{1}{2}$. Let $a := 1/(\epsilon - \epsilon')$. Obtain algorithm A'' by simulating A' for some bounded number of communication steps. Computations exchanging more than $a \cdot \bar{l}(A)$ bits are ceased; this happens with probability at most $\frac{1}{a}$. Hence the error probability of A'' is less than $\epsilon' + \frac{1}{a} = \epsilon$ and

$$l(A'') \leq \frac{3 \cdot (\bar{l}(A) + \lceil \log |Y| \rceil)}{\epsilon - 3\epsilon^2 + 2\epsilon^3}. \qquad \square$$

2. Deterministic and nondeterministic communication complexity. For our purposes a function $f : X_0 \times X_1 \to Y$ may be regarded as a $|X_0| \times |X_1|$ matrix F with entries from Y. Both processors know F, and on input (x_0, x_1) they have to look up the entry in row x_0 and column x_1. For $y \in Y$, a *y-rectangle* is defined to be a subset $A \times B \subseteq X_0 \times X_1$ such that $f(A \times B) = \{y\}$ (including the trivial case $A \times B = \emptyset$). By $\delta_y(F)$ we denote the minimum number of y-rectangles needed to cover $f^{-1}(\{y\})$, and by $\mathrm{d}_y(F)$ we denote the minimum size of a set of disjoint y-rectangles that cover $f^{-1}(\{y\})$. The sums of these quantities over all $y \in Y$ are denoted by $\delta(F)$ and $\mathrm{d}(F)$, respectively.

It is not hard to see [Y79], [AUY83] that for any $y \in Y$

$$\mathrm{C}_{\mathrm{det}}(f) \geq \lceil \log \mathrm{d}_y(F) \rceil.$$

For Boolean-valued functions, Mehlhorn and Schmidt [MS82] showed an equivalent lower bound based on the rank of the matrix over the natural numbers $\mathbf{N}$, $\mathrm{rank}_{\mathbf{N}}(F) = \mathrm{d}_1(F)$. Since $\mathrm{rank}_{\mathbf{N}}(F) \geq \mathrm{rank}_K(F)$ for any field K, the deterministic two-way complexity of a Boolean-valued function is bounded from below by $\lceil \log \mathrm{rank}_K(F) \rceil$ for any field K.

The nondeterministic complexity of a function f is quite well determined by

$$\lceil \log \delta(F) \rceil \geq \mathrm{C}_{\mathrm{ndet}}(f) \geq \lceil \log \delta_y(F) \rceil.$$

Moreover, for $Y = \{0, 1\}$ we have

$$\lceil \log(\delta_1(F) + 1) \rceil \geq \mathrm{C}_{\mathrm{L\,ndet}}(f) \geq \lceil \log \delta_1(F) \rceil.$$

Aho, Ullman, and Yannakakis [AUY83] showed that $\mathrm{C}_{\mathrm{det}}(f) \leq O(\log^2 \delta(F))$; hence $\mathrm{C}_{\mathrm{det}}(f) \leq O(\log^2 \mathrm{d}(F))$. On the other hand, no function with deterministic complexity greater than $O(\log \mathrm{d}(F))$ seems to be known. Thus there is a quadratic gap between the lower and upper bounds. For Boolean-valued functions, the following theorem improves the upper bound shown in [AUY83].

THEOREM 1. *Let F be the matrix corresponding to $f : X_0 \times X_1 \to \{0, 1\}$. Then*

$$\mathrm{C_{det}}(f) \leq \lceil \log \delta_0(F) \rceil \, (\lceil \log(\delta_1(F) + 1) \rceil + 2) + 1,$$

that is,

$$\mathrm{C_{det}}(f) \leq \mathrm{C_{L\,ndet}}(f) \cdot \mathrm{C_{coL\,ndet}}(f) \cdot (1 + o(1)).$$

Proof. Since 0-rectangles do not intersect with 1-rectangles, for any 1-rectangle u there exist at most $\frac{1}{2}\delta_0(F)$ 0-rectangles that have a row in common with u or there exist at most $\frac{1}{2}\delta_0(F)$ 0-rectangles that have a column in common with u. Therefore, one of the two processors can halve the number of 0-rectangles that have still to be considered by specifying an appropriate u that occurs in its row (respectively, column). The computation stops with result 1 if u also occurs in the column (respectively, row) of the other processor or if no 0-rectangles are left. In the remaining case there is no appropriate u for either of the two processors, and the result must be 0. □

Although for some functions this bound is smaller than the one given in [AUY83] (e.g., the identification function has $\delta_1(I_{2^n}) = 2^n$ and $\delta_0(I_{2^n}) \leq 2n$), a square difference between $\mathrm{C_{det}}(f)$ and $\log \delta_0(F) \cdot \log \delta_1(F)$ is still possible (e.g., for the n-bit ordering function $f(x, y) = 1$ if and only if $x \leq y$, which has $\delta_1(F) = 2^n$ and $\delta_0(F) = 2^n - 1$). On the other hand, the upper-bound $\log \delta_0(F) \cdot \log \delta_1(F) \approx \mathrm{C_{L\,ndet}}(f) \cdot \mathrm{C_{coL\,ndet}}(f)$ cannot be improved by more than some constant factor.

Fürer showed in [F87] an $O(\sqrt{n})$ upper bound on the Las Vegas communication complexity of some function f with deterministic complexity $\mathrm{C_{det}}(f) = n$. (The Las Vegas communication complexity of f is the expected number of bits exchanged for the worst-input computation of an optimal error-free probabilistic algorithm computing f.) Since the Las Vegas complexity of f is an upper bound on both $\mathrm{C_{L\,ndet}}(f)$ and $\mathrm{C_{coL\,ndet}}(f)$, the bound of the above theorem may be tight up to a constant factor. The following theorem shows this property also for functions f that do not satisfy $\mathrm{C_{L\,ndet}}(f) \approx \mathrm{C_{coL\,ndet}}(f)$.

THEOREM 2. *Let $a, b \in \mathbf{N}$, and let $\mu := \log a / \log(ab)$. There exists a function $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$ with*

(i) $\mathrm{C_{L\,ndet}}(f) \leq O(n^{\mu})$ *and* $\mathrm{C_{coL\,ndet}}(f) \leq O(n^{1-\mu})$,

(ii) $\mathrm{C_{det}}(f) = n$.

Proof. For any sequence of functions $g_i : X_{i,0} \times X_{i,1} \to \{0, 1\}$ the functions $g_i \sqcup g_j$, $g_i \sqcap g_j : (X_{i,0} \times X_{j,0}) \times (X_{i,1} \times X_{j,1}) \to \{0, 1\}$ are defined by

$$\begin{aligned}(g_i \sqcup g_j)((x_0, x_0'), (x_1, x_1')) &:= g_i(x_0, x_1) \vee g_j(x_0', x_1'),\\ (g_i \sqcap g_j)((x_0, x_0'), (x_1, x_1')) &:= g_i(x_0, x_1) \wedge g_j(x_0', x_1').\end{aligned}$$

For fixed $g = g_i$ and $m \in \mathbf{N}$ let $g^{\sqcup m}$ and $g^{\sqcap m}$ be defined, respectively, by

$$\begin{aligned} g^{\sqcup 1} &:= g, & g^{\sqcup i+1} &:= g^{\sqcup i} \sqcup g,\\ g^{\sqcap 1} &:= g, & g^{\sqcap i+1} &:= g^{\sqcap i} \sqcap g.\end{aligned}$$

For $n = a^{\lfloor t/2 \rfloor} \cdot b^{\lceil t/2 \rceil}$ let $q_t : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$ be defined by

$$q_0(x, y) := (x \wedge y) \vee (\bar{x} \wedge \bar{y}), \quad q_{2s+1} := q_{2s}^{\sqcup b}, \quad q_{2s+2} := q_{2s+1}^{\sqcap a}.$$

Then, for any even t it is easy to see that

$$\mathrm{C}_{\mathrm{L\,ndet}}(q_t) \leq a \cdot (\lceil \log b \rceil + \mathrm{C}_{\mathrm{L\,ndet}}(q_{t-2})),$$
$$\mathrm{C}_{\mathrm{coL\,ndet}}(q_t) \leq \lceil \log a \rceil + b \cdot \mathrm{C}_{\mathrm{coL\,ndet}}(q_{t-2}).$$

Hence

$$\mathrm{C}_{\mathrm{L\,ndet}}(q_t) \leq O(a^{t/2}) \leq O(n^{\mu}),$$
$$\mathrm{C}_{\mathrm{coL\,ndet}}(q_t) \leq O(b^{t/2}) \leq O(n^{1-\mu}). \qquad \square$$

For the lower bound on $\mathrm{C}_{\mathrm{det}}(f)$ we need the following lemma (see [T87, Thm. 13]).

LEMMA 2. *Let f and g be two Boolean-valued functions with corresponding matrices F and G, respectively. Let $F \sqcap G$ and $F \sqcup G$ be the matrices corresponding to $f \sqcap g$ and $f \sqcup g$, respectively. Then*

$$\mathrm{rank}_{\mathrm{GF}(2)}(F \sqcap G) \geq \mathrm{rank}_{\mathrm{GF}(2)}(F) \cdot \mathrm{rank}_{\mathrm{GF}(2)}(G),$$
$$\mathrm{rank}_{\mathrm{GF}(2)}(F \sqcup G) \geq (\mathrm{rank}_{\mathrm{GF}(2)}(F) - 1) \cdot (\mathrm{rank}_{\mathrm{GF}(2)}(G) - 1) - 1.$$

For the matrix Q_t corresponding to q_t, Lemma 2 implies

$$\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s+2}) \geq (\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s+1}))^a,$$
$$\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s+1}) \geq (\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s}) - 1)^b - 1.$$

This would suffice for showing that $\mathrm{C}_{\mathrm{det}}(q_t) = \Omega(n)$. For the exact bound we have to show that Q_t and $\bar{Q}_t$ have the same rank over GF(2). It can be easily seen that the number of 1's and the number of 0's in any column of Q_t is odd for every t. So, if we add the last $2^n - 1$ rows to the first row, it will contain only 1's. Now, adding the first row to all other rows changes 1's into 0's and vice versa. $\bar{Q}_t$ can then be obtained by adding again the last $2^n - 1$ rows to the first row (since $2^n - 1$ is odd). Therefore, Q_t and $\bar{Q}_t$ have the same rank over GF(2) and it follows that

$$\begin{aligned}\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s+1}) &= \mathrm{rank}_{\mathrm{GF}(2)}(\bar{Q}_{2s+1}) \\ &= \mathrm{rank}_{\mathrm{GF}(2)}(\bar{Q}_{2s}^{\sqcap b}) \\ &= \left(\mathrm{rank}_{\mathrm{GF}(2)}(\bar{Q}_{2s})\right)^b \\ &= \left(\mathrm{rank}_{\mathrm{GF}(2)}(Q_{2s})\right)^b.\end{aligned}$$

Hence, for even t, $\mathrm{rank}_{\mathrm{GF}(2)}(Q_t) = 2^{(ab)^{t/2}}$ and the lower bound in [MS82] implies that

$$\mathrm{C}_{\mathrm{det}}(q_t) \geq \log \mathrm{rank}_{\mathrm{GF}(2)}(Q_t) = (ab)^{t/2} = n.$$

For $(ab)^{t/2-1} < n \leq (ab)^{t/2}$, define f by its corresponding matrix F, which can be chosen as an arbitrary $2^n \times 2^n$ submatrix of Q_t of maximal rank.

COROLLARY 1. *For $f : X_0 \times X_1 \to Y$ and $p \in \mathbf{N}$ define $f^p : X_0^p \times X_1^p \to Y^p$ by*

$$f^p((x_{0,1}, \ldots, x_{0,p}), (x_{1,1}, \ldots, x_{1,p})) := (f(x_{0,1}, x_{1,1}), \ldots, f(x_{0,p}, x_{1,p})).$$

Then, for almost all functions $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$

$$\mathrm{C}_{\mathrm{det}}(f^p) = p\,(\mathrm{C}_{\mathrm{det}}(f) - o(1)).$$

Proof. Clearly, $\mathrm{C}_{\mathrm{det}}(f^p) \leq p \cdot \mathrm{C}_{\mathrm{det}}(f)$. Since for any $\epsilon > 0$, $\mathrm{rank}_{\mathrm{GF}(2)}(F) \geq 2^n - \epsilon n$ for almost all $2^n \times 2^n$ matrices F over $\{0, 1\}$ (see [T87]), it follows from Lemma 2 that

$$\begin{aligned}\mathrm{C}_{\mathrm{det}}(f^p) &\geq \mathrm{C}_{\mathrm{det}}(f \sqcap \cdots \sqcap f)\\ &\geq \log \mathrm{rank}_{\mathrm{GF}(2)}(F \sqcap \cdots \sqcap F)\\ &\geq p \cdot \log(2^n - \epsilon n)\\ &= p \cdot (n + \log(1 - \epsilon n \cdot 2^{-n}))\\ &\geq p \cdot (\mathrm{C}_{\mathrm{det}}(f) - o(1))\end{aligned}$$

for almost all $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$. □

3. Probabilistic communication complexity. For any two functions $f, g : X_0 \times X_1 \to Y$ let

$$\Delta(f, g) := \left|\{(x_0, x_1) \mid (x_0, x_1) \in X_0 \times X_1, f(x_0, x_1) \neq g(x_0, x_1)\}\right|$$

be the number of arguments for which f and g differ and

$$\mathrm{D}_{k,\epsilon}(f) := \min\{\mathrm{C}_{k,\mathrm{det}}(g) | \Delta(f, g) < \epsilon\, |X_0 \times X_1|\}.$$

$\mathrm{D}_{k,\epsilon}(f)$ is called the *distributional ϵ-error complexity* [Y83]. The following relation between random and distributional complexity follows easily:

LEMMA 3. $\mathrm{C}_{k,\epsilon}(f) \geq \mathrm{D}_{k,\epsilon}(f)$.

On the basis of Lemma 3 we get the following analogue to the bound on deterministic one-way complexity in the probabilistic case.

THEOREM 3. *For any function $f : X_0 \times X_1 \to \{0, 1\}$ with corresponding matrix F and all ϵ with $0 < \epsilon \leq \frac{1}{8}$, the one-way complexity of f for error probability less than ϵ can be bounded by*

$$\mathrm{C}_{1,\epsilon}(f) \geq \log \mathrm{nrow}(F) - \lceil 4\epsilon\, \mathrm{ncol}(F)\rceil \log \frac{\mathrm{e}}{4\epsilon} - \log\lceil 4\epsilon\, \mathrm{ncol}(F)\rceil - 2.$$

COROLLARY 2. *If* $\mathrm{ncol}(F) \leq c \cdot \log\, \mathrm{nrow}(F)$ (*and hence* $c \geq 1$), *then,* $\mathrm{C}_{1,\epsilon}(f) \geq 0.95 \log \mathrm{nrow}(F) - 3$ *for* $\epsilon = \frac{1}{(1024c \log(2c))}$.

COROLLARY 3. *The disjointness function $d_n : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$, defined by*

$$d_n((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = 0 \quad \text{iff} \quad x_i = y_i = 1 \textit{ for some } i \in \{1, \ldots, n\},$$

has probabilistic one-way complexity $\mathrm{C}_{1,\epsilon}(d_n) = \Theta(n)$.

We prove Theorem 3 with the help of two lemmata.

DEFINITION. Let $d_H(u, v)$ denote the Hamming distance between vectors u and v. For any $m \times n$ matrix F over $\{0, 1\}$ and $\epsilon > 0$ define

$$R_\epsilon(F) := \max\{r \mid \exists\, r \text{ distinct rows } u_1, \ldots, u_r \text{ in } F \text{ and } v \in \{0, 1\}^n : \sum_i d_H(u_i, v) < \epsilon r n\}.$$

LEMMA 4. *For any function $f : X_0 \times X_1 \to \{0, 1\}$ and all $\epsilon' > \epsilon > 0$*

$$\mathrm{D}_{1,\epsilon}(f) \geq \log \frac{|X_0|\, (1 - \frac{\epsilon}{\epsilon'})}{R_{\epsilon'}(F)} - 1.$$

Proof. Consider a deterministic one-way algorithm A. Without loss of generality we assume that for each input processor P_1 computes the result. This can be achieved at the expense of at most one additional bit of communication. The messages of A induce a partition of $X_0 \times X_1$ into disjoint rectangles $Z_1 \times X_1, \dots, Z_m \times X_1$. Within such a rectangle, all rows of the computed matrix are identical.

Suppose that A computes some function $g : X_0 \times X_1 \to \{0, 1\}$ with $\Delta(f, g) < \epsilon\, |X_0 \times X_1|$. Let ϵ_i be the fraction of differences between f and g in rectangle $Z_i \times X_1$. Let $I := \{i \,|\, \epsilon_i < \epsilon'\}$, and let $Z = \bigcup_{i \in I} Z_i$. Then

$$\epsilon\, |X_0| \;\geq\; \sum_{i \notin I} \epsilon_i\, |Z_i| \;\geq\; \epsilon' \left(|X_0| - |Z|\right),$$

which implies

$$|Z| \;\geq\; \left(1 - \frac{\epsilon}{\epsilon'}\right) |X_0|.$$

Because $|Z_i| \leq R_{\epsilon'}(F)$ for all $i \in I$, it follows that

$$m \geq |I| \geq \frac{|Z|}{R_{\epsilon'}(F)}$$

and hence

$$\mathrm{D}_{1,\epsilon}(f) + 1 \;\geq\; \log \frac{|X_0|\,(1 - \frac{\epsilon}{\epsilon'})}{R_{\epsilon'}(F)}. \qquad \square$$

LEMMA 5. *Let F be a 0–1-matrix of distinct rows such that each column of F has multiplicity at least α and at most β. Then, for* $0 < \epsilon \leq \frac{\alpha}{4\beta}$

$$R_\epsilon(F) < \lceil \gamma \,\mathrm{ncol}(F) \rceil\, 2^{\lceil \gamma \,\mathrm{ncol}(F) \rceil \log(e/\gamma)}, \quad \textit{where} \quad \gamma := \frac{2\epsilon\beta}{\alpha} \leq \frac{1}{2}.$$

Proof. Let $c := \mathrm{ncol}(F)$. Determine $a, b \geq 0$ by the equation

$$R_\epsilon(F) = \sum_{i=0}^{a} \binom{c}{i} + b$$

with maximal a. Then the number of differences between any $R_\epsilon(F) \times |X_1|$ submatrix of F and an arbitrary matrix of the same size consisting of identical rows is at least

$$\alpha \cdot \left(\sum_{i=0}^{a} i \cdot \binom{c}{i} + (a+1) \cdot b \right).$$

From the definition of $R_\epsilon(F)$ it follows that

$$\epsilon \;>\; \frac{\alpha \cdot \left(\sum_{i=0}^{a} i \cdot \binom{c}{i} + (a+1) \cdot b \right)}{\beta c R_\epsilon(F)} \;=\; \frac{\alpha}{\beta c} \cdot \frac{\sum_{i=0}^{a} i \cdot \binom{c}{i} + (a+1) \cdot b}{\sum_{i=0}^{a} \binom{c}{i} + b} \;\geq\; \frac{a\alpha}{2\beta c}.$$

Therefore, $a+1 \leq \lceil 2\epsilon\beta c/\alpha \rceil = \lceil \gamma c \rceil \leq \lceil c/2 \rceil$ and

$$R_\epsilon(F) < \sum_{i=0}^{a+1} \binom{c}{i} \leq \lceil \gamma c \rceil \cdot \binom{c}{\lceil \gamma c \rceil} \leq \lceil \gamma c \rceil \cdot 2^{\lceil \gamma c \rceil \log(e/\gamma)}. \qquad \square$$

Proof of Theorem 3. Without loss of generality we may assume that $|X_0| = \text{nrow}(F)$ and $|X_1| = \text{ncol}(F)$. Because $C_{1,\epsilon}(f) \geq D_{1,\epsilon}(f)$, the theorem follows immediately from the two lemmata above with $\epsilon' = 2\epsilon$ and $\alpha = \beta = 1$. $\square$

For the probabilistic two-way complexity, lower bounds can be obtained by the following mended and extended version of a theorem in [Y83]. Yao's claim would imply that the function $f : \{1, \dots, 2m\} \times \{1, \dots, 2m\} \to \{0, 1\}$, $f(i, j) = 1$ if and only if $i \leq m$, had probabilistic complexity $C_\epsilon(f) = \Theta(n)$.

THEOREM 4. *Let* $A \subset Y$, $0 < \mu < 1 - \mu' < 1$, $c \geq 0$, *and* $0 < \lambda \leq 1$. *Let* $f : X_0 \times X_1 \to Y$ *be a function, and let* $S_A(x_0) = \{x_1 \in X_1 | f(x_0, x_1) \in A\}$. *If, for every* $G \subseteq X_0$ *with* $|G| > |X_0|^{1-\lambda}$, f *satisfies*

(i) $|f^{-1}(A)| \geq \mu \cdot |X_0 \times X_1|$,

(ii) $|f^{-1}(A) \cap G \times X_1| \leq (1 - \mu') \cdot |G \times X_1|$,

(iii)

$$\sum_{x_0, x_0' \in G : x_0 \neq x_0'} |S_A(x_0) \cap S_A(x_0')| \leq \sum_{x_0, x_0' \in G : x_0 \neq x_0'} |S_A(x_0)| \cdot |S_A(x_0')| \cdot \frac{1 + c \cdot |X_1|^{-\lambda}}{|X_1|},$$

then for any ϵ *with* $0 < \epsilon < \frac{\mu\mu'}{1+\mu'}$ *the following inequality holds:*

$$C_\epsilon(f) \geq \lambda \cdot \log \min\left\{ |X_0|,\ |X_1| \left(\alpha\beta + \beta \cdot \frac{|X_1|^\lambda}{|X_0|} \right)^{-1/\lambda} \right\} + \log\left(\mu - \epsilon - \frac{\epsilon}{\epsilon'} \right).$$

Here ϵ' *satisfying* $\frac{\epsilon}{\mu - \epsilon} < \epsilon' < \mu'$ *may be chosen arbitrarily,* $\alpha := 4c(1 - \mu')^2$, *and* $\beta := \frac{9}{16} \frac{\mu' + 2\epsilon'}{(\mu' - \epsilon')^3}$.

We omit the proof since it is similar to the proof in [Y83]. What Yao really proved there is the following:

COROLLARY 4. *Let* μ, μ', c, λ, *and* ϵ *as above be fixed constants, let* $|X_0| = |X_1| =: N$, *and let* $n := \lceil \log N \rceil$. *If for* $f : X_0 \times X_1 \to Y$ *and for some* $A \subset Y$

(i) $|f^{-1}(A)| \geq \mu N^2$,

(ii) $|S_A(x_0)| \leq (1 - \mu') \cdot N \quad \forall x_0 \in X_0$,

(iii) $|S_A(x_0) \cap S_A(x_0')| \leq |S_A(x_0)| \cdot |S_A(x_0')| \cdot \frac{1 + cN^{-\lambda}}{N} \quad \forall x_0 \neq x_0' \in X_0$, *then*

$$\lambda n - O(1) \leq C_\epsilon(f) \leq n,$$

i.e.,

$$C_\epsilon(f) = \Theta(n).$$

This leads to the following simple result solving a problem raised in [Y83] (see also [KS92] and the references cited there).

COROLLARY 5. *For any* $\epsilon < \frac{1}{6}$ *the set intersection function* $h_n : \{0, 1\}^n \times \{0, 1\}^n \to \{0, \dots, n\}$ *defined by*

$$h_n((x_1, \dots, x_n), (y_1, \dots, y_n)) = x_1 y_1 + \cdots + x_n y_n$$

has probabilistic complexity $C_\epsilon(h_n) \geq n - O(1)$.

Proof. The last bit of $h_n((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ is the inner product mod 2 of $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$. It is easy to see that the inner product mod 2 satisfies the conditions of Theorem 4 with $\lambda = 1$. □

4. k-Round communication complexity. Duris, Galil, and Schnitger exhibited a hierarchy between k-round, $k = 1, 2, \ldots$, protocols [DGS87]. For each k they showed the existence of functions with an exponential difference between their $(k+1)$- and k-round complexity. An open problem mentioned in [DGS87] is to prove a similar result in the probabilistic case. The following theorem shows that such exponential gaps exist even between deterministic $(k+1)$-round and probabilistic k-round complexity.

For any function f let $\tilde{f}$ denote the function derived from f by interchanging the arguments, that is,

$$\tilde{f}(x_1, x_0) = f(x_0, x_1).$$

For the corresponding matrix, $\tilde{F}$ is just the transposition of F.

THEOREM 5. *For all* $k \geq 2$ *and* $0 < \epsilon < \frac{1}{2}$ *there exists a family of functions* $f_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ *such that*

(i) $C_{k,\mathrm{det}}(f_n) \leq O(k \cdot \log n)$,

(ii) $\min\{C_{k-1,\epsilon}(f_n), C_{k-1,\epsilon}(\tilde{f}_n)\} \geq \Omega(n/k)$.

This theorem follows from the following.

LEMMA 6. *For every odd k there exists a family of functions* $f_n : X_0 \times X_1 \to \{0,1\}$, $|X_0|, |X_1| \leq 2^n$ *with*

(i) $C_{k,\mathrm{det}}(f_n) \leq 5k \cdot \log n$,

(ii) $D_{k,\epsilon'}(\tilde{f}_n) \geq \frac{n}{9}$, *where* $\epsilon' = \frac{23}{622080}\, 98304^{-(k-1)/2} \geq 2^{-O(k)}$.

Proof of Theorem 5. From Lemma 6 it follows that for odd k there exists a function $f_{k,n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (by padding the corresponding matrix with 0's up to size $2^n \times 2^n$) such that

$$C_{k,\mathrm{det}}(f_{k,n}) \leq 5k \cdot \log n$$

and

$$C_{k,\epsilon}(\tilde{f}_{k,n}) \geq \frac{C_{k,\epsilon'}(\tilde{f}_{k,n})}{2a+1} - 1 \geq \frac{D_{k,\epsilon'}(\tilde{f}_{k,n})}{2a+1} - 1,$$

where

$$a = \left\lceil \frac{\log(\epsilon'/\epsilon)}{\log(4\epsilon(1-\epsilon))} \right\rceil \leq O(\log \tfrac{1}{\epsilon'}) \leq O(k).$$

The first inequality is due to the fact that by performing $2a+1$ repetitions the error probability can be reduced by a factor $\exp(-a)$ (see [H86]). Hence

$$C_{k,\epsilon}(\tilde{f}_{k,n}) \geq \frac{n}{18 \cdot \left\lceil \frac{\log(\epsilon'/\epsilon)}{\log(4\epsilon(1-\epsilon))} \right\rceil + 9} - 1 \geq \Omega\left(\frac{n}{k}\right).$$

Since $C_{k,\epsilon}(\tilde{f}_{k,n}) \leq C_{k-1,\epsilon}(\tilde{f}_{k,n})$ and $C_{k,\epsilon}(\tilde{f}_{k,n}) \leq C_{k-1,\epsilon}(f_{k,n})$, this proves the theorem for odd k. For even k let $F_{k-1,n-1}$ be the matrix corresponding to $f_{k-1,n-1}$. Define $f_{k,n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ by its matrix

$$F_{k,n} := \begin{pmatrix} \mathbf{0} & \tilde{F}_{k-1,n-1} \\ F_{k-1,n-1} & \mathbf{0} \end{pmatrix}.$$

Now it is easy to see that

$$C_{k,\mathrm{det}}(f_{k,n}) \leq 2 + C_{k-1,\mathrm{det}}(f_{k-1,n-1}) \leq O(k \cdot \log n),$$

and

$$C_{k-1,\epsilon}(\tilde{f}_{k,n}) = C_{k-1,\epsilon}(f_{k,n}) \geq C_{k-1,\epsilon}(\tilde{f}_{k-1,n-1}) \geq \Omega\left(\frac{n}{k}\right). \qquad \square$$

COROLLARY 6. *For all $k \geq 2$ there exists a family of functions $f_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ with*

(i) $C_{k,\mathrm{det}}(f_n) \leq 5k \cdot \log n + 2$,

(ii) $\min\{C_{k-1,\mathrm{det}}(f_n), C_{k-1,\mathrm{det}}(\tilde{f}_n)\} \geq \frac{n-1}{9}$.

Proof. The corollary can be proved analogously to Theorem 5 by observing that $C_{k,\mathrm{det}}(f_n) \geq D_{k,\epsilon}(f_n)$ for any $\epsilon > 0$. $\square$

Proof of Lemma 6. It suffices to consider only n with $\frac{n}{20 \log n} \geq k$. Let $n \geq n_0$ (n_0 will be specified implicitely later), let $k = 2t + 1 \leq \frac{n}{20 \log n}$, and choose $l \in \mathbf{N}$ maximal such that

$$\left\lceil \frac{\log l}{\gamma} \right\rceil l \leq n,$$

where γ is a suitable constant less than $\frac{1}{2}$. $\gamma = \frac{1}{9}$ is sufficient for our purpose here. Define

$$b := 2^{\lceil \log l / \gamma \rceil} \quad \text{and} \quad m := b^l.$$

Then $m \leq 2^n$ and $l - t = \Theta(\frac{n}{\log n})$. Furthermore, let

$$r_s := \quad b^{l-t+s} \quad \text{for } 0 \leq s \leq t,$$

$$\beta_s := \begin{cases} 2 & \text{for } s = 0, \\ b & \text{for } 1 \leq s \leq t, \end{cases}$$

$$a_s := \begin{cases} (l-t)\log b & \text{for } s = 0, \\ (l-t+s) & \text{for } 1 \leq s \leq t, \end{cases}$$

$$g_s := \begin{cases} \lfloor b^{l-t}/a_0 \rfloor & \text{for } s = -1, \\ a_s \cdot g_{s-1} & \text{for } 0 \leq s \leq t. \end{cases}$$

Then

$$\left(\frac{b}{2}\right)^{\gamma} < l \leq b^{\gamma} \quad \text{and} \quad g_t \leq b^{l-t} \prod_{s=1}^{t} (l-t+s) \leq 2^{\lceil \log l/\gamma \rceil l} b^{-t} l^t \leq 2^n.$$

For $s = 0, \ldots, t$ we define $r_s \times a_s$ matrices A_s and $r_s \times g_s$ matrices G_s. For $1 \leq i \leq r_s$ the ith row of A_s is the β_s-adic representation of $i - 1$, i.e., the binary representation for $s = 0$ and the b-adic representation for $s \geq 1$. The matrices G_s are constructed recursively as follows. We obtain G_0 from A_0 by replacing each column j by the $r_0 \times g_{-1}$ matrix consisting of g_{-1} copies of column j. The set of these columns of G_0 is called the jth *column block* of G_0.

For $0 \leq s < t$, G_{s+1} is built from A_{s+1} and G_s. For any column j, $1 \leq j \leq a_{s+1}$, of A_{s+1} and each digit $d \in \{0, \ldots, b-1\}$ the d-entries in that column form a $r_s \times 1$ submatrix (see Fig. 1). Replace each such submatrix by the $r_s \times g_s$ matrix $\pi_{d,s}(G_s)$, where $\pi_{d,s}$ is a random column permutation, which will be specified later. Doing this for all columns j of

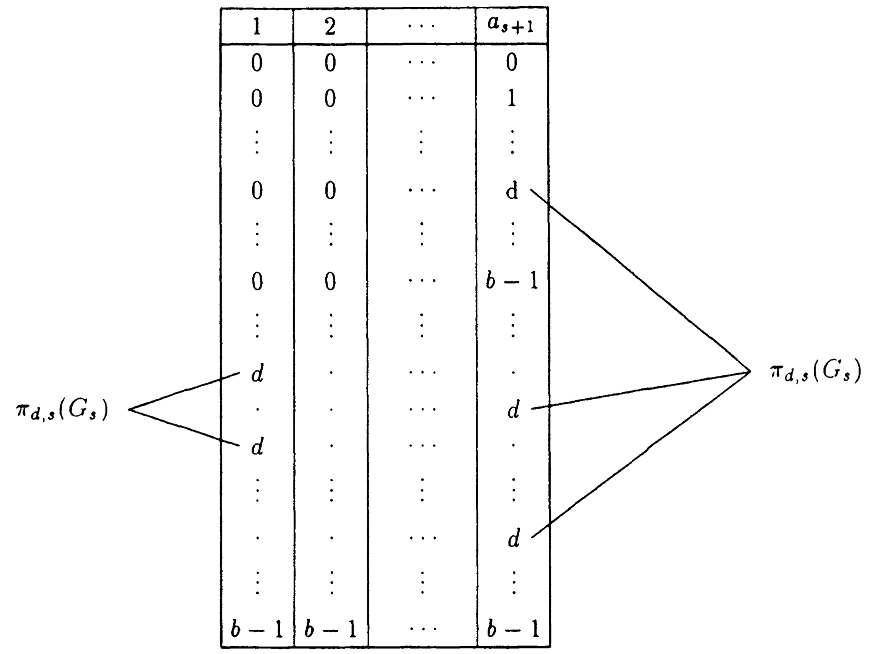


FIG. 1. *Inserting G_s into A_{s+1}.*

A_{s+1}, one obtains G_{s+1}. The set of the g_s columns of G_{s+1} resulting from column j of A_{s+1} is the jth column block of G_{s+1}. If B_j is the jth column block of G_s, then the jth column block of $\pi_{d,s}(G_s)$ is given by the permuted columns in B_j.

Let f be the function corresponding to $\tilde{G}_t$. Then the first part of Lemma 6 can be seen as follows. For $s \in \{1, \ldots, t\}$ the computation of $\tilde{G}_s$ in $2s + 1$ rounds can be reduced to the computation of $\tilde{G}_{s-1}$ in $2s - 1$ rounds: Processor P_0 sends the number j $(< a_s \leq l)$ of the column block of G_s to which its input belongs, and P_1, in reply, sends the number d $(< b)$ of the permutation $\pi_{d,s-1}$ belonging to its input in this column block. Then only $\pi_{d,s-1}(\tilde{G}_{s-1})$ remains to be computed, which is equivalent to computing $\tilde{G}_{s-1}$ because P_0 knows the permutation $\pi_{d,s-1}$. Finally, after $2t$ rounds, $\tilde{G}_0$ is left. But this is equivalent to computing $\tilde{A}_0$. P_0 sends the number j of the column of its input and P_1 can determine the result. For $\gamma = \frac{1}{9}$ and n large enough such that $\lceil \log l/\gamma \rceil \geq 2$ we have

$$\log l \leq \log\left(n/\left\lceil \frac{\log l}{\gamma} \right\rceil\right) \leq \log n - 1,$$

$$\log a_0 = \log((l-t)\log b) \leq \log\log m \leq \log n.$$

Therefore, the communication for f can be bounded by

$$\mathrm{C}_{k,\mathrm{det}}(f) \leq t \cdot (\log l + \log b) + \log a_0 \leq \frac{k-1}{2} \cdot \left(\log l + \left\lceil \frac{\log l}{\gamma} \right\rceil\right) + \log n \leq 5k \cdot \log n.$$

For the lower-bound proof we need the following two lemmata and a corollary extending the arguments in [DGS87].

LEMMA 7. *Let the parameters $\alpha, \mu, \nu \in \mathbf{R}^+$ and $b, \lambda, r \in \mathbf{N}$ satisfy*
(1) $0 < \alpha < \mu < 1$,
(2) $\nu = \frac{\mu-\alpha}{1+\gamma-\alpha} - \frac{1}{\lambda}$,
(3) $b^{1-\alpha} \geq 2$,
(4) $\lambda \leq b^{\gamma}$,
(5) $b^{\mu\lambda} \leq r$.

Let $A = (a_{i,j})$ be an $r \times \lambda$ matrix over $\{0, \dots, b-1\}$ with distinct rows. Then there are at least $\frac{\nu}{4} \cdot \lambda$ columns $A^{(j)}$ of A such that, for arbitrary sets $U_j \subset \{0, \dots, b-1\}$ of size $|U_j| \leq b^{\alpha}$, there exist sets $D_j(U_j) \subseteq \{0, \dots, b-1\} - U_j$ with the following properties:
(i) $|\{i \mid 1 \leq i \leq r,\ a_{i,j} = d\}| \geq \frac{\nu}{8} \cdot \frac{r}{b}\ \forall d \in D_j(U_j)$,
(ii) $|\{i \mid 1 \leq i \leq r,\ a_{i,j} \in D_j(U_j)\}| \geq \frac{\nu}{8} \cdot r$.

Proof. Assume first that A contains at least $\frac{\nu}{4}\lambda$ columns $A^{(j)}$ with the following property: (*) for an arbitrary set U_j of size at most b^{α} $A^{(j)}$ has at least $\frac{\nu}{4}r$ entries $a_{i,j} \notin U_j$.

Given sets U_j, let $D_j(U_j)$ consist of all values $d \notin U_j$ that occur at least $\frac{\nu}{8b}r$ times in $A^{(j)}$ to satisfy property (i). Since the average frequency of elements in $\{0, \dots, b-1\} - U_j$ in $A^{(j)}$ is larger than $\frac{\nu}{4b}r$, at least half of the elements of $A^{(j)}$ not contained in U_j belong to $D_j(U_j)$. By assumption this number is at least $\frac{\nu}{8}r$ and thus proves (ii).

If (*) does not hold for at least $\frac{\nu}{4}\lambda$ columns of A, then for each $A^{(j)}$ choose a set $U_j \subset \{0, \dots, b-1\}$ of size $|U_j| = \lfloor b^{\alpha} \rfloor$ such that $A^{(j)}$ contains a minimal number of elements not in U_j. The total number of entries $a_{i,j} \notin U_j$ of A—call this number w—can then be bounded by

$$w < \frac{\nu}{4}\lambda r + \left(\lambda - \frac{\nu}{4}\lambda\right)\frac{\nu}{4}r < \frac{\nu}{2}\lambda r.$$

We will show that this inequality cannot be satisfied. Define $u, v \in \mathbf{N}$ by

$$r = \sum_{q=0}^{u} \binom{\lambda}{q} b^{\alpha(\lambda-q)} (b - b^{\alpha})^{q} + v$$

with $v \geq 0$ and maximal u. This expression corresponds to a matrix $\bar{A}$ of r distinct rows such that the number of entries $a_{i,j}$ with $a_{i,j} \in U_j$ is maximized. There can be at most $\binom{\lambda}{q} b^{\alpha(\lambda-q)} (b-b^{\alpha})^{q}$ different sequences of length λ containing q elements that do not belong to the corresponding U_j. Then from the expression above one deduces easily that

$$w \geq \sum_{q=0}^{u} q \cdot \binom{\lambda}{q} b^{\alpha(\lambda-q)} (b - b^{\alpha})^{q} + (u+1) \cdot v.$$

Now let $q \leq \frac{u}{2}$. Condition (3) implies $b - b^{\alpha} \geq b^{\alpha}$, and clearly $\frac{\lambda}{2} - q \geq |\frac{\lambda}{2} - (u-q)|$. Hence one can conclude

$$\binom{\lambda}{q} b^{\alpha(\lambda-q)} (b - b^{\alpha})^{q} \leq \binom{\lambda}{u-q} b^{\alpha(\lambda-(u-q))} (b - b^{\alpha})^{u-q}.$$

This implies

$$\begin{aligned} q\binom{\lambda}{q} b^{\alpha(\lambda-q)} (b-b^{\alpha})^{q} + (u-q)\binom{\lambda}{u-q} b^{\alpha(\lambda-(u-q))} (b-b^{\alpha})^{u-q} \\ \geq \frac{u}{2}\left[\binom{\lambda}{q} b^{\alpha(\lambda-q)} (b-b^{\alpha})^{q} + \binom{\lambda}{u-q} b^{\alpha(\lambda-(u-q))} (b-b^{\alpha})^{u-q}\right] \end{aligned}$$

and

$$w \geq \frac{u}{2} r.$$

On the other hand,

$$r < \sum_{q=0}^{u+1} \binom{\lambda}{q} b^{\alpha(\lambda-q)} (b - b^{\alpha})^q \leq \lambda^{u+1} b^{\alpha\lambda} \cdot b^{(1-\alpha)(u+1)}$$

or, by solving this for u with (4) and (5),

$$u > \frac{\log r - \log b^{\alpha\lambda}}{\log b^{1-\alpha} + \log \lambda} - 1 \geq \frac{(\mu - \alpha) \cdot \lambda \cdot \log b}{(1 + \gamma - \alpha) \cdot \log b} - 1 = \frac{\mu - \alpha}{1 + \gamma - \alpha} \cdot \lambda - 1 = \nu\lambda.$$

Hence $w \geq \frac{\nu}{2} \cdot \lambda r$, which contradicts the bound on w from above. □

LEMMA 8. *Let $\lambda, v, c \in \mathbf{N}$, and let a set B be partitioned into λ subsets $B_1 \dot\cup \cdots \dot\cup B_\lambda$, each of size v. Let $p(u)$ denote the probability that u randomly chosen permutations $\pi_1, \ldots, \pi_u$ of B do not have the following property:*
$(*)$ $\forall\, C \subseteq B$, $|C| \geq c \exists$ permutation $\pi_d \in \{\pi_1, \ldots, \pi_u\}$

$$\forall\, 1 \leq j \leq \lambda : \frac{1}{12} \frac{|C|}{\lambda} \leq |\pi_d(B_j) \cap C| \leq 3 \frac{|C|}{\lambda}.$$

Then

$$p(u) \leq \exp\left(-c \cdot \left(\frac{u}{4\lambda} - \log(\lambda v)\right) + \log(\lambda v) + u \cdot (3 + \log \lambda)\right).$$

Proof. The proof is similar to that of Lemma 4 in [DGS87]. For our purpose here we need a somewhat better estimate and also an upper bound on the size of $\pi_d(B_j) \cap C$. For $C \subseteq B$, $|C| = c$, let $n_{C,j}(a)$ be the number of permutations π of B with $|\pi(B_j) \cap C| = a$. Then

$$n_{C,j}(a) = \binom{c}{a} \cdot \binom{\lambda v - c}{v - a} \cdot v! \cdot (\lambda v - v)!,$$

and hence the probability to choose such a permutation is given by

$$P_{C,j}(A) = \binom{c}{a} \cdot \binom{\lambda v - c}{v - a} \Big/ \binom{\lambda v}{v}.$$

Since $P_{C,j}(\lfloor \frac{c}{3\lambda} \rfloor) \leq 1$, for all $a \leq \lfloor \frac{c}{3\lambda} \rfloor$ the following inequality holds:

$$P_{C,j}(a) \leq \frac{P_{C,j}(a)}{P_{C,j}(\lfloor \frac{c}{3\lambda} \rfloor)} = \frac{\lfloor \frac{c}{3\lambda} \rfloor! \cdot (c - \lfloor \frac{c}{3\lambda} \rfloor)! \cdot (v - \lfloor \frac{c}{3\lambda} \rfloor)! \cdot (\lambda v - c - v + \lfloor \frac{c}{3\lambda} \rfloor)!}{a! \cdot (c - a)! \cdot (v - a)! \cdot (\lambda v - c - v + a)!}$$

$$\leq \left(\left\lfloor \frac{c}{3\lambda} \right\rfloor \cdot \frac{1}{c - \lfloor \frac{c}{3\lambda} \rfloor} \cdot \frac{1}{v - \lfloor \frac{c}{3\lambda} \rfloor} \cdot \left(\lambda v - c - v + \left\lfloor \frac{c}{3\lambda} \right\rfloor\right)\right)^{\lfloor \frac{c}{3\lambda} \rfloor - a}$$

$$\leq \left(\frac{\lambda v - c - v + \frac{c}{3\lambda}}{3\lambda v - c - v + \frac{c}{3\lambda}}\right)^{\lfloor \frac{c}{3\lambda} \rfloor - a} \leq 2^{-(\lfloor \frac{c}{3\lambda} \rfloor - a)}.$$

Analogously, for all $a > \frac{3c}{\lambda}$

$$P_{C,j}(a) \leq \left(\frac{1}{a - \lceil \frac{c}{\lambda} \rceil + 1} \cdot \left(c - \left\lceil \frac{c}{\lambda} \right\rceil\right) \cdot \left(v - \left\lceil \frac{c}{\lambda} \right\rceil\right) \cdot \frac{1}{\lambda v - c - v + \lceil \frac{c}{\lambda} \rceil}\right)^{a - \lceil c/\lambda \rceil} \leq 2^{-(a - \lceil c/\lambda \rceil)}.$$

Hence the probability of choosing some permutation π that does not satisfy

$$3\frac{c}{\lambda} \geq |\pi(B_j) \cap C| \geq \frac{c}{12\lambda}$$

is at most

$$\begin{aligned} P_{C,j} &\leq \sum_{a=0}^{\lceil \frac{c}{12\lambda} \rceil - 1} 2^a \cdot 2^{-\lfloor \frac{c}{3\lambda} \rfloor} + \sum_{a=\lceil \frac{3c}{\lambda} \rceil}^{v} 2^{-a} \cdot 2^{\lceil \frac{c}{\lambda} \rceil} \\ &\leq 2^{\frac{c}{12\lambda} + 1 - \frac{c}{3\lambda} + 1} + 2^{-\frac{3c}{\lambda} + 1 + \frac{c}{\lambda} + 1} \leq 2^{\frac{-c}{4\lambda} + 3}. \end{aligned}$$

Hence the probability that u randomly chosen permutations $\pi_1, \dots, \pi_u$ of B do not satisfy ($\star$) is

$$\begin{aligned} p(u) &\leq \sum_{c=c_0}^{\lambda v} \sum_{\substack{C \subset B \\ |C| = c}} \prod_{d=1}^{u} \sum_{j=1}^{\lambda} 2^{\frac{-c}{4\lambda} + 3} = \sum_{c=c_0}^{\lambda v} \sum_{\substack{C \subset B \\ |C| = c}} 2^{\frac{-cu}{4\lambda} + u(3 + \log \lambda)} \leq \sum_{c=c_0}^{\lambda v} 2^{-c(\frac{u}{4\lambda} - \log(\lambda v)) + u(3 + \log \lambda)} \\ &\leq 2^{-c_0 \cdot \left(\frac{u}{4\lambda} - \log(\lambda v)\right) + \log(\lambda v) + u \cdot (3 + \log \lambda)} \end{aligned}$$

for $u \geq 4\lambda \log(\lambda v)$. For $u < 4\lambda \log(\lambda v)$ the claim is trivial. □

COROLLARY 7. *Let $s \in \{0, \dots, t-1\}$, $c \geq a_s^2$, $\alpha > 2\gamma$, and m be sufficiently large. Then there exist b column permutations $\pi_{0,s}, \dots, \pi_{b-1,s}$ of G_s such that for any set C of at least c columns of the matrix*

$$\begin{pmatrix} \pi_{0,s}(G_s) \\ \vdots \\ \pi_{b-1,s}(G_s) \end{pmatrix}$$

there are at most b^α digits $d \in \{0, \dots, b-1\}$ for which not every column block of $\pi_{d,s}(G_s)$ has at least $\frac{1}{12}|C|/a_s$ and at most $3|C|/a_s$ columns in common with C.

Proof. Let B be the set of columns of G_s, and let B_j, $1 \leq j \leq a_s$, be the set of columns of the jth column block of G_s. Lemma 8 implies that for a random choice of $\pi_{0,s}, \dots, \pi_{b-1,s}$ the probability that there exists a set $U \subset \{0, \dots, b-1\}$ of size b^α and a set $C \subseteq B$ of size at least c such that for each $d \in U$ there exists a $j \in \{1, \dots, a_s\}$ not satisfying $\frac{1}{12}|C|/a_s \leq |\pi_{d,s}(B_j) \cap C| \leq 3|C|/a_s$ is bounded by

$$p \leq \binom{b}{b^\alpha} \cdot p(b^\alpha) \leq \exp\left(b^\alpha \log b - {a_s}^2 \cdot \left(\frac{b^\alpha}{4a_s} - \log g_s\right) + \log g_s + b^\alpha \cdot (3 + \log a_s)\right).$$

Now, by assumption

$$\log g_s \leq \log b^l \leq b^\gamma \log b \leq o\left(\frac{b^\alpha}{4a_s}\right) \quad \text{and} \quad b^\alpha \log b \leq o(b^\alpha \cdot a_s).$$

Therefore, $p \leq \exp(-\Omega(b^\alpha \cdot a_s))$, that is, $p < 1$ for sufficiently large m. □

We further need the following technical lemma.

LEMMA 9. *Let a set B be partitioned into $B = B_1 \dot\cup \cdots \dot\cup B_N$, and let to each B_i be assigned a binary string w_i such that the set $\{w_1, \ldots, w_N\}$ is prefix-free. Then the union of subsets B_i satisfying $|B_i| \cdot 2^{|w_i|} \geq |B|/2$ covers at least half of all elements in B.*

Proof. Define $I := \{i \in \{1, \ldots, N\} \mid \ |B_i|\, 2^{|w_i|} \geq |B|/2\}$. Then we can conclude

$$\sum_{i \in I} |B_i| = |B| - \sum_{i \notin I} |B_i| \geq |B| - \sum_{i \notin I} \frac{|B|}{2} \cdot 2^{-|w_i|} \geq \frac{|B|}{2}$$

by Kraft's inequality. □

A submatrix M of G_t is called *i-unseparated* (with regard to a deterministic algorithm) if for all inputs corresponding to M the same first i messages are exchanged. A submatrix M of G_t is called a *fragment of* G_s for $0 \leq s \leq t$ if some column permutation of M results in a submatrix of G_s. If this submatrix has at least $\frac{1}{12}\mathrm{col}(M)/a_s$ and at most $3\mathrm{col}(M)/a_s$ columns in each column block of G_s, where $\mathrm{col}(M)$ denotes the number of columns of M, then M is called a *column-moderate* fragment of G_s. Let $\mathrm{row}(M)$ be the number of rows.

LEMMA 10. *Let $1 - \frac{t}{l-t} - \delta > \mu > \alpha > 2\gamma$, and let $\nu := \frac{\mu-\alpha}{1+\gamma-\alpha} - \frac{1}{l-t}$. Consider a deterministic algorithm that computes G_t (with few errors) by exchanging at most δn bits. Then, for sufficiently large n there are at least $m \cdot g_t \cdot (\frac{\nu^2}{3 \cdot 2^9})^t$ elements of G_t that lie in $(2t)$-unseparated column-moderate fragments of G_0 with at least $\frac{m}{2^{\delta n}} \cdot (\frac{\nu}{16b})^t$ rows each.*

Proof. If M is an i-unseparated submatrix of G_t, let $\mathrm{Len}_{i,0}(M)$, $\mathrm{Len}_{i,1}(M)$ be the total length of the messages that are sent, respectively, by P_0 and P_1 in the first i rounds for inputs in M. By assumption, $\mathrm{Len}_{i,0}(M) \leq \delta n$ and $\mathrm{Len}_{i,1}(M) \leq \delta n$. Since any computation is started by P_0, we have $\mathrm{Len}_{2s+2,0}(M) = \mathrm{Len}_{2s+1,0}(M)$ and $\mathrm{Len}_{2s+1,1}(M) = \mathrm{Len}_{2s,1}(M)$.

By induction we will prove that for $0 \leq s \leq t$ at least $m \cdot g_t \cdot (\frac{\nu^2}{3 \cdot 2^9})^s$ elements of G_t lie in $(2s)$-unseparated column-moderate fragments M of G_{t-s} satisfying the following: (•) $\mathrm{row}(M) \cdot 2^{\mathrm{Len}_{2s,0}(M)} \geq m \cdot (\frac{\nu}{16b})^s$ and $\mathrm{col}(M) \cdot 2^{\mathrm{Len}_{2s,1}(M)} \geq \frac{g_{t-s}}{24^s}$, which for $s = t$ yields the lemma.

For $s = 0$ the claim obviously holds. Now assume that the claim holds for $s \in \{0, \ldots, t-1\}$, and let M be a $(2s)$-unseparated column-moderate fragment of G_{t-s} that satisfies (•). Assume that, for inputs in M, P_0 may send w different messages in round $2s+1$ and may partition the rows of M according to these messages. Let $M_1, \ldots, M_w$ be the corresponding submatrices of M. Then, by Lemma 9 at least half of the rows of M belong to submatrices M_i with

$$\mathrm{row}(M_i) \cdot 2^{\mathrm{Len}_{2s+1,0}(M_i)} \geq \frac{1}{2} \cdot \mathrm{row}(M) \cdot 2^{\mathrm{Len}_{2s,0}(M)}.$$

Consider now such a submatrix M_i. Then

$$\begin{aligned}
\mathrm{row}(M_i) &\geq \frac{1}{2} \cdot \frac{m}{2^{\delta n}} \cdot \left(\frac{\nu}{16b}\right)^s \\
&\geq \frac{1}{2} \cdot b^{l-s-\delta(l+o(l))} \cdot \left(\frac{\nu}{16}\right)^s \\
&\geq \frac{1}{2} \cdot b^{(1-\delta-s/(l-s)) \cdot a_{t-s} - \delta \cdot o(l)} \cdot \left(\frac{\nu}{16}\right)^s \\
&\geq b^{\mu a_{t-s}} \quad \text{for sufficiently large } n.
\end{aligned}$$

Hence, by Lemma 7 there are at least $\frac{\nu}{4} \cdot a_{t-s}$ column blocks S_j of G_{t-s} with the following property: For arbitrary sets $U_j \subset \{0, \ldots, b-1\}$ of size $|U_j| \leq b^\alpha$ there exist sets $D_j(U_j) \subseteq \{0, \ldots, b-1\} - U_j$ such that for all $d \in D_j(U_j)$ the matrix $\pi_{d,t-s-1}(G_{t-s-1})$, which lies in

S_j, has at least $\frac{\nu}{8} \cdot \text{row}(M_i)/b$ rows in common with M_i, and at least $\frac{\nu}{8} \cdot \text{row}(M_i)$ rows of M_i fall into such inserted matrices $\pi_{d,t-s-1}(G_{t-s-1})$ with $d \in D_j(U_j)$. Now consider a submatrix B_j of M_i generated by such a column block S_j. Since M is a column-moderate fragment of G_{t-s}, we have

$$\frac{1}{12} \cdot \frac{\text{col}(M)}{a_{t-s}} \leq \text{col}(B_j) \leq 3\frac{\text{col}(M)}{a_{t-s}}.$$

Partition the columns of B_j according to the w' messages that P_1 sends in round $2s+2$, and let $C_1, \ldots, C_{w'}$ be the corresponding submatrices of B_j. By applying Lemma 9 once again it follows that at least half of the columns of B_j belong to submatrices C_p with

$$\text{col}(C_p) \cdot 2^{\text{Len}_{2s+2,1}(C_p)} \geq \frac{1}{2} \cdot \text{col}(B_j) \cdot 2^{\text{Len}_{2s+1,1}(B_j)} \geq \frac{g_{t-s}}{24^s \cdot 24 a_{t-s}} = \frac{g_{t-s-1}}{24^{s+1}}.$$

For these submatrices C_p

$$\text{col}(C_p) \geq \frac{g_{t-s-1}}{2^{\delta n} \cdot 24^{s+1}} \geq \frac{b^{l-t-\delta\cdot(l+o(l))}}{24^{s+1}} - o(1) \geq a_{t-s-1}{}^2 \quad \text{for sufficiently large } n.$$

By Corollary 7 there are at most b^α digits $d \in \{0, \ldots, b-1\}$ for which the submatrix $C_{p,d}$ of C_p, generated by the inserted matrix $\pi_{d,t-s-1}(G_{t-s-1})$, is not a column-moderate fragment of G_{t-s-1}. Let $U \subset \{0, \ldots, b-1\}$ be the set of these d's. Then, for $d \in \{0, \ldots, b-1\} - U$, $C_{p,d}$ is a $(2s+2)$-unseparated column-moderate fragment of G_{t-s-1} with

$$\text{col}(C_{p,d}) \cdot 2^{\text{Len}_{2s+2,1}(C_{p,d})} \geq \frac{g_{t-s-1}}{24^{s+1}}.$$

Because of the choice of B_j, there exists a set $D(U) \subseteq \{0, \ldots, b-1\} - U$ such that for all $d \in D(U)$

$$\text{row}(C_{p,d}) \geq \frac{\nu}{8} \cdot \frac{\text{row}(M_i)}{b}.$$

Hence

$$\begin{aligned}\text{row}(C_{p,d}) \cdot 2^{\text{Len}_{2s+2,0}(C_{p,d})} &\geq \frac{\nu}{8}\frac{\text{row}(M_i)}{b} 2^{\text{Len}_{2s+1,0}(M_i)} \\ &\geq \text{row}(M_i) 2^{\text{Len}_{2s,0}(M)} \frac{\nu}{16b} \\ &\geq m\left(\frac{\nu}{16b}\right)^{s+1},\end{aligned}$$

and at least $\frac{\nu}{8} \cdot \text{row}(M_i)$ rows of M_i belong to some $C_{p,d}$ with $d \in D(U)$. Calling the entries of the matrix that belong to some $C_{p,d}$ with $d \in D(U)$ *suitable*, we have at least $\frac{\nu}{8} \cdot \text{row}(M_i) \cdot \text{col}(C_p)$ suitable entries in C_p. Hence the number of suitable entries in B_j is at least $\frac{\nu}{8} \cdot \text{row}(M_i) \cdot \text{col}(B_j)/2$, and, consequently, the number of suitable entries in M_i is at least

$$\frac{\nu}{8} \cdot \text{row}(M_i) \cdot \frac{1}{2} \cdot \frac{\text{col}(M)}{12 a_{t-s}} \cdot \frac{\nu}{4} \cdot a_{t-s} = \text{row}(M_i) \cdot \text{col}(M) \cdot \frac{\nu^2}{3 \cdot 2^8}.$$

Thus the portion of suitable entries in M is at least $\nu^2/(3 \cdot 2^9)$, and, by the induction hypothesis, the number of suitable entries in G_t is at least $m \cdot g_t \cdot (\nu^2/3 \cdot 2^9)^{s+1}$. □

LEMMA 11. *Let M be a column-moderate fragment of G_0 with at least $\frac{m}{2^{\delta n}} \cdot (\frac{\nu}{16b})^t$ rows. Then, for $\epsilon'' \le \frac{3/4-\delta}{17280}$*

$$\mathrm{D}_{1,\epsilon''}(M) \ge \left(\frac{1}{4} - \delta\right) \cdot n - o(n).$$

Proof. From $t < \frac{l}{4} \le o(n)$ it follows that

$$\mathrm{row}(M) \ge \frac{m}{2^{\delta n}} \cdot \left(\frac{\nu}{16b}\right)^t \ge 2^{n-o(n)-\delta n} \cdot b^{-l/4} \cdot \left(\frac{\nu}{16}\right)^t \ge 2^{(3/4-\delta)\cdot n - o(n)}.$$

Since M is column-moderate and therefore has at least one column in each column block of G_0, all rows of M are pairwise different. So we get as bounds for the number of different columns of M, $\mathrm{ncol}(M)$,

$$n \ge \mathrm{ncol}(M) \ge \left(\frac{3}{4} - \delta\right) \cdot n - o(n).$$

Now, by definition there exists a $\mathrm{row}(M) \times c'$ submatrix B of M, where

$$c' \ge \mathrm{ncol}(M) \cdot \frac{\mathrm{col}(M)}{12a_0} \ge \mathrm{col}(M) \cdot \frac{\frac{3}{4} - \delta}{12} - o(\mathrm{col}(M))$$

such that each column of B has multiplicity at least $\frac{1}{12}\mathrm{col}(M)/a_0$ and at most $3\mathrm{col}(M)/a_0$. For $\epsilon^{(4)} = \frac{1/12}{20\cdot 3} = \frac{1}{720}$ Lemma 5 implies

$$R_{\epsilon^{(4)}}(B) < \left\lceil \frac{n}{10} \right\rceil \cdot 2^{\lceil n/10 \rceil \cdot \log(10e)},$$

and for $\epsilon^{(3)} = \frac{1}{2}\epsilon^{(4)}$ Lemma 4 implies

$$\begin{aligned}\mathrm{D}_{1,\epsilon^{(3)}}(B) &\ge \log \frac{\mathrm{row}(M)}{2R_{\epsilon^{(4)}}(B)} - 1 \\ &> \left(\frac{3}{4} - \delta\right) \cdot n - \frac{n}{10} \cdot \log(10e) - o(n) \\ &> \left(\frac{1}{4} - \delta\right) \cdot n - o(n).\end{aligned}$$

Now for $\epsilon'' \le \epsilon^{(3)} \cdot \frac{3/4-\delta}{12} = \frac{3/4-\delta}{17280}$ it follows that

$$\mathrm{D}_{1,\epsilon''}(M) > \left(\frac{1}{4} - \delta\right) \cdot n - o(n). \qquad \square$$

Finally, to finish the proof of Lemma 6, let $\delta = \frac{1}{9}$, $\mu = \frac{4}{9}$, $\alpha = \frac{1}{3}$, and $\nu = \frac{1}{8}$. Then, for sufficiently large n we have $\nu < \frac{\mu-\alpha}{1+\gamma-\alpha} - \frac{1}{l-t}$. Suppose that there exists a deterministic algorithm that computes f with a portion of error less than

$$\frac{3/4-\delta}{17280} \cdot \left(\frac{\nu^2}{3 \cdot 2^9}\right)^t = \frac{23}{622080} \cdot 98304^{-t}$$

with at most $\frac{n}{9}$ bits of communication. Then, by Lemma 10 there are at least $m \cdot g_t \cdot \left(\frac{\nu^2}{3\cdot 2^9}\right)^t$ entries of G_t in $(2t)$-unseparated column-moderate fragments of G_0 that have at least $\frac{m}{2^{\delta n}} \cdot \left(\frac{\nu}{16b}\right)^t$

rows each. At least one of these fragments has to be computed with a portion of error less than $\epsilon'' = \frac{3/4-1/9}{17280}$ in the last round. But this, by Lemma 11, requires at least $(\frac{1}{4}-\frac{1}{9}) \cdot n - o(n)$ bits, contradicting the assumption that only $\frac{n}{9}$ bits will be exchanged. □

Whereas for fixed error $\epsilon \in (0, \frac{1}{2})$ Theorem 5 gives exponential gaps between probabilistic k- and $(k-1)$-round complexity, the following theorem shows that if Boolean-valued functions are assumed, there is no such significant gap if the error probability does not have to be bounded away from $\frac{1}{2}$. This generalizes a result of Paturi and Simon [PS86], who showed that the one-way and two-way complexities of Boolean-valued functions for unbounded error probability less than $\frac{1}{2}$ differ by at most 1.

THEOREM 6. *For any function $f : X_0 \times X_1 \to \{0,1\}$*

$$\mathrm{C}_{1,\epsilon'}(f) \leq \mathrm{C}_\epsilon(f) + 1, \quad \textit{where} \quad \frac{1}{2} - \epsilon' = \left(\frac{1}{2} - \epsilon\right) \cdot 2^{-(\mathrm{C}_\epsilon(f)+1)}.$$

The idea of the proof is simply that in a one-way protocol processor P_0 may guess the whole computation and the resulting output of an optimal two-way protocol. The guess will be correct with probability $2^{-(\mathrm{C}_\epsilon(f)+1)}$. In this case the guessed output will be chosen as the result of the computation; otherwise, it will be a random bit.

5. Arbitrary partitions of input bits. Theorem 5 solves the problem given in [DGS87], although the original problem was formulated slightly differently. Duris, Galil, and Schnitger studied the following model: A partition of $\{1, \ldots, n\}$ into two disjoint sets A and B can be described by a permutation π of $\{1, \ldots, n\}$ and an integer $a \in \{0, \ldots, n\}$ such that $A = \{\pi(1), \ldots, \pi(a)\}$. A partition (π, a) is called *permissible* if $\frac{n}{3} \leq a \leq \frac{2n}{3}$.

Given a function $f : \{0,1\}^n \to Y$ and a partition (π, a), define the function $f_{\pi,a} : \{0,1\}^a \times \{0,1\}^{n-a} \to Y$ by

$$f_{\pi,a}((x_{\pi(1)}, \ldots, x_{\pi(a)}), (x_{\pi(a+1)}, \ldots, x_{\pi(n)})) := f(x_1, \ldots, x_n).$$

Now, if C_M denotes the communication complexity for fixed partition as considered here (e.g., $M = \det$ or $M = k, \epsilon$), then the corresponding communication complexity for variable partition VC_M is defined by

$$VC_M(f) := \min\left\{\mathrm{C}_M(f_{\pi,a}) \mid (\pi, a) \text{ is a permissible partition of } \{1, \ldots, n\}\right\}.$$

Although this definition of communication complexity looks quite different (and really *is* for several functions), it can be related to the communication model for fixed partition of the input bits. Let us call M a *standard model* if for all two-argument functions $f : X_0 \times X_1 \to Y$, respectively, all corresponding matrices, the following holds:

(1) $\mathrm{C}_M(A) \leq \mathrm{C}_M(B)$ if A is a submatrix of B.

(2) $\mathrm{C}_M(A) = \mathrm{C}_M(B)$ if A is obtained from B by permuting rows and columns.

(3) $\mathrm{C}_M(A) = \mathrm{C}_M(B)$ if A is obtained from B by duplicating rows or columns.

Then one can prove the following extended version of Theorem 3 in [DGS87].

THEOREM 7. *For any function $f : X_0 \times X_1 \to Y$ with $n_0 := \lceil \log |X_0| \rceil \geq 5$ and $n_1 := \lceil \log |X_1| \rceil \geq 5$, there exists a function $f' : \{0,1\}^{6n} \to Y$, where $n = \max\left\{n_0 + n_1, \left\lceil \frac{3n_0}{2} \right\rceil, \left\lceil \frac{3n_1}{2} \right\rceil\right\}$ such that for any standard model M*

$$VC_M(f') = \min\{\mathrm{C}_M(f), \mathrm{C}_M(\tilde{f})\}.$$

Proof. Without loss of generality we may assume that $n_0 \geq \frac{n_1}{2}$ and $n_1 \geq \frac{n_0}{2}$ and hence $n = n_0 + n_1$. (Otherwise, increase n_0 or n_1 by duplicating, respectively, rows or columns of the matrix corresponding to f.)

Let π_0 be the identity permutation on $\{1, \ldots, 6n\}$. The function f' is defined by the $2^{6n_0} \times 2^{6n_1}$ matrix F'. (F' corresponds to $f'_{\pi_0, 6n_0}$.) To obtain matrix F' from F duplicate each row of F at least 2^{5n_0} times and each column of F at least 2^{5n_1} times to get the correct dimensions and then apply random permutations σ and τ (defined later) to the rows and columns, respectively, of this matrix.

Because of properties (2) and (3) of standard models we have $C_M(F') = C_M(F)$ and hence, since $(\pi_0, 6n_0)$ is a permissible partition of $\{1, \ldots, 6n\}$, $VC_M(f') \leq \min\{C_M(f), C_M(\tilde{f})\}$. For the lower bound we will show that there exists a choice of σ and τ such that, for any permissible partition (π, a) of $\{1, \ldots, 6n\}$, the matrix corresponding to $f'_{\pi,a}$ contains F or $\tilde{F}$ as a permuted submatrix and hence, by property (3) of standard models, $VC_M(f') \geq \min\{C_M(f), C_M(\tilde{f})\}$. □

LEMMA 12. *Let A be a set of cardinality $|A| = 2^{6m}$, $m \geq 5$, partitioned into $A = A_1 \dot\cup \cdots \dot\cup A_p$, where $\lceil \log p \rceil = m$ and $|A_i| \geq 2^{5m}$. Let $B_1, \ldots, B_q$ be $q = \binom{6m}{2m}$ subsets of A of size $|B_j| = 2^{2m}$. Then there exists a permutation ρ of A such that for all $i \in \{1, \ldots, p\}$ and all $j \in \{1, \ldots, q\}$ the following holds:*

$$(\circ) \qquad \rho(A_i) \cap B_j \neq \emptyset.$$

Proof. Let $N_{i,j}$ be the number of permutations ρ violating ($\circ$) for some fixed i and j. Then

$$\begin{aligned} \frac{N_{i,j}}{(2^{6m})!} &\leq \binom{2^{6m} - 2^{5m}}{2^{2m}} \cdot (2^{2m})! \cdot (2^{6m} - 2^{2m})! \cdot \frac{1}{(2^{6m})!} \\ &= \frac{(2^{6m} - 2^{5m})!}{(2^{6m} - 2^{5m} - 2^{2m})!} \cdot \frac{(2^{6m} - 2^{2m})!}{(2^{6m})!} = \prod_{i=0}^{2^{2m}-1} \frac{2^{6m} - 2^{5m} - i}{2^{6m} - i} \\ &\leq (1 - 2^{-m})^{2^{2m}} \leq e^{-2^m}. \end{aligned}$$

Therefore, the fraction of permutations ρ violating ($\circ$) for any i and j is at most

$$\binom{6m}{2m} \cdot 2^m \cdot e^{-2^m} \leq e^{2m \cdot (1 + \ln 3) + m \cdot \ln 2 - 2^m} < 1$$

for $m \geq 5$. □

Now let (π, a) be a permissible partition of $\{1, \ldots, 6n\}$ into two sets $A = \{\pi(1), \ldots, \pi(a)\}$ and $B = \{\pi(a+1), \ldots, \pi(n)\}$. Since $|A|, |B| \geq 2n$ and $\frac{n}{3} \leq n_0 \leq \frac{2n}{3}$, either

$$\{1, \ldots, 6n_0\} \cap A \geq 2n_0 \quad \text{and} \quad \{6n_0 + 1, \ldots, 6n\} \cap B \geq 2n_1$$

hold or the same relations hold with A and B interchanged. We show that for the first case we can find F as a permuted submatrix in the matrix $F'_{\pi,a}$ corresponding to $f'_{\pi,a}$, and in the second case we can find $\tilde{F}$ in the same way. So let us restrict our attention to the first case.

For each processor P_i there are at least $2n_i$ bits that it owns in both cases, for the partition (π, a) as well as for the natural partition $(\pi_0, 6n_0)$. Now consider the common submatrix of $F'_{\pi,a}$ and F' that results when only these bits are varied while the other bits are fixed (e.g., all other bits are set to 0).

Let $R = \{0, \ldots, 2^{6n_0} - 1\}$ be the set of row numbers of F', and for $x_0 \in X_0$ let $\sigma(R_{x_0})$ be the numbers of rows of F' that have been obtained from row x_0 of F, where σ (to be defined) is the permutation of rows. By construction, $\sigma(R_{x_0}) \geq 2^{5n_0}$. For a subset $I \subset \{1, \ldots, 6n_0\}$ of size $2n_0$ let $S(I) := \left\{\sum_{i \in I} a_i \cdot 2^{i-1} | a_i \in \{0, 1\}\right\}$. $S(I)$ is the set of row numbers with all bits in $\{1, \ldots, 6n_0\} - I$ set to 0. Since $A \geq 2n_0$, there exists such a set $I \subseteq A$.

From Lemma 12 it follows that we can choose the permutationσ of rows such that for any such set I and any $x_0 \in X_0$, $S(I) \cap \sigma(R_{x_0}) \neq \emptyset$, that is, for any row x_0 of F one can find a representative in the common submatrix of $F'_{\pi,a}$ and F'. The same argument applies to the columns of the submatrix of $F'_{\pi,a}$ induced by these rows, and it shows the existence of an appropriate permutation τ of columns such that F is some permuted submatrix of $F'_{\pi,a}$.

6. Uniform probabilistic protocols.

6. Uniform probabilistic protocols. We now come back to the question of a suitable probabilistic model. Are probabilistic protocols with arbitrary probability distributions more powerful than those with uniform distributions? We will show that the increase in communication complexity cannot be too large if the error probability does not converge to $\frac{1}{2}$ very quickly.

A related issue is the uniformity of message length. The protocols considered so far may be of varying length. In some rounds long messages can be transmitted, whereas in others only a few bits might be sent. Such a discrepancy in message length may happen not only with respect to different rounds but also for a fixed round if the communication for all possible input pairs is considered.

Obviously, protocols in which the length of each message is fixed regardless of the input and the specific round are preferable. Let us call such protocols *fixed-length* protocols. Each protocol can easily be converted to one with message length 1 such that the total information exchange is at most doubled. But in general the price one has to pay for this is a huge increase in the number of rounds. The following construction gives a better balance.

THEOREM 8. *Let f be computed by a k-round protocol with communication T. Then there exists a $(3k-2)$-round protocol for f with fixed length $l = \lceil T/k \rceil$ with the same error probabilities.*

Proof. The processors cut their messages into blocks of length l, filling up with zeroes if necessary. As long as the transmission of all blocks of a message is not finished, the other processor, in its rounds simply answers with a block of zeroes. Because of prefix freeness this condition can easily be checked. If the original protocol is probabilistic, one also has to assign probabilities to each block. It is easy to see how this can be done by taking the sum of the probabilities of messages with identical blocks and by using conditional probabilities for the following blocks.

It remains to estimate the number of rounds required. Let $l_1, \dots, l_j$ with $j \leq k$ be the lengths of a complete sequence of messages exchanged in the original protocol. Then the message of length l_i is split into $\lceil l_i/l \rceil$ blocks. The transmission of these blocks requires $2\lceil l_i/l \rceil - 1 = 2\lfloor (l_i-1)/l \rfloor + 1$ rounds. Thus the number of rounds necessary is at most

$$\sum_{i=1}^{j} \left(1 + 2\left\lfloor \frac{l_i - 1}{l} \right\rfloor\right) \leq j + 2\left\lfloor \frac{\sum_i (l_i - 1)}{l} \right\rfloor \leq k + 2\left\lfloor \frac{T-1}{T/k} \right\rfloor \leq 3k - 2. \qquad \square$$

THEOREM 9. *Let $\epsilon = \frac{1}{2} - \mu$, and let $\mathcal{A}$ be a probabilistic fixed-length k-round algorithm that computes a function $f : X_0 \times X_1 \to Y$ with error probability less than ϵ and $T = kl$ bits of communication. Then there exists a probabilistic fixed-length k-round algorithm $\mathcal{A}'$ that computes f with error probability less than $\epsilon' = \frac{1}{2} - \frac{\mu}{2}$ and message length $l' = 2l + \lceil \log \frac{2k}{\mu} \rceil$. For all $(x_0, x_1) \in X_0 \times X_1$ the messages in each round of the computations of $\mathcal{A}'$ on input (x_0, x_1) have either probability 0 or $p = m^{-1}$, where $m := 2^{l'-l}$.*

Proof. Let $\mathcal{A}$ be given by the transmission functions $\phi_i : X_i \times \{\{0,1\}^l\}^* \times \{0,1\}^l \to [0,1]$ and the output functions $a_i : X_i \times \{0,1\}^{kl} \times Y \to [0,1]$. For a computation $\alpha = (w_1, \dots, w_k) \in \{0,1\}^{lk}$, $a_{\hat{k}}(x_{\hat{k}}, \alpha, y)$ is the probability that its result will be y. By assumption, for any input (x_0, x_1)

$$\sum_{y \neq f(x_0,x_1)} \sum_{\alpha=(w_1\ldots w_k)} a_{\hat{k}}(x_{\hat{k}}, \alpha, y) \prod_{j=0}^{k-1} \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) < \epsilon.$$

If only the messages that are exchanged are considered, $\mathcal{A}$ may be regarded as a deterministic algorithm with an additional input of independently and uniformly distributed random numbers $r_0, \ldots, r_{k-1} \in [0, 1)$, where r_j is owned by $P_{\hat{j}}$. If on input (x_0, x_1) the messages $w_1, \ldots, w_j$ have been exchanged, the next message will be the string $w_{j+1} \in \{0, 1\}^l$ for which the integer $q = [w_{j+1}]$ represented by w_{j+1} is the least one such that

$$\sum_{[w]\leq q} \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w) > r_j.$$

For any sequence of messages $w_1, \ldots, w_j$ and input $x_{\hat{j}}$ choose $N_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) \in \mathbf{N}$, $w_{j+1} \in \{0, 1\}^l$, as an integer approximation of the corresponding $\phi_{\hat{j}}(\cdots)$ as follows:

$$\begin{aligned} N_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) := & \left\lfloor \frac{1}{2} + m \sum_{[w]\leq[w_{j+1}]} \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) \right\rfloor \\ & - \left\lfloor \frac{1}{2} + m \sum_{[w]\leq[w_{j+1}]-1} \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) \right\rfloor . \end{aligned}$$

Then

$$\left| N_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) - m \cdot \phi_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) \right| < 1$$

and

$$\sum_{w_{j+1}\in\{0,1\}^l} N_{\hat{j}}(x_{\hat{j}}, w_1 \cdots w_j, w_{j+1}) = m.$$

For algorithm $\mathcal{A}'$ the transmission functions $p_i : X_i \times \{\{0, 1\}^{l'}\}^* \times \{0, 1\}^{l'} \to \{0, p\}$ are now defined by

$$p_i(x_i, w_1v_1 \cdots w_jv_j, w_{j+1}v_{j+1}) = p \quad \text{iff} \quad [v_{j+1}] < N_i(x_i, w_1 \cdots w_j, w_{j+1}).$$

The output functions of $\mathcal{A}'$ are the same as the output functions of $\mathcal{A}$ on equivalent computations. (Two messages w_j of $\mathcal{A}$ and w'_jv_j of $\mathcal{A}'$ are called equivalent if $w_j = w'_j$.) For formal correctness we define $\mathcal{A}'$ more detailed in terms of additional random inputs as above: If on input (x_0, x_1) the messages $w_1v_1, \ldots, w_jv_j$ have been exchanged, the next message will be the string $w_{j+1}v_{j+1}$ for which $q = [w_{j+1}v_{j+1}]$ is the least integer such that

$$\sum_{[wv]\leq q} p_{\hat{j}}(x_{\hat{j}}, w_1v_1 \cdots w_jv_j, wv) > r_j.$$

Now consider computations of $\mathcal{A}$ and $\mathcal{A}'$ on the same input (x_0, x_1) and the same random numbers $r_0, \ldots, r_{k-1}$. Assume that the first j rounds are equivalent. Then, for any $w_{j+1} \in \{0, 1\}^l$ the probability that $\mathcal{A}$ sends w_{j+1} and $\mathcal{A}'$ sends some equivalent message $w_{j+1}v_{j+1}$ differs by at most p from the probability that $\mathcal{A}$ sends w_{j+1}. So the messages of round $j + 1$ are equivalent with probability at least $1 - p \cdot 2^l$ and the computations of $\mathcal{A}$ and $\mathcal{A}'$ are equivalent with probability at least $(1 - p \cdot 2^l)^k \geq 1 - pk \cdot 2^l$. Because the output functions are the same on equivalent computations, the error probability of $\mathcal{A}'$ is less than

$$\epsilon + pk \cdot 2^l \leq \epsilon + \frac{\mu}{2} = \epsilon'. \qquad \square$$

COROLLARY 8. *Let $UC_\epsilon(f)$ and $UC_{k,\epsilon}(f)$, respectively, denote the probabilistic communication complexity restricted to protocols with uniform distributions. Then for fixed $\epsilon < \frac{1}{2}$ the uniform complexity can be bounded by*

$$UC_{3k-2,\epsilon}(f) \leq O\left(\mathrm{C}_{k,\epsilon}(f)\left(1+\frac{k\log k}{\mathrm{C}_{k,\epsilon}(f)}\right)\right),$$
$$UC_\epsilon(f) \leq O\left(\mathrm{C}_\epsilon(f)\log \mathrm{C}_\epsilon(f)\right).$$

As an application of Theorems 6 and 9 we obtain Yao's lower bound on the probabilistic two-way complexity of Boolean-valued functions, which can easily be extended to general functions.

COROLLARY 9. *For any Boolean-valued function f and any fixed $\epsilon \in (0, \frac{1}{2})$*

$$\mathrm{C}_\epsilon(f) \geq \Omega(\log\log\max\{\mathrm{nrow}(F), \mathrm{ncol}(F)\}).$$

Proof. Any uniform probabilistic one-way algorithm with arbitrary error probability less than $\frac{1}{2}$ may be simulated by a deterministic one-way algorithm with at most an exponential increase of communication. (For each message, one bit suffices to specify its probability; the most probable result is correct.) Now let $\epsilon' = \frac{1}{2} - (\frac{1}{2} - \epsilon)\cdot 2^{-(\mathrm{C}_\epsilon(f)+1)}$. Then

$$\log(\mathrm{nrow}(F)-1) \leq \mathrm{C}_{1,\mathrm{det}}(f) \leq 2^{UC_{1,\epsilon'}(f)} \leq 2^{2\mathrm{C}_{1,\epsilon'}(f)+\mathrm{C}_\epsilon(f)+O(1)} \leq 2^{3\mathrm{C}_\epsilon(f)+O(1)}.$$

Hence $\mathrm{C}_\epsilon(f) \geq \Omega(\log\log\mathrm{nrow}(F))$, and clearly

$$\mathrm{C}_\epsilon(f) = \mathrm{C}_\epsilon(\tilde{f}) \geq \Omega(\log\log\mathrm{ncol}(F)). \qquad \square$$

REFERENCES

[AUY83] A. V. AHO, J. D. ULLMAN, AND M. YANNAKAKIS, *On notions of information transfer in* VLSI *circuits*, in Proc. 15th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1983, pp. 133–139.

[DGS87] P. DURIS, Z. GALIL, AND G. SCHNITGER, *Lower bounds on communication complexity*, Inform. and Comput., 73 (1987), pp. 1–22.

[F87] M. FÜRER, *The power of randomness for communication complexity*, in Proc. 19th Annual ACM Symposium of Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 178–181.

[H86] B. HALSTENBERG, *Zweiprozessor-Kommunikationskomplexität*, Diplomarbeit Fakultät für Mathematik Universität Bielefeld, Bielefeld, Germany, 1986.

[HR88] B. HALSTENBERG AND R. REISCHUK, *On different modes of communication*, in Proc. 20th Annual Symposium on Theory of Computing, Association for Computing Machinery, New York, 1988, pp. 162–172.

[KS92] B. KALYANASUNDARAM AND G. SCHNITGER, *The probabilistic communication complexity of set intersection*, SIAM J. Disc. Math., 5 (1992), pp. 545–557.

[MS82] K. MEHLHORN AND E. M. SCHMIDT, *Las Vegas is better than determinism in* VLSI *and distributed computing*, in Proc. 14th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1982, pp. 330–337.

[PS86] R. PATURI AND J. SIMON, *Probabilistic communication complexity*, J. Comput. System Sci., 33 (1986), pp. 106–123.

[T87] P. TIWARI, *Lower bounds on communication complexity in distributed computer networks*, J. Assoc. Comput. Mach., 34 (1987), pp. 921–938.

[Y79] A. C. YAO, *Some complexity questions related to distributed computing*, in Proc. 11th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1979, pp. 209–213.

[Y83] ———, *Lower bounds by probabilistic arguments*, in Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1983, pp. 420–428.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 935–948, October 1993

SUFFIX ARRAYS: A NEW METHOD FOR ON-LINE STRING SEARCHES*

UDI MANBER†‡ AND GENE MYERS†§

Abstract. A new and conceptually simple data structure, called a suffix array, for on-line string searches is introduced in this paper. Constructing and querying suffix arrays is reduced to a sort and search paradigm that employs novel algorithms. The main advantage of suffix arrays over suffix trees is that, in practice, they use three to five times less space. From a complexity standpoint, suffix arrays permit on-line string searches of the type, "Is W a substring of A?" to be answered in time $O(P + \log N)$, where P is the length of W and N is the length of A, which is competitive with (and in some cases slightly better than) suffix trees. The only drawback is that in those instances where the underlying alphabet is finite and small, suffix trees can be constructed in $O(N)$ time in the worst case, versus $O(N \log N)$ time for suffix arrays. However, an augmented algorithm is given that, regardless of the alphabet size, constructs suffix arrays in $O(N)$ *expected* time, albeit with lesser space efficiency. It is believed that suffix arrays will prove to be better in practice than suffix trees for many applications.

Key words. string searching, string matching, pattern matching, suffix trees, algorithms, text indexing, inverted indices

AMS subject classifications. 68P10, 68P20, 68Q25, 68U15

1. Introduction.

Finding all instances of a string W in a large text A is an important pattern matching problem. There are many applications in which a fixed text is queried many times. In these cases, it is worthwhile to construct a data structure to allow fast queries. The *Suffix tree* is a data structure that admits efficient on-line string searches. A suffix tree for a text A of length N over an alphabet Σ can be built in $O(N \log |\Sigma|)$ time and $O(N)$ space [Wei73], [McC76]. Suffix trees permit on-line string searches of the type, "Is W a substring of A?" to be answered in $O(P \log |\Sigma|)$ time, where P is the length of W. We explicitly consider the dependence of the complexity of the algorithms on $|\Sigma|$, rather than assume that it is a fixed constant, because Σ can be quite large for many applications. Suffix trees can also be constructed in time $O(N)$ with $O(P)$ time for a query, but this requires $O(N|\Sigma|)$ space, which renders this method impractical in many applications.

Suffix trees have been studied and used extensively. A survey paper by Apostolico [Apo85] cites over forty references. Suffix trees have been refined from tries to minimum state finite automaton for the text and its reverse [BBE85], generalized to on-line construction [MR80], [BB86], real-time construction of some features is possible [Sli80], and suffix trees have been parallelized [AIL88]. Suffix trees have been applied to fundamental string problems such as finding the longest repeated substring [Wei73], finding all squares or repetitions in a string [AP83], computing substring statistics [AP85], approximate string matching [Mye86], [LV89], [CL90], and string comparison [EH88]. They have also been used to address other types of problems such as text compression [RPE81], compressing assembly code [FWM84], inverted indices [Car75], and analyzing genetic sequences [CHM86]. Galil [Gal85] lists a number of open problems concerning suffix trees and on-line string searching.

In this paper, we present a new data structure, called the *suffix array* [MM90], that is basically a sorted list of all the suffixes of A. When a suffix array is coupled with information about the *longest common prefixes* (*lcps*) of adjacent elements in the suffix array, string searches can be answered in $O(P + \log N)$ time with a simple augmentation to a classic binary search.

*Received by the editors May 5, 1989; accepted for publication (in revised form) June 3, 1992.

†Department of Computer Science, University of Arizona, Tucson, Arizona 85721.

‡Supported in part by National Science Foundation Presidential Young Investigator Award grant DCR-8451397, with matching funds from AT&T, and by National Science Foundation grant CCR-9002351.

§Supported in part by National Institute of Health grant R01 LM04960-01, and by National Science Foundation grant CCR-9002351.

The suffix array and associated *lcp* information occupy a mere $2N$ integers, and searches are shown to require at most $P + \lceil \log_2(N-1) \rceil$ single-symbol comparisons. To build a suffix array (but not its *lcp* information) one could simply apply any string-sorting algorithm such as the $O(N \log N)$ expected-time algorithm of Baer and Lin [BL89]. But such an approach fails to take advantage of the fact that we are sorting a collection of related suffixes. We present an algorithm for constructing a suffix array and its *lcp* information with $3N$ integers[1] and $O(N \log N)$ time *in the worst case*. Time could be saved by constructing a suffix tree first, and then building the array with a traversal of the tree [Ro82] and the *lcp* information with constant-time nearest ancestor queries [SV88] on the tree. But this will require more space. Moreover, the algorithms for direct construction are interesting in their own right.

Our approach distills the nature of a suffix tree to its barest essence: A sorted array coupled with another to accelerate the search. Suffix arrays may be used in lieu of suffix trees in many (but not all) applications of this ubiquitous structure. Our search and sort approach is distinctly different and, in theory, provides superior querying time at the expense of somewhat slower construction. Galil [Gal85, Prob. 9] poses the problem of designing algorithms that are not dependent on $|\Sigma|$ and our algorithms meet this criterion, i.e., $O(P + \log N)$ search time with an $O(N)$ space structure, independent of Σ. With a few additional and simple $O(N)$ data structures, we show that suffix arrays can be constructed in $O(N)$ expected time, also independent of Σ. This claim is true under the assumption that all strings of length N are equally likely and exploits the fact that for such strings, the expected length of the *longest* repeated substring is $O(\log N / \log |\Sigma|)$ [KGO83].

In practice, an implementation based on a blend of the ideas in this paper compares favorably with an implementation based on suffix trees. Our suffix array structure requires only $5N$ bytes on a VAX, which is three to five times more space efficient than any reasonable suffix tree encoding. Search times are competitive, but suffix arrays do require three to ten times longer to build. For these reasons, we believe that suffix arrays will become the data structure of choice for the many applications where the text is very large. In fact, we recently found that the basic concept of suffix arrays (sans the *lcp* and a provable efficient algorithm) has been used in the Oxford English Dictionary (OED) project at the University of Waterloo [GBS92]. Suffix arrays have also been used as a basis for a sublinear approximate matching algorithm [My93] and for performing all pairwise comparisons between sequences in a protein sequence database [BG90].

The paper is organized as follows. In §2, we present the search algorithm under the assumption that the suffix array and the *lcp* information have been computed. In §3, we show how to construct the sorted suffix array. In §4, we give the algorithm for computing the *lcp* information. In §5, we modify the algorithms to achieve better expected running times. We end with empirical results and comments about practice in §6.

2. Searching.

2. Searching. Let $A = a_0 a_1 \cdots a_{N-1}$ be a large text of length N. Denote by $A_i = a_i a_{i+1} \cdots a_{N-1}$ the suffix of A that starts at position i. The basis of our data structure is a lexicographically sorted array, Pos, of the suffixes of A; namely, $Pos[k]$ is the start position of the kth smallest suffix in the set $\{A_0, A_1, \ldots A_{N-1}\}$. The sort that produces the array Pos is described in §3. For now, we assume that Pos is given; namely, $A_{Pos[0]} < A_{Pos[1]} < \ldots < A_{Pos[N-1]}$, where "<" denotes the lexicographical order.

For a string u, let u^p be the prefix consisting of the first p symbols of u if u contains more than p symbols, and u otherwise. We define the relation $<_p$ to be the lexicographical order of p-symbol prefixes; that is, $u <_p v$ if and only if $u^p < v^p$. We define the relations $\leq_p$, $=_p$,

[1]While the suffix array and *lcp* information occupy $2N$ integers, other N integers are needed during their construction. All the integers contain values in the range $[-N, N]$.

$\neq_p$, and $\geq_p$ in a similar way. Note that, for any choice of p, the Pos array is also ordered according to $\leq_p$, because $u < v$ implies $u \leq_p v$. All suffixes that have equal p-prefixes, for some $p < N$, must appear in consecutive positions in the Pos array, because the Pos array is sorted lexicographically. These facts are central to our search algorithm.

Suppose that we wish to find all instances of a string $W = w_0w_1 \cdots w_{P-1}$ of length $P \leq N$ in A. Let $L_W = \min(k : W \leq_p A_{Pos[k]} \textit{ or } k = N)$ and $R_W = \max(k : A_{Pos[k]} \leq_p W \textit{ or } k = -1)$. Since Pos is in $\leq_p$-order, it follows that W matches $a_ia_{i+1} \cdots a_{i+P-1}$ if and only if $i = Pos[k]$ for some $k \in [L_W, R_W]$. Thus, if L_W and R_W can be found quickly, then the number of matches is $R_W - L_W + 1$ and their left endpoints are given by $Pos[L_W]$, $Pos[L_W + 1], \ldots Pos[R_W]$. But Pos is in $\leq_p$-order; hence, a simple binary search can find L_W and R_W using $O(\log N)$ comparisons of strings of size at most P; each such comparison requires $O(P)$ single-symbol comparisons. Thus, the Pos array allows us to find all instances of a string in A in time $O(P \log N)$. The algorithm is given in Fig. 1.

```
if $W \leq_p A_{Pos[0]}$ then
    $L_W \leftarrow 0$
else if $W >_p A_{Pos[N-1]}$ then
    $L_W \leftarrow N$
else
  { $(L, R) \leftarrow (0, N - 1)$
    while $R - L > 1$ do
      { $M \leftarrow (L + R)/2$
        if $W \leq_p A_{Pos[M]}$ then
            $R \leftarrow M$
        else
            $L \leftarrow M$
      }
    $L_W \leftarrow R$
  }
```

FIG. 1. *An $O(P \log N)$ search for L_W.*

The algorithm in Fig. 1 is very simple, but its running time can be improved. We show next that the $\leq_p$-comparisons involved in the binary search need not be started from scratch in each iteration of the while loop. We can use information obtained from one comparison to speed up the ensuing comparisons. When this strategy is coupled with some additional precomputed information, the search is improved to $P + \lceil \log_2(N - 1) \rceil$ single-symbol comparisons in the worst case, which is a substantial improvement.

Let $lcp(v, w)$ be the length of the longest common prefix of v and w. When we lexicographically compare v and w in a left-to-right scan that ends at the first unequal symbol, we obtain $lcp(v, w)$ as a by-product. We can modify the binary search in Fig. 1 by maintaining two variables, l and r, such that $l = lcp(A_{Pos[L]}, W)$, and $r = lcp(W, A_{Pos[R]})$. Initially, l is set by the comparison of W and $A_{Pos[0]}$ in line 1, and r is set in the comparison against $A_{Pos[N-1]}$ in line 3. Thereafter, each comparison of W against $A_{Pos[M]}$ in line 9 permits l or r to be appropriately updated in line 10 or 12, respectively. By so maintaining l and r, $h = \min(l, r)$ single-symbol comparisons can be saved when comparing $A_{Pos[M]}$ to W, because $A_{Pos[L]} =_l W =_r A_{Pos[R]}$ implies $A_{Pos[k]} =_h W$ for all k in $[L, R]$ including M. While this reduces the number of single-symbol comparisons needed to determine the $\leq_p$-order of a midpoint with respect to W, it turns out that the worst-case running time is still $O(P \log N)$ (e.g., searching $ac_{N-2}b$ for $c^{P-1}b$).

To reduce the number of single-symbol comparisons to $P + \lceil \log_2(N-1) \rceil$ in the worst case, we use precomputed information about the lcps of $A_{Pos[M]}$ with each of $A_{Pos[L]}$ and $A_{Pos[R]}$. Consider the set of all triples (L, M, R) that can arise in the inner loop of the binary search of Fig. 1. There are exactly $N-2$ such triples, each with a unique midpoint $M \in [1, N-2]$, and for each triple $0 \le L < M < R \le N-1$. Suppose that (L_M, M, R_M) is the unique triple containing midpoint M. Let $Llcp$ be an array of size $N-2$ such that $Llcp[M] = lcp(A_{Pos[L_M]}, A_{Pos[M]})$ and let $Rlcp$ be another array of size $N-2$ such that $Rlcp[M] = lcp(A_{Pos[M]}, A_{Pos[R_M]})$. The construction of the two $(N-2)$-element arrays, $Llcp$ and $Rlcp$, can be interwoven with the sort producing Pos and will be shown in §4. For now, we assume that the $Llcp$ and $Rlcp$ arrays have been precomputed.

Consider an iteration of the search loop for triple (L, M, R). Let $h = \max(l, r)$ and let Δh be the difference between the value of h at the beginning and at the end of the iteration. Assuming, without loss of generality, that $r \le l = h$, there are three cases to consider,[2] based on whether $Llcp[M]$ is greater than, equal to, or less than h. The cases are illustrated in Fig. 2(a), 2(b), and 2(c), respectively. The vertical bars denote the lcps between W and the suffixes in the Pos array (except for l and r, these lcps are not known at the time we consider M). The shaded areas illustrate $Llcp[M]$. For each case, we must determine whether L_W is in the right half or the left half (the binary search step) and we must update the value of either l or r. It turns out that both these steps are easy to make:

Case 1 $Llcp[M] > l$ (Fig. 2(a)).
In this case, $A_{Pos[M]} =_{l+1} A_{Pos[L]} \neq_{l+1} W$, and so W must be in the right half and l is unchanged.

Case 2 $Llcp[M] = l$ (Fig. 2(b)).
In this case, we know that the first l symbols of $Pos[M]$ and W are equal; thus, we need to compare only the $l+1$st symbol, $l+2$nd symbol, and so on, until we find one, say, $l+j$, such that $W \neq_{l+j} Pos[M]$. The $l+j$th symbol determines whether L_W is in the right or left side. In either case, we also know the new value of r or l—it is $l+j$. Since $l = h$ at the beginning of the loop, this step takes $\Delta h + 1$ single-symbol comparisons.

Case 3 $Llcp[M] < l$ (Fig. 2(c)).
In this case, since W matched l symbols of L and $< l$ symbols of M, it is clear that L_W is in the left side and that the new value of r is $Llcp[M]$.

Hence, the use of the arrays $Llcp$ and $Rlcp$ (the $Rlcp$ array is used when $l < r$) reduces the number of single-symbol comparisons to no more than $\Delta h + 1$ for each iteration. Summing over all iterations and observing that $\Sigma \Delta h \le P$, the total number of single-symbol comparisons made in an on-line string search is at most $P + \lceil \log_2(N-1) \rceil$, and $O(P + \log N)$ time is taken in the worst-case. The precise search algorithm is given in Fig. 3.

3. Sorting.

The sorting is done in $\lceil \log_2(N+1) \rceil$ stages. In the first stage, the suffixes are put in buckets according to their first symbol. Then, inductively, each stage further partitions the buckets by sorting according to twice the number of symbols. For simplicity of notation, we number the stages 1, 2, 4, 8, etc., to indicate the number of affected symbols. Thus, in the Hth stage, the suffixes are sorted according to the $\le_H$-order. For simplicity, we pad the suffixes by adding blank symbols, such that the lengths of all of them become $N+1$. This padding is not necessary, but it simplifies the discussion; the version of the algorithm detailed in Fig. 4 does not make this assumption. The first stage consists of a bucket sort according to the first symbol of each suffix. The result of this sort is stored in the Pos array and in another array BH of Boolean values which demarcates the partitioning of the suffixes into m_1 buckets

[2]The first two cases can be combined in the program. We use three cases only for description purposes.

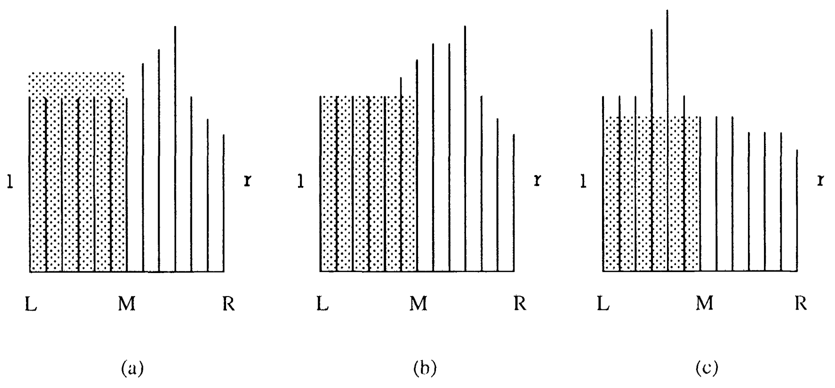


FIG. 2. *The three cases of the $O(P + \log N)$ search.*

```
$l \leftarrow lcp(A_{Pos[0]}, W)$
$r \leftarrow lcp(A_{Pos[N-1]}, W)$
if $l = P$ or $w_l \le a_{Pos[0]+l}$ then
          $L_W \leftarrow 0$
else if $r < P$ or $w_r \le a_{Pos[N-1]+r}$ then
          $L_W \leftarrow N$
else
   {  $(L, R) \leftarrow (0, N-1)$
          while $R - L > 1$ do
            {  $M \leftarrow (L + R)/2$
               if $l \ge r$ then
                    if $Lcp[M] \ge l$ then
                         $m \leftarrow l + lcp(A_{Pos[M]+l}, W_l)$
                    else
                         $m \leftarrow Lcp[M]$
               else
                    if$Rcp[M] \ge r$ then
                         $m \leftarrow r + lcp(A_{Pos[M]+r}, W_r)$
                    else
                         $m \leftarrow Rcp[M]$
               if $m = P$ or $w_m \le a_{Pos[M]+m}$ then
                    $(R, r) \leftarrow (M, m)$
               else
                    $(L, l) \leftarrow (M, m)$
            }
          $L_W \leftarrow R$
   }
```

FIG. 3. *An $O(P + \log N)$ search for L_W.*

$(m_1 \le |\Sigma|)$; each bucket holds the suffixes with the same first symbol. The array *Pos* will become progressively sorted as the algorithm proceeds. Assume that after the Hth stage the suffixes are partitioned into m_H buckets, each holding suffixes with the same H first symbols, and that these buckets are sorted according to the $\leq_H$-relation. We will show how to sort the elements in each H-bucket to produce the $\leq_{2H}$-order in $O(N)$ time. Our sorting algorithm uses similar ideas to those in [KMR72].

Let A_i and A_j be two suffixes belonging to the same bucket after the Hth step; that is, $A_i =_H A_j$. We need to compare A_i and A_j according to the next H symbols. But, the next H symbols of A_i (A_j) are exactly the first H symbols of $A_{i+H}(A_{j+H})$. By the assumption, we already know the relative order, according to the $\leq_H$-relation, of A_{i+H} and A_{j+H}. It remains to see how we can use that knowledge to complete the stage efficiently. We first describe the main idea, and we then show how to implement it efficiently.

We start with the first bucket, which must contain the smallest suffixes according to the $\leq_H$-relation. Let A_i be the first suffix in the first bucket (i.e., $Pos[0] = i$), and consider A_{i-H} (if $i - H < 0$, then we ignore A_i and take the suffix of $Pos[1]$, and so on). Since A_i starts with the smallest H-symbol string, A_{i-H} should be the first in its $2H$-bucket. Thus, we move A_{i-H} to the beginning of its bucket and mark this fact. For every bucket, we need to know the number of suffixes in that bucket that have already been moved and thus placed in $\leq_{2H}$-order. The algorithm basically scans the suffixes as they appear in the $\leq_H$-order, and for each A_i it moves A_{i-H} (if it exists) to the next available place in its H-bucket. While this basic idea is simple, its efficient implementation (in terms of both space and time) is not trivial. We describe it below.

We maintain three integer arrays, Pos, Prm, and $Count$, and two Boolean arrays, BH and $B2H$, all with N elements.[3] At the start of stage H, $Pos[i]$ contains the start position of the ith smallest suffix (according to the first H symbols), $Prm[i]$ is the inverse of Pos, namely, $Prm[Pos[i]] = i$, and $BH[i]$ is 1 if and only if $Pos[i]$ contains the leftmost suffix of an H-bucket (i.e., $A_{Pos[i]} \neq_H A_{Pos[i-1]}$). $Count$ and $B2H$ are temporary arrays; their use will become apparent in the description of a stage of the sort. A radix sort on the first symbol of each suffix is easily tailored to produce Pos, Prm, and BH for stage 1 in $O(N)$ time. Assume that Pos, Prm, and BH have the correct values after stage H, and consider stage $2H$.

We first reset $Prm[i]$ to point to the leftmost cell of the H-bucket containing the ith suffix rather than to the suffix's precise place in the bucket. We also initialize $Count[i]$ to 0 for all i. All operations above can be done in $O(N)$ time. We then scan the Pos array in increasing order, one bucket at a time. Let l and r $(l \leq r)$ mark the left and right boundary of the H-bucket currently being scanned. Let T_i (the H left extension of i) denote $Pos[i] - H$. For every i, $l \leq i \leq r$, we increment $Count[Prm[T_i]]$, set $Prm[T_i] = Prm[T_i] + Count[Prm[T_i]] - 1$, and set $B2H[Prm[T_i]]$ to 1. In effect, all the suffixes whose $H + 1$st through $2H$th symbols equal the unique H-prefix of the current H-bucket are moved to the top of their H-buckets with respect to the Prm array (Pos is updated momentarily). The $B2H$ field is used to mark

[3]We present a conceptually simpler but more space-expensive algorithm above in order to clearly expound the idea behind the sort. In fact, two N-element integer arrays are sufficient, and since the integers are always positive, we can use their sign bit for the Boolean values. Thus, the space requirement is only two integers per symbol. The trick is to remove the $Count$ array by temporarily using the Prm value of the leftmost suffix in a bucket to hold the count for the bucket. Instead of initializing $Count$ to 0 in the first step, we turn off the BH field of every bucket so that we can tell when a Prm value is the first reference to a particular bucket. The second step again consists of a scan of the Pos array. If $BH[Prm[T_i]]$ is off, then T_i is to be the first suffix in the $\leq_{2H}$ order of its H-bucket. We search the bucket for it and actually place it at the head of its bucket (as opposed to just modifying Prm to reflect where it will go in the simpler algorithm). This allows us to then use $Prm[T_i]$ as the counter for the bucket because we can restore it later knowing that the Prm-value of the first suffix in each bucket is being so used. We thus set the BH field back on and set $Prm[T_i]$ to 1. For later references to this bucket, which we know because $BH[Prm[T_i]]$ is now on, we simply adjust $Prm[T_i]$ with the count in $Prm[Pos[Prm[T_i]]]$ and bump the count. At the end of this step, the Prm fields used as counters are reset to the position of their suffix. The $B2H$ fields could not be set in the preceding steps because the BH values were being used as counter flags. In a separate pass, the $B2H$ values are updated to identify $2H$ buckets as in the simple algorithm.

those prefixes that were moved. Before the next H-bucket is considered, we make another pass through this one, find all the moved suffixes, and reset the $B2H$ fields such that only the leftmost of them in each $2H$-bucket is set to 1, and the rest are reset to 0. This way, the $B2H$ fields correctly mark the beginning of the $2H$-buckets. Thus the scan updates Prm and sets $B2H$ so that they are consistent with the $\leq_{2H}$-order of the suffixes. In the final step, we update the Pos array (which is the inverse of Prm) and set BH to $B2H$. All the steps above can clearly be done in $O(N)$ time, and, since there are at most $\lceil \log_2(N+1) \rceil$ stages, the sorting requires $O(N \log N)$ time in the worst case. A pseudocode implementation is given in Fig. 4. Average-case analysis is presented in §5.

```
# Sorting with $3N$ positive integers and
  $2N$ Booleans. "Count" can be eliminated
  and Booleans folded into sign bits, to
  produce a $2N$ integer sort.        #

var Pos, Prm, Count: array [0..$N-1$] of int;
    BH, B2H: array [0..$N$] of boolean;

# First phase bucket sort, $Bucket$ and
  $Link$ overlay $Pos$ and $Prm$, respectively #

for $a \in \Sigma$ do
      $Bucket[a] \leftarrow -1$
for $i \leftarrow 0, 1, 2, \cdots, N-1$ do
      $(Bucket[a_i], Link[i]) \leftarrow (i, Bucket[a_i])$
$c \leftarrow 0$
for $a \in \Sigma$ in order do
  {  $i \leftarrow Bucket[a]$
     while $i \neq -1$ do
       {  $j \leftarrow Link[i]$
          $Prm[i] \leftarrow c$
          if $i = Bucket[a]$ then
            {   $BH[c] \leftarrow$ true
                $Set(c, 0)$  # lcp info call #
            }
          else
                $BH[c] \leftarrow$ false
          $c \leftarrow c + 1$
          $i \leftarrow j$
       }
  }
$BH[N] \leftarrow$ true
for $i \in [0, N-1]$ do
      $Pos[Prm[i]] \leftarrow i$
```

```
# Inductive sorting stage (with lcp info calls) #

for $H \leftarrow 1, 2, 4, 8, \cdots$ while $H < N$ do
  {   for each $=_H$ bucket $[l, r]$ do
        {  $Count[l] \leftarrow 0$
           for $c \in [l, r]$ do
                 $Prm[Pos[c]] \leftarrow l$
        }
      $d \leftarrow N - H$
      $e \leftarrow Prm[d]$
      $Prm[d] \leftarrow e + Count[e]$
      $Count[e] \leftarrow Count[e] + 1$
      $B2H[Prm[d]] \leftarrow$ true
      for each $=_H$ bucket $[l, r]$ in $\leq_H$-order do
        {  for $d \in \{Pos[c] - H : c \in [l, r]\} \cap [0, N-1]$ do
             {  $e \leftarrow Prm[d]$
                $Prm[d] \leftarrow e + Count[e]$
                $Count[e] \leftarrow Count[e] + 1$
                $B2H[Prm[d]] \leftarrow$ true
             }
           for $d \in \{Pos[c] - H : c \in [l, r]\} \cap [0, N-1]$ do
                if $B2H[Prm[d]]$ then
                  {  $e \leftarrow \min(j : j > Prm[d]$ and
                                   $(BH[j]$ or not $B2H[j]))$
                     for $f \in [Prm[d] + 1, e - 1]$ do
                           $B2H[f] \leftarrow$ false
                  }
        }
      for $i \in [0, N-1]$ do
            $Pos[Prm[i]] \leftarrow i$
      for $i \in [0, N-1]$ do
           if $B2H[i]$ and not $BH[i]$ then
             {  $Set(i, H + Min\_Height(Prm[Pos[i-1] + H],$
                                        $Prm[Pos[i] + H]))$
                $BH[i] \leftarrow B2H[i]$
             }
  }
```

FIG. 4. *The $O(N \log N)$ suffix sorting algorithm.*

4. Finding longest common prefixes. The $O(P + \log N)$ search algorithm requires precomputed information about the lcps between the suffixes starting at each midpoint M and its left and right boundaries L_M and R_M. Computing a suffix array requires $2N$ integers, and we will see here that computing and recording the associated lcp information requires an extra N integers. We first show how to compute the lcps between suffixes that are *consecutive* in the sorted Pos array during the sort. We will see later how to compute *all* necessary lcps. The key idea is the following. Assume that after stage H of the sort we know the lcps between suffixes in adjacent buckets (after the first stage, the lcps between suffixes in adjacent buckets are 0). At stage $2H$ the buckets are partitioned according to $2H$ symbols. Thus, the lcps between suffixes in newly adjacent buckets must be at least H and at most $2H - 1$. Furthermore, if A_p and A_q are in the same H-bucket but are in distinct $2H$-buckets, then

$$lcp(A_p, A_q) = H + lcp(A_{p+H}, A_{q+H}). \tag{4.1}$$

Moreover, we know that $lcp(A_{p+H}, A_{q+H}) < H$. The problem is that we only have the lcps between suffixes in adjacent buckets, and A_{p+H} and A_{q+H} may not be in adjacent buckets. However, if $A_{Pos[i]}$ and $A_{Pos[j]}$, where $i < j$ have an lcp less than H and Pos is in $\leq_H$ order, then their lcp is the minimum of the lcps of every adjacent pair of suffixes between $Pos[i]$ and $Pos[j]$. That is,

$$lcp(A_{Pos[i]}, A_{Pos[j]}) = \min_{k\in[i,j-1]}(lcp(A_{Pos[k]}, A_{Pos[k+1]})). \tag{4.2}$$

Using (4.2) directly would require too much time, and maintaining the lcp of every pair of suffixes too much space. By using an $O(N)$-space height balanced tree structure that records the minimum pairwise lcp over a collection of intervals of the suffix array, we will be able to determine the lcp between any two suffixes in $O(\log N)$ time. We will describe this data structure, which we call an *interval tree*, after we firmly establish our basic approach (interval trees are similar to the Cartesian trees of Vuillemin [Vui80]).

We define $height(i) = lcp(A_{Pos[i-1]}, A_{Pos[i]})$, $1 \leq i \leq N-1$, where Pos is the final sorted order of the suffixes. These $N-1$ $height$ values are computed in an array $Hgt[i]$. The computation is performed inductively, together with the sort, such that $Hgt[i]$ achieves its correct value at stage H if and only if $height(i) < H$, and it is undefined (specifically, $N+1$) otherwise. Formally, if $height(i) < H$ then $Hgt[i] = height(i)$; otherwise, $Hgt[i] = N+1$. Notice that, if $height(i) < H$, then $A_{Pos[i-1]}$ and $A_{Pos[i]}$ must be in different H-buckets since H-buckets contain suffixes with the same H-symbol prefix. Further observe that a Hgt value is initially $N+1$, it is set to its $height$ value during the appropriate stage of the sort, and it retains this value thereafter.

Let Pos^H, Hgt^H, and Prm^H be the values of the given arrays at the end of stage H. In stage $2H$ of the sort, the $\leq_{2H}$-ordered list Pos^{2H} is produced by sorting the suffixes in each H-bucket of the $\leq_h$-ordered list Pos^H. The following lemma captures the essence of how we compute Hgt^{2H} from Hgt^H given Pos^{2H} and Prm^{2H}.

LEMMA 1. *If $H \leq height(i) < 2H$ then*

$$height(i) = H + \min(Hgt^H[k] : k \in [\min(a,b)+1, \max(a,b)]),$$

where $a = Prm^{2H}[Pos^{2H}[i-1]+H]$, and $b = Prm^{2H}[Pos^{2H}[i]+H]$.

Proof. Let $p = Pos^{2H}[i-1]$ and $q = Pos^{2H}[i]$. As we have observed, $height(i) < 2H$ implies $height(i) = H + lcp(A_{p+H}, A_{q+H})$. Next observe that $Pos^{2H}[a] = p + H$ and $Pos^{2H}[b] = q + H$ by the choice of a and b. Without loss of generality, assume that $a < b$. We now know that $height(i) = H + lcp(u, v)$, where $u = A_{Pos^{2H}[a]}$, $v = A_{Pos^{2H}[b]}$, $lcp(u,v) < H$, and $u <_H v$. Observe that $x <_H z$ and $x \leq_H y \leq_H z$ imply $lcp(x,z) = \min(lcp(x,y), lcp(y,z))$. It follows, by induction, that if $x_0 <_H x_n$ and $x_0 \leq_H x_1 \leq_H \cdots \leq_H x_n$, then $lcp(x_0, x_n) = \min(lcp(x_{k-1}, x_k) : k \in [1,n])$. Thus, $lcp(u,v) = \min(lcp(A_{Pos^{2H}[k-1]}, A_{Pos^{2H}[k]}) : k \in [a+1, b])$. Now $lcp(u,v) < H$ implies that at least one term in the minimum is less than H. For those terms less than H, $lcp(A_{Pos^{2H}[k-1]}, A_{Pos^{2H}[k]}) =$

$height(k) = Hgt^H[k]$. This, combined with the fact that $Hgt^H[k] = N + 1 \geq H$ for all other terms, gives the result. □

We are now ready to describe the algorithm. In the first stage, we set $Hgt[i]$ to 0 if $a_{Pos^1[i-1]} \neq a_{Pos^1[i]}$, and to $N + 1$ otherwise. This correctly establishes Hgt^1. At the end of stage $2H > 1$, we have computed Pos^{2H}, Prm^{2H}, and BH^{2H} (which marks the $2H$-buckets). Thus, by Lemma 1, the following code correctly establishes Hgt^{2H} from Hgt^H when placed at the end of a sorting stage. Essential to its correctness is the fact that Hgt^{2H} is Hgt^H except for the elements whose *height* values are in the range $[H, 2H - 1]$; their *Hgt* values are changed from $N + 1$ to their correct value.

```
for $i \in [1, N - 1]$ such that $BH[i]$ and $Hgt[i] > N$ do
  { $a \leftarrow Prm[Pos[i - 1] + H]$
    $b \leftarrow Prm[Pos[i] + H]$
    $Set\ (i, H + Min\_Height(\min(a, b) + 1, \max(a, b)))$ { these routines are defined below }
  }
```

The routine $Set(i, h)$ sets $Hgt[i]$ from $N + 1$ to h in our interval tree, and $Min\_Height(i, j)$ determines $\min(Hgt[k] : k \in [i, j])$ using the interval tree. We now show how to implement each routine in time $O(\log N)$ in the worst case. Consider a balanced and full binary tree with $N - 1$ leaves, which, in left-to-right order, correspond to the elements of the array *Hgt*. The tree has height $O(\log N)$ and $N - 2$ interior vertices. Assume that a value $Hgt[v]$ is also kept at each interior vertex v. We say that the tree is *current* if for every interior vertex v, $Hgt[v] = \min(Hgt[left(v)], Hgt[right(v)])$, where $left(v)$ and $right(v)$ are the left and right children of v.

Let T be a current tree. We need to perform two operations on the tree, a query $Min\_Height(i, j)$, and a dynamic operation $Set(i, h)$. The query operation $Min\_Height(i, j)$ computes $\min(Hgt[k] : k \in [i, j])$. It can be answered in $O(\log N)$ time as follows. Let $nca(i, j)$ be the nearest common ancestor of leaves i and j. The *nca* of leaves i and j can be found in $O(\log N)$ time by simply walking from the leaves to the root of the tree (it can be done in constant time using a more complicated data structure [SV88], but it is not necessary here). Let P be the set of vertices on the path from i to $nca(i, j)$ excluding $nca(i, j)$, and let Q be the similar path for leaf j. $Min\_Height(i, j)$ is the minimum of the following values: (1) $Hgt[i]$, (2) $Hgt[w]$ such that $right(v) = w$ and $w \notin P$ for some $v \in P$, (3) $Hgt[w]$ such that $left(v) = w$ and $w \notin Q$ for some $v \in Q$, and (4) $Hgt[j]$. These $O(\log N)$ vertices can be found, and their minimum can be computed in $O(\log N)$ time. The operation $Set(i, h)$ sets $Hgt[i]$ to h and then makes T current again by updating the *Hgt* values of the interior vertices on the path from i to the root. This takes $O(\log N)$ time.

Overall, the time taken to compute the *height* values in stage H is $O(N + \log N \cdot Set_H)$, where Set_H is the number of indices i for which $height(i) \in [H, 2H - 1]$. Since $\Sigma Set_H = N$ over all stages, the total additional time required to compute *Hgt* during the sort is $O(N \log N)$.

The *Hgt* array gives the *lcps* of suffixes that are consecutive in the *Pos* array. Moreover, an interior vertex of our interval tree gives the *lcp* between the suffixes at its leftmost and rightmost leaves. We now show that not only are the arrays *Llcp* and *Rlcp* computable from the array *Hgt* but are directly available from the interval tree by appropriately choosing the shape of the tree (heretofore we only asserted that it needed to be full and balanced). Specifically, we use the tree based on the binary search of Fig. 1. This implicitly-represented tree consists of $2N - 3$ vertices, each labeled with one of the $2N - 3$ pairs (L, R) that can arise at entry and exit from the while loop of the binary search. The root of the tree is labeled $(0, N - 1)$, and the remaining vertices are labeled either (L_M, M) or (M, R_M) for some midpoint $M \in [1, N - 2]$. From another perspective, the tree's $N - 2$ interior vertices are (L_M, R_M) for each midpoint M, and its $N - 1$ leaves are $(i - 1, i)$ for $i \in [1, N - 1]$ in left-to-right order. For each interior vertex, $left((L_M, R_M)) = (L_M, M)$ and $right((L_M, R_M)) = (M, R_M)$. Since the tree is full and balanced, it is appropriate for realizing *Set* and *Min_Height* if we let leaf $(i - 1, i)$ hold the value of $Hgt[i]$. Moreover, at the end of the sort, $Hgt[(L, R)] = \min(height(k) : k \in [L + 1, R]) = lcp(A_{Pos[L]}, A_{Pos[R]})$. Thus, $Llcp[M] = Hgt[(L_M, M)]$ and $Rlcp[M] = Hgt[(M, R_M)]$. So with this tree, the arrays *Llcp* and *Rlcp* are directly available upon completion of the sort.[4]

5. Linear time expected-case variations. We now consider the expected time complexity of constructing and searching suffix arrays. The variations presented in this section require additional $O(N)$ structures and so lose some of the space advantage. We present them primarily to show that linear time constructions are possible and because some of the ideas here are central to the implementation we found to be best in practice. We assume that all N-symbol strings are equally likely.[5] Under this input distribution, the expected length of the longest repeated substring has been shown to be $2 \log_{|\Sigma|} N + O(1)$ [KGO83]. This fact provides the central leverage for all the results that follow. Note that it immediately implies that, in the expected case, *Pos* will be completely sorted after $O(\log \log N)$ stages, and the sorting algorithm of §3 thus takes $O(N \log \log N)$ expected time.

The expected sorting time can be reduced to $O(N)$ by modifying the radix sort of the first stage as follows. Let $T = \lfloor \log_{|\Sigma|} N \rfloor$ and consider mapping each string of T symbols over Σ to the integer obtained when the string is viewed as a T-digit, radix-$|\Sigma|$ number. This frequently used encoding is an isomorphism onto the range $[0, |\Sigma|^T - 1] \subseteq [0, N - 1]$, and the $\leq$-relation on the integers is identical with the $\leq_T$-relation on the corresponding strings. Let $Int_T(A_p)$ be the integer encoding of the T-symbol prefix of suffix A_p. It is easy to compute $Int_T(A_p)$ for all p in a single $O(N)$ sweep of the text by employing the observation that $Int_T(A_p) = a_p|\Sigma|^{T-1} + \lfloor Int_T(A_{p+1})/|\Sigma| \rfloor$. Instead of performing the initial radix sort on the first symbol of each suffix, we perform it on the integer encoding of the first T symbols of each suffix. This radix sort still takes just $O(N)$ time and space because the choice of T guarantees that the integer encodings are all less than N. Moreover, it sorts the suffixes according to the $\leq_T$-relation. Effectively, the base case of the sort has been extended from $H = 1$ to $H = T$ with no loss of asymptotic efficiency. Since the expected length of the longest repeated substring is $T \cdot (2 + O(1/T))$, at most, two subsequent stages are needed to complete the sort in the expected case. Thus, this slight variation gives an $O(N)$ expected algorithm for sorting the suffixes.

Corresponding expected-case improvements for computing the lcp information, in addition to the sorted suffix array, are harder to come by. We can still achieve $O(N)$ expected-case time as we now show. We employ an approach to computing $height(i)$ that uses identity (4.1) recursively to obtain the desired lcps. Let the *sort history* of a particular sort be the tree that models the successive refinement of buckets during the sort. There is a vertex for each H-bucket except those H-buckets that are identical to the $(H/2)$-buckets containing them. The sort history thus has $O(N)$ vertices, as each leaf corresponds to a suffix and each interior

[4]The interval tree requires $2N - 3$ positive integers. However, the observation that one child of each interior vertex has the same value as its parent, permits interval trees (and thus the *Llcp* and *Rlcp* arrays) to be encoded and manipulated as $N - 1$ signed integers. Specifically, if the number at a vertex is positive, then the vertex contains the *Llcp* values; if it is negative, then its positive part is the *Rlcp* value. The other value is that of its parent. One must store both the *Llcp* and *Rlcp* values for the root but this is only one extra integer. Note that in order to have both the *Llcp* and *Rlcp* values available at a vertex we must descend to it from the root. This is naturally the case for searches, and for *Set* and *Min_height* we simply traverse from root to leaf and back again at no increase in asymptotic complexity.

[5]The ensuing results also hold under the more general model where each text is assumed to be the result of N independent Bernoulli trials of a $|\Sigma|$-sided coin toss that is not necessarily uniform.

vertex has at least two children. Each vertex contains a pointer to its parent, and each interior vertex also contains the stage number at which its bucket was split. The leaves of the tree are assumed to be arranged in an N element array, so that the singleton bucket for suffix A_p can be indexed by p. It is a straightforward exercise to build the sort history in $O(N)$ time overhead during the sort. Notice that we determine the values $height(i)$ only after the sort is finished.

Given the sort history produced by the sort, we determine the lcp of A_p and A_q as follows. First, we find the nearest common ancestor (nca) of suffixes A_p and A_q in the sort history using an $O(1)$ time nca algorithm [HT84], [SV88]. The stage number H associated with this ancestor tells us that $lcp(A_p, A_q) = H + lcp(A_{p+H}, A_{q+H}) \in [H, 2H - 1]$. We then recursively find the lcp of A_{p+H} and A_{q+H} by finding the nca of suffixes A_{p+H} and A_{q+H} in the history, and so on, until an nca is discovered to be the root of the history. At each successive level of the recursion, the stage number of the nca is at least halved, and so the number of levels performed is $O(\log L)$, where L is the lcp of A_p and A_q. Because the longest repeated substring has expected length $O(\log_{|\Sigma|} N)$, the $N - 1$ lcp values of adjacent sorted suffices are found in $O(N \log\log N)$ expected time.

The scheme above can be improved to $O(N)$ expected time by strengthening the induction basis as was done for the sort. Suppose that we stop the recursion above when the stage number of an nca becomes less than $T' = \lfloor \frac{1}{2} \log_{|\Sigma|} N \rfloor$. Our knowledge of the expected maximum lcp length implies that, on average, only three or four levels are performed before this condition is met. Each level takes $O(1)$ time, and we are left having to determine the lcp of two suffices (say, A_p and A_q) that is known to be less than T'. To answer this final lcp query in constant time, we build a $|\Sigma|^{T'}$-by-$|\Sigma|^{T'}$ array $Lookup$, where $Lookup\,[Int_{T'}(x), Int_{T'}(y)] = lcp(x, y)$ for all T'-symbol strings x and y. By the choice of T' there are no more than N entries in the array, and they can be computed incrementally in an $O(N)$ preprocessing step along with the integer encodings $Int_{T'}(A_p)$ for all p. So for the final level of the recursion, $lcp(A_p, A_q) = Lookup[Int_{T'}(A_p), Int_{T'}(A_q)]$ may be computed in $O(1)$ time via table lookup. In summary, we can compute the lcp between any two suffixes in $O(1)$ expected time, and so can product the lcp array in $O(N)$ expected time.

The technique of using integer encodings of $O(\log N)$-symbol strings to speedup the expected preprocessing times, also provides a *pragmatic* speed-up for searching. For any $K \le T$, let $Buck[k] = \min\{i : Int_K(A_{Pos[k]}) = i\}$. This bucket array contains $|\Sigma|^K$ nondecreasing entries and can be computed from the ordered suffix array in $O(N)$ additional time. Given a word W, we know immediately that L_W and R_W are between $Buck[k]$ and $Buck[k+1] - 1$ for $k = Int_K(W)$. Thus, in $O(K)$ time we can limit the interval to which we apply the search algorithm proper to one whose average size is $N/|\Sigma|^K$. Choosing K to be T or very close to T, implies that the search proper is applied to an $O(1)$ expected-size interval and thus consumes $O(P)$ time in expectation *regardless* of whether the algorithm of Fig. 1 or Fig. 3 is used. While the use of bucketing does not asymptotically improve either worst-case or expected-case times, we found this speed-up very important in practice.

6. Practice. A primary motivation for this paper was to be able to efficiently answer on-line string queries for very long genetic sequences (on the order of one million or more symbols long). In practice, it is the space overhead of the query data structure that limits the largest text that may be handled. Throughout this section, we measure space in numbers of integers where typical current architectures model each integer in 4 bytes. Suffix trees are quite space expensive, requiring roughly 4 integers of overhead per text character. Utilizing an appropriate blend of the suffix array algorithms given in this paper, we developed an implementation requiring 1.25 integers of overhead per text character whose construction and search speeds are competitive with suffix trees.

There are three distinct ways to implement a data structure for suffix trees, depending on how the outedges of an interior vertex are encoded. We characterize the space occupied by that part of the structure needed for searches and ignore the extra integer (for "suffix links") needed during the suffix tree's construction. Using a $|\Sigma|$-element vector to model the outedges gives a structure requiring $2N + (|\Sigma| + 2) \cdot I$ integers, where I is the number of interior nodes in the suffix tree. Encoding each set of outedges with a binary search tree requires $2N + 5I$ integers. Finally, encoding each outedge set as a linked list requires $2N + 4I$ integers. The parameter $I < N$ varies as a function of the text. The first four columns of Table 1 illustrate the value of I/N and the per-text-symbol space consumption of each of the three coding schemes assuming that an integer occupies 4 bytes. These results suggest that the linked scheme is the most space parsimonious. We developed a tightly coded implementation of this scheme for the timing comparisons with our suffix array software.

TABLE 1
Empirical results for texts of length 100,000.

	Space (Bytes/text symbol)					Construction Time		Search Time	
		S. Trees							
	I/N	Link	Tree	Vector	*S. Arrays*	*S. Trees*	*S. Arrays*	*S. Trees*	*S. Arrays*
Random ($|\Sigma| = 2$)	.99	23.8	27.8	19.8	5.0	2.6	7.1	6.0	5.8
Random ($|\Sigma| = 4$)	.62	17.9	20.4	18.9	5.0	3.1	11.7	5.2	5.6
Random ($|\Sigma| = 8$)	.45	15.2	17.0	20.8	5.0	4.6	11.4	5.8	6.6
Random ($|\Sigma| = 16$)	.37	13.9	15.4	30.6	5.0	6.9	11.6	9.2	6.8
Random ($|\Sigma| = 32$)	.31	13.0	14.2	46.2	5.0	10.9	11.7	10.2	7.0
Text ($|\Sigma| = 96$)	.54	16.6	18.8	220.0	5.0	5.3	28.3	22.4	9.5
Code ($|\Sigma| = 96$)	.63	18.1	20.6	255.0	5.0	4.2	35.9	29.3	9.0
DNA ($|\Sigma| = 4$)	.72	19.5	22.4	25.2	5.0	2.9	18.7	14.6	9.2

For our practical implementation, we chose to build just a suffix array and use the radix-N initial bucket sort described in §5 to build it in $O(N)$ expected time. Without the lcp array the search must take $O(P \log N)$ worst-case time. However, keeping variables l and r as suggested in arriving at the $O(P + \log N)$ search significantly improves search speed in practice. We further accelerate the search to $O(P)$ expected time by using a bucket table with $K = \log_{|\Sigma|} N/4$ as described in §5. Our search structure thus consists of an N integer suffix array and an $N/4$ integer bucket array, and so consumes only 1.25 integers/5 bytes per text symbol assuming an integer is 4 bytes. As discussed in §3, $2N$ integers are required to construct the suffix array (without lcp information). So constructing an array requires a little more space than is required by the search structure, as is true for suffix trees. Given that construction is usually once only, we choose to compare the sizes of the search structures in Table 1.

Table 1 summarizes a number of timing experiments on texts of length 100,000. All times are in seconds and were obtained on a VAX 8650 running UNIX. Columns 6 and 7 give the times for constructing the suffix tree and suffix array, respectively. Columns 8 and 9 give the time to perform 100,000 successful queries of length 20 for the suffix tree and array, respectively. In synopsis, suffix arrays are 3–10 times more expensive to build, 2.5–5 times more space efficient, and can be queried at speeds comparable to suffix trees.

Acknowledgment. The authors wish to thank the referees for the insightful comments, especially one anonymous referee whose meticulous comments were beyond the call of duty.

REFERENCES

[Apo85] A. Apostolico, *The myriad virtues of subword trees*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil, eds., NATO ASI Series F: Computer and System Sciences, Vol. 12, Springer-Verlag, Berlin, New York, 1985, pp. 85–96.

[AIL88] A. Apostolico, C. Iliopoulos, G. Landau, B. Schieber, and U. Vishkin, *Parallel construction of a suffix tree with applications*, Algorithmica, 3 (1988), pp. 347–366.

[AP83] A. Apostolico and F. P. Preparata, *Optimal off-line detection of repetitions in a string*, Theoret. Comput. Sci., 22 (1983), pp. 297–315.

[AP85] ———, *Structural properties of the string statistics problem*, J. Comput. System Sci., 31 (1985), pp. 394–411.

[BL89] J. Baer and Y. Lin, *Improving quicksort performance with a codeword data structure*, IEEE Trans. Software Engrg., 15 (1989), pp. 622–631.

[BG90] R. Baeza-Yates and G. Gonnet, *All-against-all sequence matching*, Tech. Rep., Dept. of Computer Science, University of Chile, 1990.

[BB86] J. Blumer and A. Blumer, *On-line Construction of a Complete Inverted File*, Tech. Rep. UCSC-CRL-86-11, Department of Computer Science, University of Colorado, Boulder, CO, 1986.

[BBE85] J. Blumer, A. Blumer, E. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas, *The smallest automation recognizing the subwords of a text*, Theoret. Comput. Sci., 40 (1985), pp. 31–35.

[Car75] A. F. Cardenas, *Analysis and performance of inverted data base structures*, Comm. ACM, 18 (1975), pp. 253–263.

[CHM86] B. Clift, D. Haussler, R. McConnell, T. D. Schneider, and G. D. Stormo, *Sequence landscapes*, Nucleic Acids Research, 4 (1986), pp. 141–158.

[CL90] W. I. Chang and E. L. Lawler, *Approximate string matching in sublinear expected time*, Proc. IEEE 31st Annual Symp. on Foundations of Computer Science, St. Louis, MO, October 1990, pp. 116–124.

[EH88] A. Ehrenfeucht and D. Haussler, *A new distance metric on strings computable in linear time*, Discrete Appl. Math., 40 (1988), pp. 191–203.

[FWM84] C. Fraser, A. Wendt, and E. W. Myers, *Analyzing and compressing assembly code*, Proceedings of the SIGPLAN Symposium on Compiler Construction, 1984, pp. 117–121.

[Gal85] Z. Galil, *Open problems in stringology*, in Combinatorial Algorithms on Words, A. Apostolico and Z. Galil., eds., NATO ASI Series F: Computer and System Sciences, Vol. 12, Springer-Verlag, Berlin, New York, 1985, pp. 1–8.

[GBS92] G. Gonnet, R. Baeza-Yates, and T. Snider, *New indices for text: PAT trees and PAT arrays*, in Information Retrieval: Data Structures and Algorithms, B. Frakes and R. Baeza-Yates, eds., Prentice-Hall, Englewood Cliffs, NJ, 1992.

[HT84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[KGO83] S. Karlin, G. Ghandour, F. Ost, S. Tavare, and L. J. Korn, *New approaches for computer analysis of nucleic acid sequences*, Proc. Natl. Acad. Sci. USA, 80 (September 1983), pp. 5660–5664.

[KMR72] R. M. Karp, R. E. Miller, and A. L. Rosenberg, *Rapid identification of repeated patterns in strings, trees and arrays*, Fourth Annual ACM Symposium on Theory of Computing, (May 1972), pp. 125–136.

[LV89] G. M. Landau and U. Vishkin, *Fast parallel and serial approximate string matching*, J. Algor., 10 (1989), pp. 157–169.

[McC76] E. M. McCreight, *A space-economical suffix tree construction algorithm*, J. ACM, 23 (1976), pp. 262–272.

[MM90] U. Manber and E. W. Myers, *Suffix arrays: A new method for on-line string searches*, First ACM-SIAM Symposium on Discrete Algor., San Francisco, CA, January 1990, pp. 319–327.

[MR80] M. E. Majster and A. Reiser, *Efficient on-line construction and correction of position trees*, SIAM J. Comput., 9 (1980), pp. 785–807.

[Mye86] E. W. Myers, *Incremental alignment algorithms and their applications*, Tech. Report TR86-22, Department of Computer Science, University of Arizona, Tucson, AZ.

[My93] ———, *A sublinear algorithm for approximate keyword searching*, Algorithmica, 1993, to appear.

[Ro82] M. Rodeh, *A fast test for unique decipherability based on suffix tree*, IEEE Trans. Inform. Theory, 28 (1982), pp. 648–651.

[RPE81] M. Rodeh, V. R. Pratt, and S. Even, *Linear algorithm for data compression via string matching*, J. ACM, 28 (1982), pp. 16–24.

[Sli80] A. O. Slisenko, *Detection of periodicities and string-matching in real time*, J. Soviet Math., 22 (1983), pp. 1316–1387; translated from Zpiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR, 105 (1980), pp. 62–173.

[SV88] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.

[Vui80] J. Vuillemin, *A unified look at data structures*, Comm. ACM, 23 (1980), pp. 229–239.

[Wei73] P. Weiner, *Linear pattern matching algorithm*, Proc. 14th IEEE Sympos. Switching and Automata Theory, (1973), pp. 1–11.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 949–975, October 1993

RANDOMIZING REDUCTIONS OF SEARCH PROBLEMS*

ANDREAS BLASS† AND YURI GUREVICH‡

Abstract. This paper closes a gap in the foundations of the theory of average-case complexity. First, it clarifies the notion of a feasible solution for a search problem and proves its robustness. Second, it gives a general and usable notion of many–one randomizing reductions of search problems and proves that it has desirable properties. All reductions of search problems to search problems in the literature on average-case complexity can be viewed as such many–one randomizing reductions, including those reductions in the literature that use iterations and therefore do not look many–one. As an illustration, this paper presents a careful proof of a theorem of Impagliazzo and Levin in the framework of the present work.

Key words. average case, search problems, reduction, randomization

AMS subject classification. 68Q15

1. Introduction and results.

Reduction theory for average-case computational complexity was pioneered by Levin [9]. Recently, one of us (Gurevich) wrote a survey on the subject [6], and we refer the reader there for a general background. However, the present paper is largely self-contained; we recall all necessary definitions.

We develop the foundations of the theory of many–one reductions of search problems in the context of randomizing algorithms. Many–one reductions are easier to use than Turing reductions, but one may wonder how restrictive they are. Indeed, there are cases in the literature on average-case complexity [1], [8] in which reductions of search problems to search problems are not many–one; those reductions are iterations of many–one reductions. It turns out, however, that iteration is needed, not for reductions, but only for the resulting search algorithms. Thus the theory of reductions can be simplified by treating reductions separately from iteration. Our notion of many–one reduction was influenced by the specific reductions used in [1], [8], [11].

As a general framework for the study of average-case complexity, we use domains. Definitions of domains and polynomiality on average are recalled in §2. Essentially, a domain is a set of strings with a size function and a probability distribution. If X is a subset of positive probability of a domain A, then the restriction of A to X is the domain obtained by assigning zero probability to elements of $A - X$ and renormalizing the probabilities on X. The phrase "polynomial on average" is abbreviated AP.

We consider search problems of the following sort. A domain A (of instances) is given, along with a binary relation W between elements of A and arbitrary strings in some fixed alphabet Σ. If $W(x, w)$ holds, the string w is a *witness* for the instance x (with respect to W) and x is a *positive* instance; an instance x is *negative* if it has no witnesses. We assume that there exists an algorithm that, given an instance x of nonzero probability and a witness w for x, computes $W(x, w)$ in time polynomial in $|x| + |w|$, where $|x| = |x|_A$ is the size of x with respect to A and $|w|$ is the ordinary length of w. The search problem SP(A, W) is the

*Received by the editors November 14, 1991; accepted for publication (in revised form) June 9, 1992. An extended abstract of this paper was published earlier in [3].

†Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109-1003 (ablass@math.lsa.umich.edu). This author's work was partially supported by National Science Foundation grant DMR 88-01988.

‡Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2122 (gurevich@eecs.umich.edu). This author's work was partially supported by National Science Foundation grant CCR 89-04728 and Office of Naval Research grant N00014-91-J-11861.

following: Given a positive instance $x \in A$ of nonzero probability, find a witness w such that $W(x, w)$.

Let A^+ be the restriction of domain A to positive instances. An algorithm for $\mathrm{SP}(A, W)$ is supposed to find witnesses for instances $x \in A^+$ of nonzero probability; it is immaterial what the algorithm does for other instances.

In the context of polynomial-time (PTime) search algorithms, it is reasonable to assume that the length of a witness is bounded by a polynomial of the size of the instance and that $W(x, w)$ is computable in time polynomial in $|x|$ since such an algorithm, given x, could produce only w's that are polynomially bounded with respect to x. Search problems of this kind are called NP search problems; every NP decision problem gives rise to an NP search problem. However, we shall work, not with polynomial-time algorithms but with (randomizing) algorithms whose running time is polynomial on average (AP-time algorithms). For such algorithms there may be an occasional input x (and a random string r) for which the running time and the length of the output are very long. Therefore, we do not require that the length of a witness be bounded by a polynomial of the size of the instance or that $W(x, w)$ be computable in time polynomial in $|x|$. However, if we require that the length of a witness be bounded by a polynomial of the size of the instance or that $W(x, w)$ be computable in time polynomial in $|x|$, all of our theorems remain true. On the other hand, the condition that $W(x, w)$ is PTime computable relative to $|x| + |w|$ can be relaxed to a hybrid—PTime with respect to w and AP time with respect to x—without affecting the results of this paper [6].

It is often useful to consider search problems in which one seeks witnesses w for certain instances but does not care what the algorithm does for other instances, even if those other instances have witnesses. For example, in the unique search problems, considered in [1], the algorithm applied to an instance x (and a random string) should produce a witness w when w is the unique witness for x, but it does not matter what the algorithm does when it is applied to an instance for which there is either no witness or more than one witness. Such problems can be incorporated into our framework by replacing A with its restriction to important instances.

An interesting case is one in which every instance of nonzero probability has a witness. This can be achieved by restricting the domain to positive instances, but there are other ways as well. Consider, for example, the usual Hamiltonian circuit search problem with some probability distribution on graphs. Not every graph has a Hamiltonian circuit. However, we can require, in the case of a graph without a Hamiltonian circuit, that a witness establish the nonexistence of a Hamiltonian circuit. For example, a witness may be a computation establishing the nonexistence of any Hamiltonian circuit. Such a witness may be long, but $W(x, w)$ is required to be computable in time polynomial in $|x| + |w|$, not in $|x|$, so that long witnesses may be all right. Indeed, there exists an algorithm for a randomized version of the Hamiltonian circuit search (with an arbitrary but constant edge probability) that finds a Hamiltonian circuit or establishes the nonexistence of one in linear (in the number of vertices) expected time [7].

Before we explain our notion of feasible solutions, let us recall a useful notion of random function [2]. Deterministic algorithms compute deterministic functions. Similarly, randomizing (coin-flipping) algorithms compute random functions. Formally, a random function f on a domain A is a deterministic function on an extension D_f of A whose elements are pairs (x, s), where $x \in A$ and s is a binary string to be regarded as the sequence of coin flips used by the algorithm. In this introduction we allow ourselves to be sloppy about the distinction between a randomizing algorithm and the random function it computes. The definition of random functions is recalled in §2.

There are different approaches to defining what constitutes a good randomizing algorithm for a search problem. One approach, closer to the actual use of such algorithms, requires that a correct answer be obtained, with probability equal to 1 or at least very nearly 1, in reasonable time (i.e., in AP time). Another approach requires only that the algorithms have, for every input of positive probability, a reasonably high probability of success, say, at least 1%. For many purposes the two approaches are equivalent since an algorithm of the second, weaker sort can be iterated with independent random strings to obtain a very high probability of success at the cost of a moderate increase in the running time. A polynomial number of iterations suffices to improve success probabilities from as low as a reciprocal of a polynomial of $|x|$ to as high as exponentially (relative to $|x|$) close to 1. For average-case complexity the situation is even better. We can start with a success probability whose reciprocal is not polynomially bounded but only polynomial on average, and we can iterate the algorithm to obtain a success probability 1 without increasing the time beyond AP.

To make these ideas precise we introduce in §3 the notions of AP-*time* randomizing algorithms and *almost-total* randomizing algorithms. An almost-total, AP-time randomizing algorithm succeeds with probability 1, but an arbitrary AP-time algorithm needs only to succeed often enough on each instance. The notion of AP-time randomizing algorithms is the central notion of this paper. Roughly speaking, a randomizing algorithm M with inputs in a domain A is AP time if it has a set of good inputs (x, s), where $x \in A$ and s is a finite sequence of coin flips, such that M terminates on every good input, the proportion of good inputs (among all inputs) is at least the reciprocal of an AP function, and the computation time of M on good inputs is not too large. It is not required that the set of good inputs be easily recognizable in any way.

DEFINITION 1.1. A randomizing algorithm M is an AP-time solution (respectively, almost-total, AP-time solution) for a search problem $\mathrm{SP}(A, W)$ if M is AP time (respectively, almost total, AP time) on A^+ and every terminating computation of M with input $x \in A^+$ of nonzero probability (and arbitrary coin flips) produces as output a witness w with $(x, w) \in W$.

It might seem more natural to require that an AP-time algorithm produce appropriate outputs only for enough, but not all, of the terminating computations, but our definition is somewhat simpler and is essentially equivalent: Every AP-time algorithm M for a search problem Π that produces correct outputs on its good inputs can be converted to an AP-time solution M' for Π.

Choose any iteration technique, subject to some natural fairness and carefulness conditions specified in §4. Theorem 4.4, the main result of §4, implies the following theorem.

THEOREM 1.2. *A randomizing algorithm M is an* AP-*time solution for a search problem Π if and only if the iteration M^* of M is an* AP-*time solution for Π that is almost total.*

Thus an AP-time solution is iterable to an almost-total, AP-time solution, and this is optimal. In addition, several natural iteration techniques are shown to satisfy the fairness and carefulness conditions in question.

Now let us turn our attention to reductions. What is a (randomizing) many–one reduction of a search problem $\Pi = \mathrm{SP}(A, W)$ to another search problem $\Pi' = \mathrm{SP}(A', W')$? Such a reduction consists of two parts, the first producing, for each instance $x \in A^+$ of nonzero probability and some random strings s, an instance $x' = f(x, s) \in A'$, and the second producing, for each witness w' with $(x', w') \in W'$, a witness $w = g(x, s, w')$ with $(x, w) \in W$. A reduction and any algorithm M' solving Π' yield, by composition, an algorithm M (equal to $g \circ M' \circ f$ if g depends only on w') that solves Π. We wish to determine conditions on randomizing algorithms f and g to ensure that when M' has a good average-case solution, namely, either almost total and AP time or simply AP time, then so does M. The conditions are to be independent of M', so that a reduction makes sense even when there is no algorithm M' available. Indeed, one of the most common uses of reductions is to prove that Π' is complete in one sense or another, and therefore no good algorithm solving Π' is expected to exist. (Also, one may want to require that if M' be an efficient algorithm that uses some oracle; then M should be a similarly efficient algorithm that uses the same oracle.)

The interesting parts of our conditions on f and g concern f; we shall simply require g to be polynomial time. Notice that it would make no sense to weaken this to, say, AP time on the set $(\Sigma')^*$ of potential witnesses because we are not given a probability distribution on $(\Sigma')^*$. Of course, M' together with the probability distribution $\mathbf{P}_{A'}$ of A' induces a distribution on $(\Sigma')^*$, but our conditions are to be independent of M'. And if we required g to be AP time with respect to some natural distribution on $(\Sigma')^*$, then some algorithm M' might, with high probability, produce witnesses w' from a small set to which our natural distribution assigned low probability and on which g takes very long to compute. Because of this situation, we assume that g is polynomial time, and then we can ignore g in the following sense: M is AP time or almost total and AP time if and only if $M' \circ f$ is. (There is a natural way to weaken the condition of PTime computability of $g(x, s, w')$ to a hybrid—Ptime with respect to w' and AP time with respect to (x, s)—that preserves the results of this paper [4], but again we stick here to the simpler assumption of PTime computability.)

What conditions should f satisfy? Consider first deterministic reductions. Obviously, f should be AP-time computable in this case. What else? Nontrivial examples of deterministic reductions may be found in [5], [9]. Actually, the two papers are devoted to NP decision problems, but one can consider the search versions of those problems instead; the deterministic reductions of decision problems naturally transform to deterministic reductions of the corresponding search problems. We tried to give a natural motivation and analysis for the additional requirement (beyond AP-time computability) in the notion of deterministic reductions.

THEOREM 1.3 [2]. *For every (deterministic) function f from A to A' the following two conditions are equivalent*:

- *For every* AP *function T on A' the composition $T \circ f$ is* AP.
- *The function $x \mapsto |fx|_{A'}$ is* AP *and A' dominates A with respect to f, i.e., the ratio $\mathbf{P}_A[f^{-1}\{fx\}]/\mathbf{P}_{A'}\{fx\}$ is* AP *on A.*

Thus if f is AP-time computable and A' dominates A with respect to f, then for every AP time algorithm M' on A' the composite algorithm $M' \circ f$ is AP on A.

This gives rise to the following reduction notions.

DEFINITION 1.4. A deterministic AP-time reduction of a domain A to a domain A' is a deterministic AP-time-computable function f from A to A' such that A' dominates A with respect to f.

Since the domination relation is transitive [2], deterministic reducibility of domains is transitive.

DEFINITION 1.5. A deterministic AP-time reduction of a search problem SP(A, W) to a search problem SP(A', W') consists of the following:

- A deterministic AP-time reduction f of A^+ to $(A')^+$.
- A polynomial-time computable function g such that if $x' = f(x)$ and w' is a W'-witness for x', then $g(x, w')$ is W-witness for x.

COROLLARY 1.6. *Deterministic* AP-*time reducibility of search problems is transitive. Further, a search problem Π is solvable by a deterministic* AP-*time algorithm if it is deterministically* AP-*time reducible to some Π' that is solvable in deterministic* AP *time.*

Unfortunately, deterministic reductions are too weak for many purposes [5], and stronger randomizing reductions are used in [1], [5], [8], [11]. The task of cleaning up the notion of randomizing reductions was the main motivation of this paper.

So suppose that f is a random function from a domain A to a domain A', and let T be an AP-time random function on A'. There are four situations, according to whether T is assumed to be almost total or not and whether $T \circ f$ is required to be almost total and AP time or only

AP time. It will be easy to see that one of the four situations is absurd; no f can convert arbitrary AP-time functions T into almost-total $T \circ f$, because if T succeeds with probability $\frac{1}{2}$ for every x', then $T \circ f$ will do no better.

Of the three reasonable situations, the easiest to analyze is the one in which T is assumed to be almost total and $T \circ f$ is required to be almost total and AP time. It is clear that such an f must itself be AP time and almost total, because a computation of f is an initial segment of any successful computation of $T \circ f$, no matter how trivial T is. It turns out that Theorem 1.3 generalizes nicely in this case.

THEOREM 1.7. *Let f be an almost-total,* AP*-time-computable random function from a domain A to a domain A' such that A' dominates D_f with respect to f. Then for every almost-total,* AP*-time randomizing algorithm M' on A' the composite algorithm $M' \circ f$ is* AP *time and almost total.*

Theorem 1.7 follows from the more informative Theorem 5.2. It gives rise to the following reduction notions.

DEFINITION 1.8. An almost-total, AP-time randomizing reduction of a domain A to a domain A' is an almost total, AP-time-computable random function f from A to A' such that A' dominates D_f with respect to f.

DEFINITION 1.9. An almost-total, AP-time randomizing reduction of a search problem SP(A, W) to a search problem SP(A', W') consists of the following:

- An almost-total, AP-time randomizing reduction of A^+ to $(A')^+$.
- A polynomial-time computable function g such that if $x' = f(x, s)$ and w' is a W'-witness for x', then $g(x, s, w')$ is a W-witness for x.

COROLLARY 1.10. *Almost-total,* AP*-time randomizing reducibility on search problems is transitive. Further, a search problem Π is solvable by an almost-total,* AP-time *randomizing algorithm if it is reducible, by an almost-total,* AP*-time randomizing algorithm, to some Π' that is solvable by an almost-total,* AP*-time randomizing algorithm.*

Finally, we turn to the most complicated situations, in which $T \circ f$ is only required be AP time. This sort of reduction has the advantage that $f(x, s)$ needs to be a reasonable instance of Π' only for so many random strings s. The random function f must be AP-time computable. A set Γ of good inputs for f forms a domain in a natural way; see the notion of dilation in §2 in this connection.

THEOREM 1.11. *Let f be an* AP*-time-computable random function from a domain A to a domain A' with a domain Γ of good inputs dominated, with respect to f, by A'. Then for every* AP*-time algorithm M' on A' the composition $M' \circ f$ is* AP*-time.*

Theorem 1.11 follows from the much more informative Theorem 5.4. It gives rise to the following reduction notions.

DEFINITION 1.12. An AP-time reduction of a domain A to a domain A' is an AP-time-computable random function f from A to A' with a set Γ of good inputs such that A' dominates Γ with respect to f.

DEFINITION 1.13. An AP-time reduction of a search problem SP(A, W) to a search problem SP(A', W') consists of the following:

- An AP-time reduction (f, Γ) of A^+ to $(A')^+$.
- A polynomial-time-computable function g such that if $(x, s) \in \Gamma$, $x' = f(x, s)$ and w' is a W'-witness for x', then $g(x, s, w')$ is a W-witness for x.

THEOREM 1.14. AP*-time reducibility of search problems is transitive. Further, a search problem Π is solvable by an almost-total,* AP*-time algorithm if it is reducible, by an* AP*-time algorithm, to some Π' that is solvable by an* AP*-time algorithm.*

Proof. To prove the second part of the theorem suppose that Π' is solvable by an AP-time algorithm. By Theorem 1.11, Π has an AP-time solution M. By Theorem 1.2, M^* is an almost total, AP-time solution for Π. □

As an illustration of the theory of randomizing many–one reductions, we rewrite in §6 the proof of a theorem of Impagliazzo and Levin [8]; we believe that our version of the proof is easier to comprehend.

2. Preliminaries.

For the reader's convenience we recall here some definitions and facts.

DEFINITION 2.1 [2], [5] . A domain A consists of all of the following:

- An underlying set, often called A as well, whose members are strings over some finite alphabet.
- A size function that assigns to each $x \in A$ a positive integer $|x| = |x|_A$ called the *size* of x.
- A probability distribution $\mathbf{P}_A$ on A.

Remark 1. In [2] we required that the number of nonzero-probability elements of any given size be finite. The requirement seemed to be nonrestricting and useful. However, it turned out to be too restrictive in the further analysis of randomizing computations, and in this paper we remove it.

Because the elements of a domain A are strings, we can use the usual computation model based on the Turing machine. Traditional concepts of (worst-case) complexity for such functions are defined by means of the size function $|x|$, which is usually polynomially related to length of strings. For example, polynomial time would mean that there is a polynomial p such that $p(|x|)$ bounds the time needed on input x. Concepts of average-case complexity are defined by averaging with respect to the probability distribution $\mathbf{P}_A$.

As was pointed out by Levin [9] and discussed in some detail in [1], [5], the most obvious definition of the concept *polynomial time on average* has inappropriate consequences, and some care is needed to obtain a suitable definition. We use the following definition due to Levin [9], as modified in [2] to permit ∞ as a value.

DEFINITION 2.2. Let T be a function from a domain A to the set $Ra\bar{\mathcal{R}}^+$ of nonnegative reals augmented with ∞. T is linear on average if $T(x)/|x|$ has finite expectation,

$$\mathop{\mathbf{E}}_x \frac{1}{|x|} T(x) = \sum_x \mathbf{P}_A(x) \frac{1}{|x|} T(x) < \infty$$

and T is polynomial on average (AP) if it is bounded by a polynomial of a function that is linear on average. In other words, T is AP if, for some $\varepsilon > 0$,

$$\mathop{\mathbf{E}}_x \frac{1}{|x|} (Tx)^{\varepsilon} = \sum_x \mathbf{P}_A(x) \frac{1}{|x|} (Tx)^{\varepsilon} < \infty.$$

We use the convention that $0 \cdot \infty = 0$; thus an AP function can take the value ∞ but only at points of probability 0.

LEMMA 2.3 [5]. *The collection of* AP *functions over a given domain is closed under addition and multiplication.*

A (deterministic) algorithm, taking elements of a domain A as inputs, is *polynomial time on average*, or AP *time*, if its running time on input x is an AP function of x. We consider the running time to be ∞ if the algorithm fails to terminate, so that an AP-time algorithm must terminate on all inputs of positive probability.

DEFINITION 2.4. Consider a set X with a size function. A probability distribution ν on X dominates a probability distribution μ on X if there exists a function g from X to $Ra\bar{\mathcal{R}}^+$ such that g is AP with respect to X and μ and $\mu\{x\} \leq g(x) \cdot \nu\{x\}$ for all x in X.

The original notion of domination is due to Levin [9]. It is analyzed and generalized in [2] and [5].

DEFINITION 2.5 [2]. Let f be a function from a domain A to a domain B. Then B dominates A with respect to f, written $A \leq_f B$, if the ratio $\mathbf{P}_A[f^{-1}\{fx\}]/\mathbf{P}_B\{fx\}$ is AP on A.

We will use the following easy characterization of domination.

LEMMA 2.6 [2]. *Suppose that f is a function from a domain A to a domain B. Then $A \leq_f B$ if and only if $\mathbf{P}_A$ is dominated by a probability distribution ν such that the image of ν under f is $\mathbf{P}_B$, i.e., $\mathbf{P}_B\{y\} = \sum_{x \in f^{-1}\{y\}} \nu\{x\}$ for all $y \in B$.*

It will be convenient to restrict our attention to domains satisfying the following proviso.

Proviso. $|x|$ is bounded by a polynomial of the length of x.

The proviso is needed only to derive the following consequence.

COROLLARY 2.7. *If f is an* AP*-time function from a domain A to a domain B, then the function $x \mapsto |f(x)|_B$ is* AP.

Proof. The length of the string $f(x)$ cannot exceed the time needed to compute $f(x)$. Therefore, the length of $f(x)$ is an AP function of x. Now use Lemma 2.3. □

Instead of adopting the proviso, we could change the formulations of some of our theorems by explicitly requiring that the reducing functions not increase the size beyond AP. It seemed easier to adopt the proviso and to get the problem of size blowup out of the way. We did not come across any need to consider a size function that is not bounded by a polynomial of the length. However, as the following definition shows, we did come across a need to consider a size function that may be much smaller than the length.

The notion of dilation was introduced in [5] and used in [2]. The idea is to combine the probability distribution on instances and the probability distribution on coin flips into one probability distribution. For this purpose we introduce a new domain consisting of pairs (x, s), where x ranges over instances of the search problem under consideration, i.e., over elements of a given domain A, and s ranges over the possible finite sequences of coin flips used by some algorithm (in its terminating computations on input x). The size of (x, s) is taken to be $|x|_A$, not (as one might at first guess) $|x|_A + |s|$; this is done so that the complexity of computations is measured relative to the size of the instance, not the number of coin flips. (If we used $|x|_A + |s|$, then a very inefficient computation could be made to look efficient—e.g., linear time—by appending a lot of unnecessary coin flips to make $|x|_A + |s|$ greater than half the running time.) The probability of (x, s) is, except for a normalization factor, the probability $\mathbf{P}_A(x) \cdot 2^{-|s|}$ for independent choices of x and s. The normalization factor, i.e., the lack of actual independence, can be intuitively understood this way: Given x, the algorithm starts to flip coins (independent of x) until the computation terminates, having used random string s. But it may happen—and the probability of this *does* depend on x—that the algorithm does not terminate, in which case we restart the algorithm with a new, independent sequence of coin flips. This restarting leads to the denominator in the probability clause of the following definition.

DEFINITION 2.8. A dilation of a domain A is a domain $\Delta \subset A \times \{0, 1\}^*$ satisfying the following conditions, where $\Delta(x) = \{s : (x, s) \in \Delta\}$:

- For every x no member of $\Delta(x)$ is a proper prefix of another member of $\Delta(x)$.
- For every x with $\mathbf{P}_A(x) > 0$, $\Delta(x) \neq \emptyset$.
- $|(x, s)|_\Delta = |x|_A$.
- $\mathbf{P}_\Delta(x, s) = \mathbf{P}_A(x) 2^{-|s|} / \sum_{t \in \Delta(x)} 2^{-|t|}$.

This definition is more general than the definition of dilation used in [2] because we do not require $\sum_{s \in \Delta(x)} 2^{-|s|} = 1$. This sum occurs in the denominator of our definition of $\mathbf{P}_\Delta$ to ensure that $\mathbf{P}_\Delta$ is indeed a probability measure, i.e., that the total probability of Δ is 1. We say that a dilation Δ_1 of A is a *subdilation* of a dilation Δ_2 of A if $\Delta_1 \subseteq \Delta_2$.

DEFINITION 2.9. Let Δ be a dilation of A. The function

$$U_{\Delta(x)} = \frac{1}{\sum_{s\in\Delta(x)} 2^{-|s|}}$$

is the *rarity function* of D. The dilation Δ is almost total if $U_{\Delta(x)} = 1$ for every $x \in A$ of positive probability. This means that if we repeatedly flip a fair coin to produce a string of 0's and 1's, then with probability 1 we shall eventually obtain a string in $\Delta(x)$.

The notion of randomizing (coin-flipping) algorithms motivates a useful notion of random function [2].

DEFINITION 2.10. A *random function* on a domain A is a function f on a dilation D_f of A. Such a random function f is almost total if the dilation D_f is almost total and the probability, with respect to D_f, that the value of f is finite equals 1.

Composition of randomizing algorithms motivates composition of random functions.

DEFINITION 2.11 [2]. Suppose that f is a random function from a domain A to a domain B such that $\mathbf{P}_B(f(x,s)) > 0$ whenever $\mathbf{P}_A(x) > 0$, and let g be a random function on B. The *composition* $g \circ f$ of f and g is the random function h on A such that for every $x \in A$ we have the following:

- $D_h(x) = \{st : s \in D_f(x) \text{ and } t \in D_g(f(x,s))\}$.
- If $s \in D_f(x)$ and $t \in D_g(f(x,s))$, then $h(x,st) = g(f(x,s),t)$.

It is easy to see that the composition of almost-total random functions is almost total.

3. Randomizing algorithms.

A randomizing algorithm on a domain A can be formalized as a Turing machine M that takes as input an instance $x \in A$ and has access to an auxiliary read-only tape, called the *random tape*, containing an infinite sequence r of random bits (0's and 1's). The random tape is bounded on the left and unbounded on the right; its head can move only to the right. The set $\{0,1\}^\infty$ of all possible r's is endowed with the probability measure λ that is the product measure derived from the uniform measure on $\{0,1\}$. Both the measure and M's mode of access to the random tape can be described informally by saying that M is allowed to flip a fair coin at various stages of its computation and that these coin flips are independent.

Consider the computation of a fixed randomizing algorithm M with a fixed input x but with varying sequence $r \in \{0,1\}^\infty$. For each r let Read(r) (or Read$_{M,x}(r)$ if necessary for clarity) be the initial segment of r that is actually read during the computation. If the computation terminates, then Read(r) is finite; if not, then Read(r) may be finite or equal to r. Because the computation with M and x fixed depends only on Read(r), it is clear that if $r' \in \{0,1\}^\infty$ has Read(r) as an initial segment, then the computation using r' is the same as that using r and, in particular, Read(r') = Read(r). Thus the set

$$\mathrm{R}(x) = \mathrm{R}_M(x) = \{\mathrm{Read}_{M,x}(r) : r \in \{0,1\}^\infty\}$$

has the property that no member of R(x) is a proper initial segment of another. We define the probability measure $\rho = \rho_{M,x}$ on R(x) to be the image, under the function Read$_{M,x}$, of the standard measure λ on $\{0,1\}^\infty$. For any $E \subseteq R(x)$

$$\rho(E) = \lambda\{r : \mathrm{Read}(r) \in E\}.$$

Each R(x) is the disjoint union of the sets RF(x) and RI(x) consisting of the finite and infinite strings in R(x), respectively. The measure ρ gives the probability $2^{-|s|}$ to each $s \in$ RF(x) and agrees on subsets of RI(x) with λ. Every $r \in \{0,1\}^\infty$ either is in RI(x) or has an initial segment in RF(x), namely Read(r). In [2] randomizing algorithms were assumed to terminate for all x with $\mathbf{P}_A(x) > 0$ and all r; this ensured that if $\mathbf{P}_A(x) > 0$, then RI(x) is empty and every $r \in \{0, 1\}^\infty$ has an initial segment in $\mathrm{RF}(x)$ and therefore $\mathrm{RF}(x)$ is finite. In the present paper we consider algorithms that may fail to terminate even on inputs of positive probability, so that we must consider both $\mathrm{RF}(x)$ and $\mathrm{RI}(x)$; furthermore, $\mathrm{RF}(x)$ may well be infinite.

If $s \in \{0, 1\}^*$ is a finite sequence with no proper initial segment in $\mathrm{RF}(x)$, then s occurs (with probability $2^{-|s|}$) as the random string read by some stage in a computation of M on x. If, furthermore, $s \notin \mathrm{RF}(x)$, then this computation will read at least one additional random bit, which is equally likely to be 0 or 1. If, on the other hand, $s \in \mathrm{RF}(x)$, then the computation will read no additional random bits. Writing "s is a prefix" for the event that the random string actually read by M has s as an initial segment, we have (for future reference) the following lemma.

LEMMA 3.1. *The distribution ρ satisfies the equation*

$$\rho(s \text{ is a prefix}) = \begin{cases} 2^{-|s|} & \text{if } s \text{ has no proper initial segment in } \mathrm{RF}(x), \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, this equation completely determines ρ.

Proof. The proof is obvious. □

We restrict our attention to randomizing algorithms M such that $\mathrm{RF}_M(x) \neq \emptyset$ whenever $\mathbf{P}_A(x) > 0$. It follows that the set

$$\mathrm{RF}_M = \{(x, s) : x \in A \text{ and } s \in \mathrm{RF}_M(x)\}$$

forms a dilation of the domain A.

DEFINITION 3.2. The random function F_M on A computed by M is the deterministic function on $D_F = \mathrm{RF}_M$ with $F(x, s)$ equal to the output of M on (x, s).

We write $\mathrm{Time}(x, r)$ or $\mathrm{Time}_M(x, r)$ for the time taken by the computation of M on x and r; if this computation does not terminate, then $\mathrm{Time}(x, r) = \infty$. $\mathrm{Time}(x, r)$ depends only on x and $\mathrm{Read}(r)$ since the unread part of r cannot influence the computation time.

DEFINITION 3.3. The *restrained time function* of a randomizing algorithm M on A is the random function $T(x, s) = T_M(x, s)$ on A such that $D_T = \mathrm{RF}_M$ and

$$\mathrm{Time}(x, r) = T(x, \mathrm{Read}(r))$$

for every r with a finite $\mathrm{Read}(r)$ part.

Time_M is not the only function on $A \times \{0, 1\}^\infty$ that we have to consider. In this paper a *functional* is a measurable function $\mathcal{F}$ from $A \times \{0, 1\}^\infty$ to $Ra\bar{\mathcal{R}}^+$ such that every finite value of $\mathcal{F}$ is a positive integer. We say that a functional $\mathcal{F}$ is *continuous* if it is continuous with respect to the product topology on $A \times \{0, 1\}^\infty$ and the natural topology on $Ra\bar{\mathcal{R}}^+$. The continuity implies that for every x and every $r \in \{0, 1\}^\infty$ with $\mathcal{F}(x, r) < \infty$ there exists a finite initial segment s of r such that $\mathcal{F}(x, r) = \mathcal{F}(x, r')$ for every r' with prefix s. The shortest initial segment s with this property will be denoted $\mathrm{Read}_{\mathcal{F},x}(r)$. Define the *restrained version* of a functional $\mathcal{F}$ to be the random function F on A such that $D_F = \{(x, \mathrm{Read}_{\mathcal{F},x}(r)) : \mathcal{F}(x, r) < \infty\}$ and $F(x, \mathrm{Read}(r)) = \mathcal{F}(x, r)$. Thus Time_M is a continuous functional, and T_M is the restrained version of Time_M.

DEFINITION 3.4. A functional $\mathcal{F}$ on A is almost total if for every $x \in A$ of positive probability

$$\lambda\{r \in \{0, 1\}^\infty : \mathcal{F}(x, r) < \infty\} = 1.$$

LEMMA 3.5. *A functional is almost total if and only if its restrained version is so.*

Proof. The proof is clear. □

DEFINITION 3.6. An almost-total functional $\mathcal{F}$ on a domain A is AP if there is a positive ε such that

$$\mathop{\mathrm{E}}_{x}\mathop{\mathrm{E}}_{r}\left(\frac{1}{|x|}\mathcal{F}(x,r)^{\varepsilon}\right) < \infty,$$

where E_x and E_r mean expectation with respect to x and r, the relevant probability distributions being $\mathbf{P}_A$ and λ.

The proof of Lemma 2.3 works for almost-total functionals.

COROLLARY 3.7. *The collection of* AP *almost-total functionals over a given domain is closed under addition and multiplication.*

By definition of ρ the expectation of a continuous almost-total functional with respect to λ and the expectation of its restrained version with respect to ρ agree for every x.

LEMMA 3.8. *An almost-total, continuous functional $\mathcal{F}$ is* AP *if and only if its restrained version F is* AP *as a deterministic function on D_F.*

Proof.

$$\mathop{\mathrm{E}}_{x}\mathop{\mathrm{E}}_{r}\left(\frac{1}{|x|}\mathcal{F}(x,r)^{\varepsilon}\right) = \mathop{\mathrm{E}}_{x}\mathop{\mathrm{E}}_{s}\left(\frac{1}{|x|}F(x,s)^{\varepsilon}\right) = \mathop{\mathrm{E}}_{(x,s)\in D_F}\frac{1}{|x|}F(x,s)^{\varepsilon},$$

where the expectation E_s is with respect to ρ. □

Not all almost-total functionals of interest to us are continuous. This is why we have defined directly when an almost-total functional is AP.

DEFINITION 3.9. A randomizing algorithm M is almost total if the functional Time_M is so.

By Lemma 3.5, M is almost total if and only if the restrained time function T_M is almost total.

DEFINITION 3.10. An almost-total randomizing algorithm M is AP time if the continuous functional Time_M is AP.

The intuitive content of this definition of an almost-total, AP-time algorithm is that for each input x with $\mathbf{P}_A(x) > 0$ the algorithm will almost surely (i.e., with probability 1 with respect to r) terminate and will usually (with respect to x and r) require only a reasonable amount of time.

Now we turn our attention to the main notion of this section, the notion of AP-time algorithms. We consider randomizing algorithms that terminate with a not too small probability (rather than almost surely) and that, in fact, require reasonable time with a not too small probability. To avoid confusion we remark that we do not require *all* the terminating computations to take reasonable time, only *enough* of them to have reasonable probability.

DEFINITION 3.11. Let F be a random function on a domain A. F is AP on A if there is a subdilation Γ of D_F such that both of the following conditions hold:

1. The rarity function U_Γ is AP.
2. For some $\varepsilon > 0$

$$\sum_{(x,s)\in\Gamma}\frac{1}{|x|}\mathbf{P}_A(x)2^{-|s|}F(x,s)^{\varepsilon} < \infty.$$

Condition 1 formalizes the part of the informal requirement above about *enough* terminating computations having reasonable probability. The meaning of condition 2 is clarified below (Lemma 3.12). Notice that the statement "F is AP on A" is, in general, weaker than the statement "F is AP as a deterministic function on D_F." We shall not need to generalize the definition of an AP random function to functionals.

If Γ satisfies these conditions, we call it a set of *good inputs* for F. The dilation of A formed by Γ may be called the *domain of good inputs*. The fact that F is AP means intuitively that reasonably often F will be defined by virtue of a good input and that these values of F are reasonably small on average. (F may also be defined on some inputs that are not good, and the resulting values of F need not be small at all.) It is not required that good inputs be easily recognizable in any way.

LEMMA 3.12. *Let F be a random function on a domain A, and let Γ be a subdilation of D_F such that the rarity function U_Γ is* AP. *Then the following conditions are equivalent*:

1. *F is* AP *on A with Γ as a set of good inputs.*
2. *The restriction of F (viewed as a deterministic function on D_F) to Γ is* AP, *i.e., there exists $\varepsilon > 0$ such that*

$$\mathop{\mathbf{E}}_{(x,s)\in\Gamma} \frac{1}{|x|} F(x,s)^{\varepsilon} = \sum_{(x,s)\in\Gamma} \frac{1}{|x|} \mathbf{P}_A(x) 2^{-|s|} U_\Gamma(x) F(x,s)^{\varepsilon} < \infty.$$

Proof. Since $U_\Gamma(x) \geq 1$, condition 2 implies condition 1. Given condition 1, multiply and divide each summand in part 2 of the definition of AP by $U(x) = U_\Gamma(x)$ to produce $\mathbf{P}_\Gamma$ as a factor; we obtain

$$\mathop{\mathbf{E}}_{x,s} \frac{1}{|x|} \frac{F(x,s)^{\varepsilon}}{U(x)} < \infty,$$

where the expectation over (x, s) is with respect to $\mathbf{P}_\Gamma$. Also, because U is AP on A, we have for some $\delta > 0$

$$\mathop{\mathbf{E}}_{x,s} \frac{1}{|x|} U(x)^{\delta} = \mathop{\mathbf{E}}_{x} \frac{1}{|x|} U(x)^{\delta} < \infty,$$

where the expectation over x is with respect to $\mathbf{P}_A$ and we used the fact that $\mathbf{P}_A$ is the projection of $\mathbf{P}_\Gamma$. Adding the last two inequalities and using the fact that any weighted geometric mean of two positive numbers is less then their sum, we find

$$\mathop{\mathbf{E}}_{x,s} \left[\left(\frac{1}{|x|} \frac{F(x,s)^{\varepsilon}}{U(x)} \right)^{\delta/(1+\delta)} \cdot \left(\frac{1}{|x|} U(x)^{\delta} \right)^{1/(1+\delta)} \right] < \infty.$$

Algebraically simplifying the expression in brackets, we get

$$\mathop{\mathbf{E}}_{x,s} \left[\frac{1}{|x|} F(x,s)^{\varepsilon\delta/(1+\delta)} \right] < \infty,$$

which is the desired inequality with $\varepsilon\delta/(1+\delta)$ in place of ε. □

COROLLARY 3.13. *An almost-total continuous functional $\mathcal{F}$ is* AP *if and only if its restrained version F is* AP *with D_F as a set of good inputs.*

Proof. Use Lemmas 3.8 and 3.12. □

DEFINITION 3.14. A randomizing algorithm M on A is AP time if the restrained time function T_M is AP. A set of good inputs for T is also called a set of good inputs for M.

In the case of an almost-total randomizing algorithm M we have now two definitions of polynomiality on average. It is easy to see that they are equivalent.

COROLLARY 3.15. *For every almost-total randomizing algorithm M the following are equivalent*:

- *The functional* Time_M *is* AP.
- *The random function* T_M *is* AP *on the input domain.*

Proof. Use Lemma 3.8. □

DEFINITION 3.16. A randomizing algorithm M is an AP-time solution (respectively, almost-total, AP-time solution) to search problem $\mathrm{SP}(A, W)$ if it is AP time (respectively, almost total and AP time) and every terminating computation of M with input $x \in A$ (and arbitrary random string r) produces as output a witness w with $(x, w) \in W$.

It might seem more natural to require appropriate output only for enough, but not all, of the terminating computations, but our definition is somewhat simpler and is essentially equivalent.

LEMMA 3.17. *Every* AP-*time algorithm* M *for a search problem* $\Pi = \mathrm{SP}(A, W)$ *that produces correct outputs on its good arguments can be converted to an* AP-*time solution* M' *for* Π.

Proof. Append to M instructions, saying that when M terminates, M' should check whether the output w satisfies $W(x, w)$ and M' should terminate if and only if the check succeeds. Because our definition of "search problem" required W to be polynomial-time computable, the time used by M' is bounded by a polynomial of the sum of $|x|$ and the time used by M, so that replacing M with M' does not ruin complexity estimates of the sort we are considering. □

LEMMA 3.18. *Suppose that a search problem* Π *has an* AP-*time solution. Then the length of the shortest witness is a* (*deterministic*) AP *function.*

Proof. Let M be an AP-time solution to Π with a set Γ of good inputs. Recall that the rarity function U of Γ is the reciprocal of $\rho_{M,x}(\Gamma(x)) = \sum_{s\in\Gamma(x)} 2^{-|s|}$. Write $l(x)$ for the length of the shortest witness for x. Clearly, $T(x, s) \geq l(x)$ for all $(x, s) \in \Gamma$ because it takes time $l(x)$ just to write a witness on the output tape. For every $\varepsilon > 0$, we have

$$\begin{aligned}\sum_{x\in A} \frac{1}{|x|} l(x)^{\varepsilon} \mathbf{P}_A(x) &= \sum_{x\in A} \frac{1}{|x|} l(x)^{\varepsilon} \mathbf{P}_A(x) U(x) \rho(\Gamma(x)) \\ &\leq \sum_{(x,s)\in\Gamma} \frac{1}{|x|} T(x, s)^{\varepsilon} \mathbf{P}_A(x) U(x) 2^{-|s|}.\end{aligned}$$

Now use the fact that T is AP, and apply Lemma 3.12. □

4. Iteration. It is well known that a randomizing algorithm that solves a search problem with a certain probability can be iterated, by using independent sequences of coin flips, to obtain a much higher success probability. A polynomial number of iterations suffices to improve success probabilities from as low as a reciprocal of a polynomial of $|x|$ to as high as exponentially (relative to $|x|$) close to 1. For average-case complexity the situation is even better. We can start with a success probability whose reciprocal is not polynomially bounded but only polynomial on average, and we can iterate the algorithm to obtain a success probability 1 without increasing the time beyond AP. The main result in this section will establish a precise version of this claim and will show that it is optimal.

Let M be a randomizing algorithm on a domain A that may or may not terminate on a given input. In our application M is an algorithm for a search problem $\mathrm{SP}(A, W)$ whose successful computations produce desired witnesses. For simplicity we assume in this section that every termination of M is successful. It is easy to remove this assumption. Also, see Lemma 3.17.

For technical reasons we start with what we call a *perpetual iteration* M^{∞} of a given randomizing algorithm M. Given an input (x, r), where $x \in A$ and r is an infinite random sequence, M^∞ simulates many computations of M on inputs (x, s) for the same x but different random bit strings s (disjoint substrings of r); these computations of M will be called subcomputations of M^∞. M^∞ is called a perpetual iteration because it never halts. Even after some subcomputations have terminated, M^∞ continues to run other subcomputations and to start new subcomputations and it continues to allocate its random bits to subcomputations. The corresponding *iteration* M^* of M is like M^∞ except that M^* halts if and when one of the subcomputations terminates. The advantage of working with M^∞ rather than M^* is that in a computation of M^∞ every subcomputation runs either forever or until it terminates according to the rules of M, whereas in a computation of M^* a subcomputation can stop unnaturally because another subcomputation terminated and thus stopped the whole computation of M^*.

Presupposing a fixed perpetual iteration M^∞ and a fixed element $x \in A$, for each sequence $r \in \{0, 1\}^\infty$, let $\hat{s}_j = \hat{s}_j(r)$ be the subsequence of r that M^∞ with input (x, r) passes to subcomputation number j, and let $s_j = s_j(r)$ be the subsequence of $\hat{s}_j$ that is actually read by subcomputation number j. Here are some examples of perpetual iteration.

Example 1. In *eager perpetual iteration* stage n of the computation of M^∞ consists of one step in the first, second, ..., nth subcomputations, in that order. So one new subcomputation is started at each stage, and each previously started subcomputation is carried one step further. Also, during stage n, M^∞ reads n new bits from r and distributes them, in order, to these n subcomputations. Thus each projection $\hat{s}_j$ is infinite. Note that this projection operation on $\{0, 1\}^\infty$ does not depend on M or on its input x.

Example 2. *Lazy perpetual iteration* is similar to the eager perpetual iteration except that it distributes random bits only as the subcomputations ask for them. There is no longer an a priori defined infinite sequence $\hat{s}_j$ for each subcomputation; in fact, $\hat{s}_j = s_j$. To decide which subcomputation will receive a particular bit from r it is no longer sufficient to know the position of that bit in r; one needs to know both the input x and the preceding bits of r.

Example 3. Another variation of the eager perpetual iteration is obtained as follows. Fix a polynomially bounded injection $P(j, t)$ from pairs of positive integers to positive integers, and reserve random bits $P(j, 1)$, $P(j, 2)$, $P(j, 3)$, ... for the jth subcomputation. This perpetual iteration is executed more naturally by a random-access machine than by a Turing machine.

DEFINITION 4.1. A perpetual iteration M^∞ is *polynomially fair* if the number $m(j, t)$ of steps of M^∞ required to achieve the tth step or termination (whichever occurs first) of subcomputation j is bounded by a polynomial of $j + t$.

It follows that new subcomputations are started fairly often: the number $m(j, 1)$ of steps of M^∞ required to start the jth computation is bounded by a polynomial of j. Notice also that each subcomputation gets enough random bits to proceed until termination or forever.

Remark 2. To deal with parallel iterations (in a sense they are more natural), add to the definition of polynomial fairness the requirement that the number of subcomputations started during the first n steps of M^∞ is bounded by a polynomial of n.

We call a perpetual iteration M^∞ *careful* if each subcomputation gets a sequence of fair coin flips and the sequences for different subcomputations are independent (provided that M^∞ gets a sequence of fair coin flips). We formulate this definition more formally. Recall the probability distribution λ of §3.

DEFINITION 4.2. A perpetual iteration M^∞ is *careful* if for each input x of positive probability the function

$$r \mapsto (s_1, s_2, \ldots) \text{ from } \{0, 1\}^\infty \text{ to } (\{0, 1\}^* \cup \{0, 1\}^\infty)^\infty$$

sends λ to the product measure on $(\{0, 1\}^* \cup \{0, 1\}^\infty)^\infty$ induced by the measure $\rho_{M,x}$ on $\{0, 1\}^* \cup \{0, 1\}^\infty$.

LEMMA 4.3. *M^∞ is careful if and only if for any input $x \in A$ and any finite list $t_1, \ldots, t_k$ of binary strings the probability*

$$\lambda\{r : t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \ldots, k\}$$

is 0 if some t_j has a proper initial segment in $\mathrm{RF}_M(x)$ and is 2^{-n} otherwise, where n is the sum of the lengths of the t_j's.

Proof. Use Lemma 3.1. □

THEOREM 4.4. *Let M^∞ be a careful and polynomially fair perpetual iteration of M. Then M is* AP *time if and only if the iteration M^* is* AP *time and almost total.*

Proof. Suppose that M is AP time, i.e., the restrained time function $T = T_M$ is AP. Let Γ and ε be as in the definition of the AP random function for T, and let $U = U_\Gamma$ be the rarity function of Γ. We first check that, for any input x of positive probability in A, M^* terminates with probability 1 (with respect to r). In fact, we show more, namely, that for almost all r some s_j is good, i.e., belongs to $\Gamma(x)$. (If $s_j \in \Gamma(x)$ and $\mathbf{P}_A(x) > 0$, then the computation of M on (x, s_j) terminates, so that we are indeed proving more than originally claimed.) The event (= set of r's such) that *no* s_j is in $\Gamma(x)$ is the intersection of the infinitely many events $s_j \notin \Gamma(x)$ $(j = 1, 2, 3, \ldots)$, which are independent and have probability $1 - \rho(\Gamma(x)) = 1 - (1/U(x))$. So their intersection has probability 0, as required.

We are now ready to estimate the computation time $T^*(x, r)$ of M^* on input (x, r) and to prove that M^* is AP time. For any $x \in A$ of positive probability and any $r \in \{0, 1\}^\infty$ let $k = k(x, r)$ be the smallest positive integer j with $s_j \in \Gamma(x)$. We saw in the preceding paragraph that for all $x \in A$ of positive probability such a k exists for almost all r. (Define $k(x, r) = \infty$ on the measure-zero set of pairs (x, r) where no such k exists.) Let $T' = T'(x, r)$ be the time taken by the computation of M with input $(x, s_{k(x,r)})$, which is the kth subcomputation of M^∞ on input (x, r). Since M^∞ is polynomially fair, the computation time T^* of M^* is bounded by a polynomial of $(k + T')$. By Corollary 3.7 it suffices to prove that $k(x, r)$ and $T'(x, r)$ are AP.

First we treat k.[1] We begin with a well-known and quite general observation. Suppose an experiment succeeds with probability $p > 0$, and suppose we make many independent repetitions of this experiment until one, say, the kth, succeeds. Then k has expectation

$$\sum_{i=1}^{\infty} i(1-p)^{i-1} p = \frac{1}{p}.$$

In the situation at hand the experiments are the subcomputations of $M^\infty(x, r)$ (with random r), which are independent by carefulness, "success" means that the string read by the subcomputation is good, and thus $1/p = U(x)$, $k = k(x, r)$, and $\mathbf{E}_r\, k(x, r) = U(x)$.

Choose positive $\delta < 1$ witnessing that U is AP:

$$\mathop{\mathbf{E}}_x \left(\frac{1}{|x|} U(x)^\delta \right) < \infty.$$

We check that this δ also witnesses that k is AP. The function $f(y) = y^\delta$ is a concave function on the open real interval $I = (0, \infty)$. By Jensen's inequality (see Corollary 7.2 below),

$$\mathop{\mathbf{E}}_x \mathop{\mathbf{E}}_r \left(\frac{1}{|x|} k(x, r)^\delta \right) = \mathop{\mathbf{E}}_x \left(\frac{1}{|x|} \mathop{\mathbf{E}}_r \left[k(x, r)^\delta \right] \right)$$

[1] Curiously, the useful functional $k(x, r)$ is not continuous.

$$\leq \mathop{\mathbf{E}}_{x}\left(\frac{1}{|x|}\left[\mathop{\mathbf{E}}_{r}(k(x,r))\right]^{\delta}\right)$$
$$= \mathop{\mathbf{E}}_{x}\left(\frac{1}{|x|}U(x)^{\delta}\right) < \infty.$$

Next we treat $T'(x,r)$. Clearly, $T'(x,r) = T(x,s_k)$, where $k = k(x,r)$. Recall that a positive ε, one of the witnesses that M is AP, was chosen. We begin by computing the expectation of $T'(x,r)^{\varepsilon} = T(x,s_k)^{\varepsilon}$ with respect to r.

$$\mathop{\mathbf{E}}_{r}(T(x,s_k)^{\varepsilon}) = \sum_{j=1}^{\infty}\mathbf{P}(k(x,r)=j)\cdot\mathop{\mathbf{E}}_{r}\left(T(x,s_j)^{\varepsilon} \mid k(x,r)=j\right).$$

The event $k(x,r) = j$ whose probability occurs here is the intersection of the event $s_j \in \Gamma(x)$ and all events $s_i \notin \Gamma(x)$ with $i < j$. Since M^{∞} is careful, these are independent, so the product $(1 - 1/U(x))^{j-1} \cdot 1/U(x)$ of their probabilities equals $\mathbf{P}(k(x,r) = j)$. The conditional expectation of $T(x,s_j)^{\varepsilon}$ relative to this intersection of events equals the conditional expectation of $T(x,s_j)$ relative to $s_j \in \Gamma(x)$ since, by the carefulness of M^{∞}, $T(x,s_j)^{\varepsilon}$ is independent of the events $s_i \notin \Gamma(x)$ with $i \neq j$. Further,

$$\mathop{\mathbf{E}}_{r}\left(T(x,s_j)^{\varepsilon} \mid s_j \in \Gamma(x)\right) = \mathop{\mathbf{E}}_{s\in\Gamma(x)} T(x,s)^{\varepsilon} = \sum_{s\in\Gamma(x)} T(x,s)^{\varepsilon}\cdot 2^{-|s|}\cdot U(x).$$

Thus we have

$$\begin{aligned}\mathop{\mathbf{E}}_{r}(T'(x,r)^{\varepsilon}) &= \sum_{j=1}^{\infty}\left(1-\frac{1}{U(x)}\right)^{j-1}\cdot\frac{1}{U(x)}\cdot\sum_{s\in\Gamma(x)}\left(T(x,s)^{\varepsilon}\cdot 2^{-|s|}\cdot U(x)\right)\\ &= \left(\sum_{j=1}^{\infty}\left(1-\frac{1}{U(x)}\right)^{j-1}\right)\cdot\sum_{s\in\Gamma(x)}\left(T(x,s)^{\varepsilon}\cdot 2^{-|s|}\right)\\ &= U(x)\cdot\sum_{s\in\Gamma(x)}\left(T(x,s)^{\varepsilon}\cdot 2^{-|s|}\right).\end{aligned}$$

Recall that Γ, U, and ε were chosen to witness that M is AP. Thus U is AP and the factor multiplying it is AP as well. (In fact, this factor is average linear.) So the product is AP. Choose a positive $\delta < 1$ to witness this. Using Jensen's inequality again, we have

$$\begin{aligned}\mathop{\mathbf{E}}_{x}\mathop{\mathbf{E}}_{r}\left(\frac{1}{|x|}T'(x,r)^{\varepsilon\delta}\right) &= \mathop{\mathbf{E}}_{x}\left(\frac{1}{|x|}\mathop{\mathbf{E}}_{r}\left[T'(x,r)^{\varepsilon\delta}\right]\right)\\ &\leq \mathop{\mathbf{E}}_{x}\left(\frac{1}{|x|}\left[\mathop{\mathbf{E}}_{r}(T'(x,r)^{\varepsilon})\right]^{\delta}\right) < \infty.\end{aligned}$$

Thus we have the desired estimate for T', which completes the proof of one direction of the theorem.

To prove the converse assume M^* is an almost-total, AP-time algorithm. We must produce a Γ and a positive ε witnessing that M is AP. Fix a positive $\varepsilon < 1$ witnessing that T^* is AP:

$$\mathop{\mathbf{E}}_{x}\mathop{\mathbf{E}}_{r}\left(\frac{1}{|x|}T^*(x,r)^{\varepsilon}\right) < \infty.$$

That is,

$$\mathop{\mathbf{E}}_x\left(\frac{1}{|x|}F(x)\right) < \infty, \text{ where } F(x) = \mathop{\mathbf{E}}_r\left(T^*(x,r)^{\varepsilon}\right).$$

Observe that, quite generally, if X is a nonnegative random variable with expectation m, then by Markov's inequality $\mathbf{P}(X \le 2m) \ge 1/2$. (Indeed, if $\mathbf{P}(X \le 2m) = p$, then

$$\begin{aligned} m = \mathbf{E}(X) &= p \cdot \mathbf{E}(X \mid X \le 2m) + (1-p)\,\mathbf{E}(X \mid X > 2m) \\ &\ge p \cdot 0 + (1-p) \cdot 2m, \end{aligned}$$

so that $1 - p \le 1/2$, as claimed.)

Applying this observation to the random variable $T^*(x,r)^{\varepsilon}$, for a fixed x and random $r \in \{0,1\}^{\infty}$ we find that the set

$$\Gamma'(x) = \{r \in \{0,1\}^{\infty} : T^*(x,r)^{\varepsilon} \le 2F(x)\}$$

has probability $\ge 1/2$. For each $r \in \Gamma'(x)$ let $s(x,r)$ be the string of random bits actually read by the subcomputation of M^* on (x,r) that produced a witness for x. We define

$$\Gamma = \left\{(x, s(x,r)) : x \in A \text{ and } r \in \Gamma'(x)\right\},$$

so that $\Gamma(x) = \{s(x,r) : r \in \Gamma(x)\}$, and we claim that this Γ and the ε fixed above are as required by the definition of M being AP. By the definition of $s(x,r)$, Γ forms a dilation of A.

To check that the rarity function $U = U_{\Gamma}$ is AP, recall that for $r \in \Gamma(x)$, $T^*(x,r) \le (2F(x))^{1/\varepsilon}$. Therefore, the number $q = q(x)$ of subcomputations started by M^* on input (x,r) is bounded by a polynomial of $F(x)$. Since $F(x)$ is AP (in fact, linear on average), $q(x)$ is AP. We have

$$\begin{aligned} \frac{1}{2} &= \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} \mathbf{P}\{r \in \Gamma'(x) : s(x,r) = s \text{ and} \\ &\qquad\qquad \text{subcomputation } j \text{ is the one that succeeds}\} \\ &\le \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} \mathbf{P}\{r \in \Gamma'(x) : \text{subcomputation } j \text{ reads exactly } s\} \\ &= \sum_{s \in \Gamma(x)} \sum_{j=1}^{q} 2^{-|s|} = q(x) \cdot \frac{1}{U(x)}. \end{aligned}$$

So $U(x)$ is bounded by the AP function $2q(x)$.

To prove the convergence condition in the definition of T as being AP, recall that each s in $\Gamma(x)$ is $s(x,r)$ for some $r \in \Gamma'(x)$. Hence $T(x,s)^{\varepsilon} \le T^*(x,r)^{\varepsilon} \le 2F(x)$. So

$$\sum_{s \in \Gamma(x)} T(x,s)^{\varepsilon} 2^{-|s|} \le 2F(x) \cdot \sum_{s \in \Gamma(x)} 2^{-|s|} \le 2F(x),$$

which implies the desired convergence because F is linear on average. This completes the proof of Theorem 4.4. □

It is easy to see that the perpetual iterations of Examples 1–3 are polynomially fair and that the perpetual iterations of Examples 1 and 3 are careful as well. It is not quite obvious that the lazy perpetual iteration is careful. In the case of the lazy perpetual iteration, some information about a given subcomputation j can be learned from watching other subcomputations. Imagine, for example, that the first m random bits received by subcomputation $j > 1$ happened to have the same values as the first m random bits received by subcomputation 1 and that subcomputation 1 requested another random bit. We know then that subcomputation j will request another random bit as well.

THEOREM 4.5. *Suppose that a perpetual iteration M^∞ has the following properties:*

- *M^∞ starts infinitely many subcomputations on every input (x, r).*
- *No subcomputation of M^∞ is ever stalled: either it terminates or else it executes infinitely many steps.*
- *M^∞ serves random bits on a first come, first served basis.*

Then M^∞ is careful.

Proof. Fix an arbitrary $x \in A$ of positive probability. Recall that $s_j = s_j(r)$ is the string of random bits passed to the jth subcomputation. By Lemma 4.3 we must show that for any input $x \in A$ and any list $t_1, \dots, t_k$ of binary strings the probability

$$\lambda\{r : \ t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \dots, k\}$$

is 0 if some t_j has a proper initial segment in $\mathrm{RF}_M(x)$ and that otherwise it is 2^{-n}, where n is the sum of the lengths of the t_j's.

The first half of this is clear, because if t_j has a proper initial segment in $\mathrm{RF}_M(x)$, then no subcomputation can read all of t_j, and so $s_j \neq t_j$. To prove the second half consider a list $t_1, \dots, t_k$ of cumulative length n and with no proper initial segments of any t_j being in $\mathrm{RF}_M(x)$. Note that $s_j(r)$ cannot, for any r, be a proper initial segment of t_j because any finite s_j is in $\mathrm{RF}_M(x)$, whereas no t_j has a proper initial segment in $\mathrm{RF}_M(x)$.

Call a finite binary string q, which we regard as a possible initial segment of $r \in \{0, 1\}^\infty$, *pertinent* (to the fixed list $t_1, \dots, t_k$) if, when M^∞ reads q from its random tape, the strings $s_j(q)$ that it passes to subcomputations $j = 1, \dots, k$ are consistent with the t_j's. ("Consistent" means one is an initial segment, not necessarily proper, of the other.) For pertinent q write $s'_j(q)$ for $s_j(q)$ or t_j, whichever is an initial segment of the other; this is the part of t_j already read by subcomputation j when M^∞ has read q. Write $m(q)$ for the sum of the lengths of the strings $s'_j(q)$. We have $0 \le m(q) \le n$.

Let E denote the event

$$\{r : \ t_j \text{ is a prefix of } s_j(r) \text{ for all } j = 1, \dots, k\}.$$

Recall that our objective is to prove that $\lambda(E) = 2^{-n}$. Call $q \in \{0, 1\}^*$ *strange* if it is pertinent and $\lambda[E \mid q \text{ is a prefix of } r] \neq 2^{-n+m(q)}$. Observe that since the empty string e is pertinent and prefixes every r, e is strange if and only if $\lambda(E) \neq 2^{-n}$, which is the negation of what we want to prove. We therefore assume e is strange and attempt to deduce a contradiction.

LEMMA 4.6. *If q is strange, then so is at least one of its one-bit extensions $q0$ or $q1$.*

Proof. Consider a computation of M^∞ (on our fixed input x) that has read q (and passed its bits to the appropriate subcomputations).

Case 1. The next request for a bit comes from subcomputation i with $i > k$ or with $i \le k$ and $s'_i(q) = t_i$. Then both $q0$ and $q1$ are pertinent, and $s'_j(q0) = s'_j(q1) = s'_j(q)$ for all $j = 1, \dots, k$, so that $m(q0) = m(q1) = m(q)$. Because q is strange,

$$2^{-n+m(q)} \neq \lambda[E \mid q \text{ is a prefix}] = \frac{1}{2}\left(\lambda[E \mid q0 \text{ is a prefix}] + \lambda[E \mid q1 \text{ is a prefix}]\right),$$

so that the conditional probabilities on the right cannot both equal $2^{-n+m(q)}$. This means that one of $q0$ and $q1$ is strange.

Case 2. The next request for a random bit comes from subcomputation i such that $i \leq k$ and $s'_i(q)$ is a proper initial segment of t_i. Then the next random bit read by M^∞ will go to subcomputation i, and so $q0$ (respectively, $q1$) will be pertinent if and only if the next bit in t_i after s'_i is 0 (respectively, 1). Suppose without loss of generality that this bit is 0, so that $q0$ is pertinent. We have $s'_i(q0) = s'_i(q)0$ and $s'_j(q0) = s'_j(q)$ for j in $\{1, \ldots, k\} - \{i\}$. We also have, because q is strange and $\lambda[E \mid q1 \text{ is a prefix}] = 0$ (since $q1$ is impertinent),

$$2^{-n+m(q)} \neq \lambda[E \mid q \text{ is a prefix}] = \frac{1}{2} \cdot \lambda[E \mid q0 \text{ is a prefix}],$$

so that

$$\lambda[E \mid q0 \text{ is a prefix}] \neq 2^{-n+m(q)+1} = 2^{-n+m(q0)}$$

and $q0$ is strange. □

We obtain a contradiction from the assumption that e is strange as follows. Repeatedly apply the lemma, starting with $q = e$, to obtain a sequence of strange strings, each a one-bit extension of the previous one. Thus we have an $r \in \{0, 1\}^\infty$ each of whose finite initial segments is strange. Fix such an r. By fairness each subcomputation j for $j = 1, \ldots, k$ either terminates or is infinite on r.

For $j = 1, \ldots, k$ each $s_j(r)$ is consistent with t_j because all finite initial segments of r, being strange, are pertinent. We saw earlier that $s_j(r)$ cannot be a proper initial segment of t_j, so that t_j must be an initial segment of $s_j(r)$ for each $j = 1, \ldots, k$. Thus for a sufficiently long finite initial segment q of r, each t_j is an initial segment of the corresponding $s_j(q)$, and therefore $s'_j(q) = t_j$ and $m(q) = n$. But then the event "q is a prefix" is a part of E, so that

$$\lambda[E \mid q \text{ is a prefix}] = 1 = 2^{-n+m(q)}.$$

This contradicts the fact that q, a finite initial segment of r, is strange. □

5. Randomizing many-one reductions. Let us clarify the notion of composition $M = M_2 \circ M_1$ of randomizing algorithms. Suppose that M_1 computes a random function f from a domain A to a domain B and that M_2 is a randomizing algorithm on B. Given an instance $x \in A$ and a random sequence $r \in \{0, 1\}^\infty$, M begins by simulating M_1 on x and r. If this computation terminates with output x' and if s is the finite initial segment of r read during this computation, then M continues by simulating M_2 on x' and the part of r after s. Thus

$$R_M(x) = \{st : s \in \mathrm{RF}_{M_1}(x) \text{ and } t \in R_{M_2}(f(x, s))\} \cup \mathrm{RI}_{M_1}(x).$$

LEMMA 5.1. *Suppose that* $\mathbf{P}_B(f(x, s)) > 0$ *whenever* $\mathbf{P}_A(x) > 0$, *and let* $T = T_{M_2}$ *be the restrained time function of* M_2. *Then both of the following hold*:

- *M is almost total if f and T are almost total.*
- *M is* AP *time if f is* AP *time and the composition $T \circ f$ is* AP.

Proof. The first claim is clear. To prove the second claim recall that, by definition, a randomizing algorithm is AP if and only if its restrained time function is so. Let Γ be a set of good inputs for $T \circ f$. It is easy to see that Γ is also a set of good inputs for f. Except for a small bookkeeping overhead, T_M is the sum of T_{M_1} and $T \circ f$ on Γ. Thus Γ is a set of good inputs for T_M, so that T_M is AP. □

In the rest of this section f is a random function from a domain A to a domain B and T ranges over random functions on B. If Γ is a dilation of A and Δ is a dilation of B, define $\Gamma * \Delta$ to be the dilation of A comprising pairs (x, st) such that $(x, s) \in \Gamma$ and $(f(x, s), t) \in \Delta$. The notion of domination is recalled in §2.

THEOREM 5.2. *Suppose that f is almost total. Then the following are equivalent*:

1. *For every almost-total,* AP *random T, the composition $S = T \circ f$ is almost total and* AP.
2. *The almost total function $(x, s) \mapsto |f(x, s)|_B$ is* AP, *and $D_f \leq_f B$.*

Proof. First, suppose part 1. Every deterministic function on B is an almost-total, random function on B with good inputs (y, e), where $\mathbf{P}_B(y) > 0$ and e is the empty string. By Theorem 1.3, $D_f \leq_f B$ and the restriction of the function $(x, s) \mapsto |f(x, s)|_B$ to D_f is AP. This implies part 2.

Second, suppose part 2, and let T be an almost-total, AP random function on B. The domain $\Gamma = D_f$ of the AP function $(x, s) \mapsto |f(x, s)|_B$ is an almost-total dilation of A. Similarly, the domain Δ of T is an almost-total dilation of B. These two facts and the domination condition imply that the dilation $D_S = \Gamma * \Delta$ is almost total. It remains to check that S, as a deterministic function $S(x, st) = T(f(x, s), t)$ on $\Gamma * \Delta$, is AP. This function is the composite of T with the function $j(x, st) = (f(x, s), t)$ from $\Gamma * \Delta$ to Δ. By Theorem 1.3 it suffices to prove that Δ dominates $\Gamma * \Delta$ with respect to j. We have

$$\begin{aligned}\mathbf{P}_{\Gamma*\Delta}[j^{-1}\{j(x, st)\}] &= \mathbf{P}_{\Gamma*\Delta}\{(x_0, s_0 t) : f(x_0, s_0) = f(x, s)\}\\ &= \mathbf{P}_\Gamma[f^{-1}\{f(x, s)\}] \cdot 2^{-|t|}\end{aligned}$$

and

$$\mathbf{P}_\Delta\{j(x, st)\} = \mathbf{P}_B\{f(x, s)\} \cdot 2^{-|t|},$$

so that the ratio in the definition of domination for j is $\mathbf{P}_\Gamma[f^{-1}\{f(x, s)\}]/\mathbf{P}_B\{f(x, s)\}$, which is AP on Γ because $\Gamma \leq_f B$. It follows immediately that this ratio is AP as a function on $\Gamma * \Delta$ (a function not depending on the t part of its argument). □

COROLLARY 5.3. *Let f be a function from a domain A to a domain B, computed by an almost-total,* AP*-time randomizing algorithm M_1. Suppose that $D_f \leq_f B$. Then for every almost-total,* AP*-time randomizing algorithm M_2 on B the composite algorithm $M = M_2 \circ M_1$ is almost total and* AP *time.*

Now we turn to the case when the composite is not required to be almost total.

THEOREM 5.4. *Suppose that Γ is a dilation of A. The following four statements are equivalent.*

(a) *For every* AP *random function T on B with Δ as a set of good inputs, $T \circ f$ is AP with $\Gamma * \Delta$ as a set of good inputs.*

(b) *For every almost-total,* AP *random function T on B with Δ as a set of good inputs, $T \circ f$ is* AP *with $\Gamma * \Delta$ as a set of good inputs.*

(c) *For every deterministic* AP *function T on B, $T \circ f$ is* AP *with Γ as a set of good inputs.*

(d) *The function $x \mapsto |f(x)|_B$ is* AP *with Γ as a set of good inputs and $\Gamma \leq_f B$.*

Proof. Clearly, (a) implies (b).

To derive (c) from (b) suppose (b) and let T be as in (c). Consider the dilation Δ of B comprising pairs (y, e), where y ranges over B and e is the empty string. Now apply (b).

To derive (d) from (c) use Lemma 3.12 to restate (c) as follows: If T is any AP (deterministic) function on B, then $T \circ f$ is an AP deterministic function on Γ. By Theorem 1.3 (and another application of Lemma 3.12), this implies (d).

It remains to derive (a) from (d). Assume (d), and let T and Δ be as in (a). Our first task is to show that the rarity function $U_{\Gamma*\Delta}$ for the composite dilation $\Gamma * \Delta$ of A is AP. It follows from (d) that U_Γ is AP. By the hypothesis of (a), U_Δ is AP. We compute

$$U_{\Gamma*\Delta}(x) = \left(\sum_{s\in\Gamma(x)} \sum_{t\in\Delta(f(x,s))} 2^{-|s|}2^{-|t|}\right)^{-1} = \left(\sum_{s\in\Gamma(x)} 2^{-|s|} \cdot \sum_{t\in\Delta(f(x,s))} 2^{-|t|}\right)^{-1}$$

$$= \left(\sum_{s \in \Gamma(x)} 2^{-|s|} \cdot \frac{1}{U_\Delta(f(x,s))} \right)^{-1},$$

and therefore for any δ with $0 < \delta < 1$

$$\sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) U^\delta_{\Gamma * \Delta}(x) = \sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) \left(\sum_{s \in \Gamma(x)} 2^{-|s|} \cdot \frac{1}{U_\Delta(f(x,s))} \right)^{-\delta}$$
$$= \sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) \left(\sum_{s \in \Gamma(x)} 2^{-|s|} \cdot U_\Gamma(x) \frac{1}{U_\Gamma(x) U_\Delta(f(x,s))} \right)^{-\delta}$$
$$= \sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) \left(\mathop{\mathbf{E}}_{s \in \Gamma(x)} \left[\frac{1}{U_\Gamma(x) U_\Delta(f(x,s))} \right] \right)^{-\delta},$$

where the expectation over $s \in \Gamma(x)$ is with respect to the probability distribution $\mathbf{P}(s) = 2^{-|s|} \cdot U_\Gamma(x)$.

Apply Jensen's inequality (in the form of Corollary 7.4) to the last part of the computation above:

$$\sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) U^\delta_{\Gamma * \Delta}(x) \le \sum_{x \in A} \frac{1}{|x|} \mathbf{P}_A(x) \mathop{\mathbf{E}}_{s \in \Gamma(x)} \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s)))^\delta \right]$$
$$= \sum_{x \in A} \sum_{s \in \Gamma(x)} \frac{1}{|x|} \mathbf{P}_A(x) \cdot 2^{-|s|} \cdot U_\Gamma(x) \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s)))^\delta \right]$$
$$= \sum_{(x,s) \in \Gamma} \frac{1}{|(x,s)|_\Gamma} \mathbf{P}_\Gamma(x,s) \left[(U_\Gamma(x) \cdot U_\Delta(f(x,s)))^\delta \right].$$

Thus $U_{\Gamma * \Delta}$ is AP if the product of $U_\Gamma(x)$ and $U_\Delta(f(x,s))$ is AP on Γ, i.e., if both factors are AP on Γ. Since $U_\Gamma(x)$ is AP on A, it is AP on Γ. Applying Theorem 1.3, we find that the composition $U_\Delta(f(x,s))$ is AP. Hence $U_{\Gamma * \Delta}$ is AP.

It remains to verify the convergence condition of the definition "$T \circ f$ is AP with $\Gamma * \Delta$ as a set of good points," which, by Lemma 3.12 is equivalent to the statement that $T \circ f$, as a deterministic function on $\Gamma * \Delta$, is AP. By the hypothesis of (a) plus Lemma 3.12, T, as a deterministic function on Δ, is AP. Since $T \circ f$ is obtained from T by composition with the function $g(x, st) = (f(x,s), t)$ from $\Gamma * \Delta$ to Δ and $|g|$ is AP on $\Gamma * \Delta$, the desired conclusion will follow by Theorem 1.3 if we show that Δ dominates $\Gamma * \Delta$ with respect to g.

For this purpose we use Lemma 2.6 and seek a measure ν on $\Gamma * \Delta$ that projects by g to $\mathbf{P}_\Delta$ and that dominates $\mathbf{P}_{\Gamma * \Delta}$. Assumption (d) that B dominates Γ with respect to f provides a measure μ on Γ that projects to $\mathbf{P}_B$ by f and dominates $\mathbf{P}_\Gamma$. We use μ to define ν by

$$\nu(x, st) = \mu(x,s) \cdot 2^{-|t|} \cdot U_\Delta(f(x,s)).$$

This projects to $\mathbf{P}_\Delta$ by g because for any $(x', t') \in \Delta$

$$\sum_{(x,st) \in g^{-1}(x',t')} \nu(x, st) = \sum_{(x,s) \in f^{-1}(x')} \nu(x, st')$$
$$= \sum_{(x,s) \in f^{-1}(x')} \mu(x,s) \cdot 2^{-|t'|} \cdot U_\Delta(f(x,s)) = 2^{-|t'|} U_\Delta(x') \sum_{(x,s) \in f^{-1}(x')} \mu(x,s)$$
$$= 2^{-|t'|} U_\Delta(x') \mathbf{P}_B(x') = \mathbf{P}_\Delta(x', t').$$

So it remains to prove that ν dominates $\mathbf{P}_{\Gamma*\Delta}$, that is, that the ratio $\mathbf{P}_{\Gamma*\Delta}(x, st)/\nu(x, st)$ is an AP function on $\Gamma * \Delta$. But this ratio is

$$\frac{\mathbf{P}_A(x) \cdot 2^{-|s|} \cdot 2^{-|t|} \cdot U_{\Gamma*\Delta}(x)}{\mu(x,s) \cdot 2^{-|t|} \cdot U_\Delta(f(x,s))} = \frac{\mathbf{P}_\Gamma(x,s) \cdot U_{\Gamma*\Delta}(x)}{U_\Gamma(x)\mu(x,s)U_\Delta(f(x,s))} \le \frac{\mathbf{P}_\Gamma(x,s)}{\mu(x,s)} \cdot U_{\Gamma*\Delta}(x).$$

To show that this is AP on $\Gamma * \Delta$ it suffices to show that both $\mathbf{P}_\Gamma(x,s)/\mu(x,s)$ and $U_{\Gamma*\Delta}(x)$, considered as functions of (x, st), are AP on $\Gamma * \Delta$. Of course, the former is AP as a function of (x, s) on Γ by our choice of μ, and the latter is AP as a function of x on A, as proved above, but neither of these is exactly what is needed. The desired information about $U_{\Gamma*\Delta}(x)$ follows easily because $\mathbf{P}_{\Gamma*\Delta}$ projects to $\mathbf{P}_A$ under the map $(x, st) \mapsto x$, which preserves sizes. Indeed, we have

$$\begin{aligned}
&\sum_{(x,st)\in\Gamma*\Delta} \frac{1}{|(x,st)|_{\Gamma*\Delta}} \cdot \mathbf{P}_{\Gamma*\Delta}(x,st) \cdot (U_{\Gamma*\Delta}(x))^\varepsilon \\
&= \sum_{x\in A} \left(\frac{1}{|x|} \cdot (U_{\Gamma*\Delta}(x))^\varepsilon \cdot \sum_{(x,st)\in\Gamma*\Delta} \mathbf{P}_{\Gamma*\Delta}(x,st) \right) \\
&= \sum_{x\in A} \frac{1}{|x|} \cdot (U_{\Gamma*\Delta}(x))^\varepsilon \mathbf{P}_A(x) < \infty
\end{aligned}$$

for sufficiently small $\varepsilon > 0$ since $U_{\Gamma*\Delta}$ is AP on A.

The same approach does not succeed with the function $\mathbf{P}_\Gamma/\mu$ on Γ because the projection $(x, st) \mapsto (x, s)$ from $\Gamma * \Delta$ to Γ need not send $\mathbf{P}_{\Gamma*\Delta}$ to $\mathbf{P}_\Gamma$; factors of $U_{\Gamma*\Delta}$ and U_Γ get in the way. Fortunately, Lemma 3.12 allows us to ignore those factors. Since $\mathbf{P}_\Gamma/\mu$ is AP on Γ, the lemma guarantees an $\varepsilon > 0$ such that

$$\sum_{(x,s)\in\Gamma} \frac{1}{|x|}\mathbf{P}_A(x) \cdot 2^{-|s|} \left(\frac{\mathbf{P}_\Gamma(x,s)}{\mu(x,s)}\right)^\varepsilon < \infty.$$

For any fixed $(x, s) \in \Gamma$ the numbers $2^{-|t|}$, where t ranges over $\Delta(f(x, s))$, add up to at most 1 since no such t is a proper initial segment of another. So

$$\sum_{(x,st)\in\Gamma*\Delta} \frac{1}{|x|}\mathbf{P}_A(x) \cdot 2^{-|s|}2^{-|t|} \left(\frac{\mathbf{P}_\Gamma(x,s)}{\mu(x,s)}\right)^\varepsilon < \infty.$$

By Lemma 3.12, $\mathbf{P}_\Gamma/\mu$ is AP on $\Gamma * \Delta$, and the proof of Theorem 5.4 is complete. $\square$

Question. Do parts (a), (b), and (c) of Theorem 5.4 remain equivalent if they are weakened to assert only that $T \circ f$ is AP, without specifying a particular set of good inputs? Is there an analog of (d) for this situation?

COROLLARY 5.5. *Let f be a random function from a domain A to a domain B computed by an* AP-*time randomizing algorithm M_1. If there exists a domain of good inputs for f dominated, with respect to f, by B, then for every* AP *randomizing algorithm M_2 on B the composite algorithm $M = M_2 \circ M_1$ is* AP *time.*

6. Impagliazzo and Levin's theorem. To formulate the theorem in question, we need a couple of definitions. We start with the definition of uniform domains. In the case of domains with finite many elements, it would be natural to call a domain uniform if all elements have the same probability. This definition makes no sense in the case of infinite domains, which is the only case of interest to us. Another natural way to define uniform domains requires a default probability distribution on positive integers; it is customary to assign the probability $1/n(n + 1)$ to a positive integer n.

DEFINITION 6.1. A domain is *uniform* if it has a finite number of elements of any given size, all elements of a given size have the same probability, and
$\mathbf{P}\{x : |x| = n\} = 1/n(n+1)$.

DEFINITION 6.2 [1]. A domain A is *samplable* if there exists a randomized algorithm S such that all of the following conditions hold:

- S takes no input (but tosses coins) and outputs an element of A if it converges.
- $\mathbf{P}_A(x)$ is proportional to the probability that S outputs x.
- The computation time of S is bounded by a polynomial of the size of the output.

The restriction on the computation time of S can be relaxed [4].

Impagliazzo and Levin [8] deal with domains (though they do not use the term) for which the size $|x|$ of an element x is its length (recall that domain elements are strings); this restriction is not necessary [4], but it simplifies the exposition and we stick to it in this section. Call a sampling algorithm S *length preserving* if it uses exactly n coin tosses to produce a string of length n. The following fact is well known, and we omit the proof.

LEMMA 6.3 [1], [4], [11]. *Every search problem on a samplable domain reduces to a search problem on a domain sampled by a length-preserving algorithm.*

Impagliazzo and Levin prove that every NP search problem on a samplable domain reduces to an NP search problem on a uniform domain. Recall that a search problem SP(A, W) is NP if the length of a witness is bounded by a polynomial of the size of the instance and the witness relation $W(x, w)$ is PTime computable relative to $|x|$. In fact, the argument of Impagliazzo and Levin does not require the restriction to NP problems.

THEOREM 6.4 [8]. *Every search problem* $\Pi_1 = \mathrm{SP}(A_1, W_1)$ *on a samplable domain* A_1 *reduces to a search problem on the uniform domain* BS *of binary strings. If the source problem is* NP, *then the target problem may be chosen to be* NP *as well.*

Proof. First, we indicate the motivation behind the proof. There are two simple but unsuccessful methods for attempting to reduce SP(A_1, W_1) to a search problem on a uniform domain. One is simply to redefine the probability distribution on A_1 to be uniform. This fails because the obvious instance transformer, the identity map, violates the domination condition. It is entirely possible for some instances x to have vastly larger probability in the given samplable A_1 than in the uniform domain; this occurs when $S^{-1}(x)$ is large (since S preserves length). A second method is to use the search problem on the domain BS defined by $W'(u) = W(S(u))$. The trouble with this is that an instance transformer should produce, for any x, some $u \in S^{-1}(x)$, and this may be difficult even for a randomizing algorithm. Indeed, one could imagine that each such u encodes a witness w for x, so that the new search problem is trivial. Notice, however, that this second method fails only when $S^{-1}(x)$ is rather small; when it is large, an element of it can be found by guessing random elements and checking them (as S is quickly computable).

The strategy of the proof is to interpolate between these two methods, leaning toward the first (respectively, second) when $S^{-1}(x)$ is small (respectively, large). More precisely, an instance of the new problem should be (approximately) $n = |x|$ bits long, consisting of $l = \lceil \log(||S^{-1}(x)||) \rceil$ bits of information about some $u \in S^{-1}(x)$ and $n - l$ bits about x. (Here $||S^{-1}(x)||$ is the cardinality of the set $S^{-1}(x)$.)

There are some obvious difficulties with this. For one thing, we cannot efficiently compute l from x. But we can guess it; it lies between 0 and n, so that the probability of guessing correctly is the reciprocal of a polynomial, which is good enough to give a nonrare dilation. Another issue is how to select the right bits of information about u and x. This, too, is solved by randomization. We randomly choose two matrices L and M (with entries in the two-element field $\{0, 1\}$) of the appropriate size to hash u and x to vectors Lu and Mx of lengths l and

$n - l$, respectively. A key point in the proof is that randomly chosen hash matrices have a reasonable probability of working the way we want.

Thus, finally, an instance of the new problem will consist of the two hash matrices L and M and the results Lu and Mx of hashing u and v. A witness for such an instance will consist of a u and a w such that Lu is as specified in the instance, $S(u)$ is an x whose hashing Mx is as specified in the instance, and $W(x, w)$ holds. (Actually, there are a couple of minor technical modifications in the actual proof, but this is the essential idea.)

Now we give the actual proof. We construct search problems $\Pi_i = \mathrm{SP}(A_i, W_i)$ for $i = 2, 3, 4, 5$ with $A_5 = \mathrm{BS}$ and reduce each Π_i with $i < 5$ to Π_{i+1}. Accordingly, we have four reduction lemmas. In each lemma the desired reduction is called (Γ, f, g). The probability distribution of A_i is denoted $\mathbf{P}_i$.

By Lemma 6.3 we may suppose that S is size preserving.

Notation. In the rest of this section x is an instance of Π_1 with $W_1(x) \neq \emptyset$, $n = |x|$, m is the cardinality of the set $S^{-1}(x)$, and $l = \lceil \log m \rceil$. If j is a natural number, then $b(j)$ is the (shortest) binary notation for j.

We define Π_2. A_2 comprises pairs (x, l), where x is an element of A_1 and l is as above. The size and probability of (x, l) are the size and probability of x in A_1. Further, $W_2(x, l) = W_1(x)$.

LEMMA 6.5. *Π_1 reduces to Π_2.*

Proof. Define $\Gamma(x)$ to contain one string, namely, $b(l)$. We have $\mathrm{Rarity}_\Gamma(x) = 2^{|b(l)|} \leq 2l \leq 2n$, so that Γ is not rare.

Define $f(x, b(l)) = (x, l)$. To check the domination property, take into account that f is injective:

$$\frac{\mathbf{P}_\Gamma(x, b(l))}{\mathbf{P}_2(x, l)} = \frac{\mathbf{P}_1(x) \cdot 2^{-|b(l)|}}{\mathbf{P}_1(x)} \leq 1.$$

Finally, define $g((x, b(l)), w) = w$. □

Notation. u ranges over binary strings of length n, u' ranges over binary strings of length l, and L ranges over $l \times n$ matrices over the field of two elements. If I is a matrix over the field of two elements, then $b(I)$ is the binary string obtained from I by writing down the first row of I, then the second row of I, and so on. If s and s' are strings, then $s * s'$ denotes the concatenation of s and s'.

We define Π_3. A_3 comprises triples (x, L, u') such that L has full rank (i.e., rank l) and there exists $u \in S^{-1}(x)$ with $Lu = u'$. A_3 is isomorphic to a subdomain of $A_2 \times \mathrm{BS}$; the isomorphism is $\iota_3(x, L, u') = ((x, l), b(L) * u')$. Further,

$$W_3(x, L, u') = \{(u, w) : S(u) = x, Lu = u', \text{ and } w \in W_1(x)\}.$$

LEMMA 6.6. *Π_2 reduces to Π_3.*

Proof. Define Γ to be the ι_3-image of A_3. To prove that Γ is not rare fix an instance (x, l) of Π_2.

CLAIM 6.7. *In the uniform probability space of $l \times n$ matrices, the probability that a matrix has full rank exceeds a positive constant independent of l and n, e.g., $1/4$.*

Proof. Only one row (namely, the row of zeros) cannot serve as the first row of a full-rank matrix. Given the first row u_1, only two rows (namely, u_1 and the zero row) cannot serve as the second row of a full-rank matrix. Given the first two rows u_1 and u_2, only four rows (namely, the four linear combinations of u_1 and u_2) cannot serve as the third row of a full-rank matrix, and so on. Thus the number of full-rank matrices is

$$(2^n - 1)(2^n - 2)(2^n - 4) \cdots (2^n - 2^{l-1}) = \prod_{i=0}^{l-1} (2^n - 2^i).$$

The total number of $l \times n$ matrices is 2^{ln}. Hence the probability of full rank is

$$\begin{aligned}\frac{\prod_{i=0}^{l-1}(2^n - 2^i)}{2^{ln}} &= \prod_{i=0}^{l-1} \frac{2^n - 2^i}{2^n} \\ &= \prod_{i=0}^{l-1}\left(1 - \frac{1}{2^{n-i}}\right) > \prod_{i=0}^{\infty}\left(1 - \frac{1}{2^i}\right) = \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{4}\right)\left(1 - \frac{1}{8}\right)\cdots.\end{aligned}$$

To estimate this product consider the following probabilistic experiment. For each i let U_i be an urn with 2^i balls such that exactly one of the balls is red and the others are green. Let A_i be the event of selecting a green ball from U_i, and let B_i be the event of selecting the red ball, so that $\mathbf{P}(A_i) = 1 - \mathbf{P}(B_i) = 1 - 1/2^i$. We have

$$\begin{aligned}\prod_{i=2}^{\infty}(1 - \frac{1}{2^i}) &= \mathbf{P}\left[\bigcap_{i=2}^{\infty} A_i\right] \\ &= 1 - \mathbf{P}\left[\bigcup_{i=2}^{\infty} B_i\right] > 1 - \sum_{i=2}^{\infty}\mathbf{P}(B_i) \\ &= 1 - \left(\frac{1}{4} + \frac{1}{8} + \cdots\right) \\ &= 1 - \frac{1}{2} = \frac{1}{2}.\end{aligned}$$

Hence the probability of full rank exceeds $(1 - 1/2)1/2 = 1/4$. □

CLAIM 6.8. *Consider the probability space of pairs (L, u'), where L is of full rank. For each x the probability of the event*

$$\{(L, u') : (\exists u \in S^{-1}(x))(Lu = u')\}$$

is at least $3/8$.

Proof. Let u, v range over $S^{-1}(x)$. For each u, let $E(u) = \{(L, u') : Lu = u'\}$. Then the cardinality $||E(u)||$ of $E(u)$ is the number F of full-rank matrices L. The event in question is $\bigcup E(u)$, and the probability in question is $||\bigcup E(u)||/(F2^l)$.

We check whether, if $u \neq v$, the set $E(u, v) = E(u) \cap E(v)$ contains at most $F/2^l$ elements. Notice that

$$[(L, u') \in E(u, v)] \Longleftrightarrow [Lu = u' = Lv] \Longrightarrow [L(u - v) = 0]$$

and that for each L that annihilates $u - v$ there exists a unique u' such that $(L, u') \in E(u, v)$. Thus $||E(u, v)||$ equals the number of full-rank matrices L that annihilate $u - v$. Among all $l \times n$ matrices, the probability of annihilating $u - v$ is exactly $(1/2)^l$; among full-rank matrices, the probability is $\leq (1/2)^l$. Let us make this precise.

If $l = n$, then the probability that a full-rank matrix annihilates $u - v$ equals 0; so assume that $l < n$. The probability p that a random full-rank matrix L annihilates a specified nonzero vector $z \in \{0, 1\}^n$ is independent of z. For any z, z' there is a nonsingular $n \times n$ matrix A such that $Az = z'$. Then L annihilates z' if and only if LA annihilates z, and LA has full rank if and only if L does.

So without loss of generality $z = (0, 0, \ldots, 0, 1)$. It is clear now that p equals the number of full-rank $l \times n$ matrices with the last column of zeros divided by the total number of full-rank $l \times n$ matrices. In other words, p equals the number of full-rank $l \times (n - 1)$ matrices divided by the number of full-rank $l \times n$ matrices. Recall our counting of full-rank matrices above. We have

$$p = \frac{\prod_{i=0}^{l-1}(2^{n-1} - 2^i)}{\prod_{i=0}^{l-1}(2^n - 2^i)} \le \frac{\prod_{i=0}^{l-1}(2^{n-1} - 2^{i-1})}{\prod_{i=0}^{l-1}(2^n - 2^i)} = \prod_{i=0}^{l-1} \frac{1}{2} = \left(\frac{1}{2}\right)^l.$$

By the inclusion–exclusion principle [10]

$$\frac{1}{F2^l} \| \bigcup_u E(u) \| \ge \frac{1}{F2^l} \left(\sum_u \|E(u)\| - \sum_{u \ne v} \|E(u) \cap E(v)\| \right).$$

Since the number of u's is m,

$$\frac{1}{F2^l} \sum_u \|E(u)\| = \frac{1}{F2^l} mF = \frac{m}{2^l}.$$

Further,

$$\frac{1}{F2^l} \sum_{u \ne v} \|E(u) \cap E(v)\| \le \frac{1}{F2^l} \frac{m(m-1)}{2} \frac{F^l}{2} = \frac{m(m-1)}{2^{2l+1}}.$$

Thus the probability we want is bounded below by $m/2^l - m(m-1)/2^{2l+1} > t - \frac{1}{2}t^2$, where $t = m/2^l$ is between $\frac{1}{2}$ and 1 (because of the definition of l). But the minimum of the quadratic function $t - \frac{1}{2}t^2$ on the interval $[\frac{1}{2}, 1]$ is $\frac{3}{8}$. So the probability we want is greater than $\frac{3}{8}$. □

The two claims imply that Γ is nonrare. Define $f = \iota_3^{-1}$ and $g(((x, l), b(L)*u'), (u, w)) = w$. It is easy to see that (Γ, f, g) is indeed the desired reduction. □

Notation. $k = n + 1 - l$ and M ranges over $k \times n$ matrices over the two-element field.

We define Π_4. A_4 comprises 5-tuples (x, L, u', M) such that $(x, L, u') \in A_3$, M is as above (a $k \times n$ matrix), and there is no $u \in \{0, 1\}^n - S^{-1}(x)$ such that $Lu = u'$ and $M(S(u) - x) = 0$. It is isomorphic to a subdomain of $A_3 \times$ BS by the isomorphism $\iota_4(x, L, u', M) = ((x, L, u'), b(M))$. Further, $W_4(x, L, u', M) = W_3(x, L, u')$.

LEMMA 6.9. *Π_3 reduces to Π_4.*

Proof. Define Γ to be the ι_4-image of A_4. To prove that Γ is not rare fix an element (x, L, u') of A_3 and call a matrix M *bad* if there exists $u \in \{0, 1\}^n - S^{-1}(x)$ such that $Lu = u'$ and $M(S(u) - x) = 0$.

CLAIM 6.10. *The fraction of bad matrices M is at most* $1/2$.

Proof. The number of strings u satisfying the equation $Lu = u'$ is $2^{n-l} = 2^{k-1}$. For each $u \in \{0, 1\}^n - S^{-1}(x)$ there are exactly 2^{kn-k} matrices M satisfying the equation $M(S(u) - x) = 0$. Thus the number of bad matrices M is at most $2^{k-1}2^{kn-k} = 2^{kn-1}$, which is exactly half of the total number of matrices M. □

The claim implies that Γ is not rare. Define $f = \iota_4^{-1}$ and $g(((x, L, u'), b(M))(u, w)) = (u, w)$. It is easy to see that Γ, f, g is indeed a reduction. □

The domain A_5 of our final problem Π_5 is BS. We do some work before defining W_5. Consider the function

$$f(x, L, u', M) = b(L) * u' * b(M) * b(Mx) * b(l)$$

from A_4 to A_5. (It will serve eventually as the instance transformer of the desired reduction of Π_4 to Π_5.)

CLAIM 6.11. *f is injective, and the components L, u', and M as well as the numbers n, l, and k are* PTime *computable from $f(x, L, u', M)$.*

Proof. Because $|f(x, L, u', M)| = ln + l + kn + k + l = (n + 1)^2 + l$ and $l \leq n$, the numbers n, l, and k are easily computable from $|f(x, L, u', M)|$. It follows that L, u', and M are easily computable from $f(x, L, u', M)$. To prove that f is injective, suppose that $f(x_2, L, u', M) = f(x, L, u', M)$. Then $M(x_2 - x) = 0$, and there exists $u \in S^{-1}(x_2)$ such that $Lu = u'$ and $M(S(u) - x) = 0$. Since M is not bad for (x, L, u') (in the sense of the proof of Lemma 6.9), $x_2 = S(u) = x$. □

CLAIM 6.12. *f deterministically reduces A_4 to A_5.*

Proof. Clearly, f is PTime computable. We need only to check that f satisfies the domination condition. Since f is injective,

$$\frac{\mathbf{P}_4(f^{-1}(f(x, L, u', M))}{\mathbf{P}_5(f(x, L, u', M))} = \frac{\mathbf{P}_4(x, L, u', M)}{\mathbf{P}_5(f(x, L, u', M))} = \frac{\mathbf{P}_1(x)\mathbf{P}_{\mathrm{BS}}(b(L) * u' * b(M))}{\mathbf{P}_{\mathrm{BS}}(b(L) * u' * b(M) * b(Mx) * b(l))}.$$

We may ignore factors polynomial in $|(x, L, u')|$, i.e., polynomial in n. In that sense,

$$\frac{\mathbf{P}_1(x)\mathbf{P}_{\mathrm{BS}}(b(L) * u' * b(M))}{\mathbf{P}_{\mathrm{BS}}(b(L) * u' * b(M) * b(Mx) * b(l))} \approx \frac{2^{-(n-l)}2^{-(ln+l+kn)}}{2^{-(ln+l+kn+k+\log l)}} \approx 1. \quad \Box$$

Now we are ready to define W_5.

$$W_5(s) = \begin{cases} W_4(x, L, u', M) & \text{if } s = f(x, L, u', M) \\ \emptyset & \text{otherwise.} \end{cases}$$

Since S is PTime computable and Π_1 is an NP search problem, the search problem Π_5 is NP as well.

LEMMA 6.13. *Π_4 deterministically reduces to Π_5.*

Proof. The instance transformer f of the desired reduction is already defined. The witness transformer is $g((x, L, u', M), (u, w)) = (u, w)$. It is easy to see that (f, g) is indeed a reduction. □

Theorem 6.4 is proved. □

7. Appendix (Jensen's inequality). For the reader's convenience we prove here two forms of Jensen's inequality that we need. The proof of Jensen's inequality from [12] is used.

THEOREM 7.1. *Consider an increasing concave function f on an interval (a, ∞) of the real line, and set $f(\infty) = \lim_{x\uparrow\infty} f(x)$. For every random variable X with values in the interval $(a, \infty]$ of the real line extended with ∞,*

$$\mathrm{E}(f(X)) \leq f(\mathrm{E}(X)).$$

Proof. If $\mathrm{E}(X) = \infty$, then $f(\mathrm{E}(X)) = \lim_{x\uparrow\infty} f(x) \geq \mathrm{E}(f(X))$. Suppose that $\mathrm{E}(X) < \infty$. The fact that f is concave means that for $a < u < v < w$

$$\frac{f(v) - f(u)}{v - u} \geq \frac{f(w) - f(v)}{w - v}.$$

It is clear that the monotone limits

$$A(v) =\downarrow \lim_{u\uparrow v} \frac{f(v) - f(u)}{v - u}, \; B(v) =\uparrow \lim_{w\downarrow v} \frac{f(w) - f(v)}{w - v}$$

exist and that $A(v) \geq B(v)$. For all positive real v, x and every c in $[B(v), A(v)]$ we have $f(x) \leq c(x - v) + f(v)$. In particular, there is c such that $f(X) \leq c(X - \mathrm{E}(X)) + f(\mathrm{E}(x))$ and the desired inequality follows after expectations are taken. □

COROLLARY 7.2. *Suppose* $0 < \delta < 1$, *and let* X *be any random variable with values in* $(0, \infty]$. *Then*

$$\mathrm{E}(X^{\delta}) \leq (\mathrm{E}(X))^{\delta}.$$

THEOREM 7.3. *Consider a decreasing convex function* g *on an interval* $(0, b)$ *of the real line and set* $g(0) = \lim_{x \downarrow 0} g(x)$. *For every random variable* X *with values in the interval* $[0, b)$ *of the real line,*

$$\mathrm{E}(g(X)) \geq g\left(\mathrm{E}(X)\right).$$

Proof. First, suppose $\mathrm{E}(X) = 0$. Since all values of X are ≥ 0, we must have with probability 1 that $X = 0$ and therefore $g(X) = g(0)$. Then $E(g(X)) = g(0) = g(E(X))$.

In the case $\mathrm{E}(X) > 0$ the proof is similar to the part of the proof of Theorem 7.1 for the case $\mathrm{E}(X) < \infty$; just reverse some arrows and inequalities. □

COROLLARY 7.4. *Suppose* $0 < \delta < 1$, *and let* X *be any random variable with values in* $[0, b)$. *Then*

$$\mathrm{E}(X^{-\delta}) \geq (\mathrm{E}(X))^{-\delta}.$$

REFERENCES

[1] S. BEN-DAVID, B. CHOR, O. GOLDREICH AND M. LUBY, *On the theory of average case complexity*, J. Comput. System Sci., 44 (1992), pp. 193–219.

[2] A. BLASS AND Y. GUREVICH, *On the reduction theory for average-case complexity*, in Proc. CSL '90, 4th Workshop on Computer Science Logic, E. Börger, H. K. Büning and M. Richter, eds.,, Springer Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1991.

[3] ———, *Randomizing reductions of search problems*, in Proc. 11th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer Lecture Notes in Computer Science 560, Springer-Verlag, Berlin, 1991, pp. 10–24.

[4] ———, *Randomizing reductions of decision problems* (tentative title), in preparation.

[5] Y. GUREVICH, *Average case complexity*, J. Comput. System Sci., 42 (1991), pp. 346–398.

[6] ———, *Average case complexity*, in Proc. International Colloquium on Automata, Languages and Programming, Springer Lecture Notes in Computer Science 510, Springer-Verlag, Berlin, 1991, pp. 615–628.

[7] Y. GUREVICH AND S. SHELAH, *Expected computation time for Hamiltonian path problem*, SIAM J. Comput., 16 (1987), pp. 486–502.

[8] R. IMPAGLIAZZO AND L. A. LEVIN, *No better ways to generate hard* NP *instances than picking uniformly at random*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Washington, D.C., 1990, pp. 812–821.

[9] L. A. LEVIN, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.

[10] R. P. STANLEY, *Enumerative Combinatorics* I, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1986.

[11] R. VENKATESAN AND L. LEVIN, *Random instances of a graph coloring problem are hard*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1988, pp. 217–222.

[12] D. WILLIAMS, *Probability with Martingales*, Cambridge University Press, London, 1991.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 976–993, October 1993

AN OPTIMAL ALGORITHM FOR THE MAXIMUM THREE-CHAIN PROBLEM*

RUEY-DER LOU† AND MAJID SARRAFZADEH†

Abstract. Given a two-dimensional point set ρ, a chain C is a subset of ρ in which for every two points one is dominated by the other. A k-chain is a subset of ρ that can be partitioned into k-chains. The size of a k-chain is the total number of its points. A k-chain with maximum size, among all possible k-chains, is called a *maximum* k-chain. First geometric properties of k-chains are studied for an arbitrary k. Then a $\Theta(n \log n)$-time algorithm is presented for finding a maximum three-chain in a point set ρ, where $n = |\rho|$.

Key words. point dominance, chains, computational geometry, lower bound, optimal algorithm

AMS subject classifications. 52B55, 68Q20, 68Q25

1. Introduction. Consider a point set $\rho = \{P_1, P_2, \dots, P_n\}$ in the xy-plane, where $P_i = (x_i, y_i)$ with $x_i > 0$ and $y_i > 0$; by convention, $x_i > x_j$ for $i > j$ and all the y's are distinct. Following [6], we say P_i *dominates* P_j if $x_i > x_j$ and $y_i > y_j$. Points P_i and P_j are *crossing* if $x_i > x_j$ and $y_j > y_i$ or vice versa; otherwise, they are *noncrossing*. A subset of ρ is called a *chain* if its points are pairwise noncrossing. A subset is called a *k-chain* if it can be partitioned into k chains. The *size* of a k-chain is the number of its points. A *maximum* chain is a chain with maximum size among all possible chains. A *maximum k-chain* is a k-chain with maximum size among all possible k-chains. The problem of finding a maximum k-chain is equivalent to the following problems:

Graph Theory: In a given representation of a permutation graph as a family of chords, find a maximum k-partite subgraph. (The more general problem of finding a maximum k-partite subgraph in an overlap graph, i.e., a circle graph, is NP-hard [8].)

Sequence Manipulation: Given a sequence of integers, find a maximum k-increasing subsequence. (A k-increasing subsequence is a subset that can be partitioned into k increasing subsequences.)

VLSI Layout: Solve the maximum k-layer subset problem and the k-layer topological by minimization problem in a channel. Normally, k is a small constant (2, 3, or 4) [3], [7]–[10].

The problem of the unweighted maximum k-partite subgraph in transitive graphs (which includes the class of permutation graphs) was solved in $O(kn^2)$ time, where n is the number of vertices [4]. This result was extended to the weighted case in [3], [10] (both algorithms run in $O(kn^2)$ time). By using the geometric nature of permutation graphs (i.e., solving the equivalent maximum k-chain problem) the following results have been obtained: A $\Theta(n \log n)$-time algorithm for finding a maximum one-chain has been proposed [2]. This result was extended to find a maximum two-chain in $\Theta(n \log n)$ time [5]. In this paper we study geometric properties of k-chains. Then we propose a $\Theta(n \log n)$-time algorithm for finding a maximum three-chain.

This paper is organized as follows. In §2, preliminary results are presented. The main idea of the algorithm is presented in §3. In §4, details of the implementation and the time complexity of the algorithm are discussed.

2. Geometric properties of chains. In this section we first present an overview of the approach. Next we provide some preliminary results that are essential for establishing the main result.

*Received by the editors June 24, 1991; accepted for publication (in revised form) June 9, 1992. This work was supported in part by the National Science Foundation under grant MIP-8921540 and MIP 9207267.

†Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208.

2.1. Overview. The problem of finding a maximum k-chain and the more general problem of finding a maximum k-covering in a transitive graph G are well understood. The idea is to repeatedly (i.e., k times) find a minimum-cost flow in G. At the end of each step, the graph G needs to be *modified*. The currently known implementation of this idea requires $O(kn^2)$ time, where n is the number of points (e.g., see [4]).

In the xy-plane a minimum-cost flow corresponds to finding a maximum chain. This task can be efficiently accomplished by using the notion of point dominance [2]. To repeat this task k times—as is done in the graph version of the problem—we need to *modify* the xy-plane. Explicitly this is not possible. We will show that modifying the plane is equivalent to redefining the cost of the chains. This idea is implicitly used in [5]. Here we use this idea explicitly. We also explain why the same idea cannot be efficiently used for a k-chain, $k > 3$.

First we show various geometric relations between a maximum h-chain and a maximum $(h+1)$-chain. Then we show that a maximum h-chain can be effectively combined with a collection of chains and antichains (which we call a z-chain) to obtain a maximum $(h+1)$-chain. We also show, on the basis of the notion of dominance, that a z-chain can be efficiently obtained.

2.2. Preliminary results. Consider a chain $C = \{P_{\pi_1}, P_{\pi_2}, \ldots, P_{\pi_m}\}$ in a point set ρ, where $x_{\pi_i} < x_{\pi_{i+1}}$ for $1 \leq i < n$. In the xy-plane an *extended path* $\mathcal{P}(C)$ of C is a path vertically from $(x, y) = (x_{\pi_1}, -\infty)$ to P_{π_1}, then to (x_{π_2}, y_{π_1}), then to $P_{\pi_2}, \ldots, P_{\pi_m}$, and finally from P_{π_m} to (x_{π_m}, ∞). Extended path $\mathcal{P}(C)$ partitions the plane into three connected regions: points on $\mathcal{P}(C)$, points to its left (i.e., points containing $x = -\infty$), and points to its right (i.e., points containing $x = +\infty$). The points to the left of $\mathcal{P}(C)$ are called the left-hand side thereof and the points to the right of $\mathcal{P}(C)$ are called the right-hand side thereof. The points in ρ are also partitioned into three sets by the extended path $\mathcal{P}(C)$:

(a) the subset of points on the left-hand side of $\mathcal{P}(C)$;

(b) chain C;

(c) the subset of points on the right-hand side of $\mathcal{P}(C)$.

We partition an h-chain H into h chains such that the ith chain (from left to right) is on the left-hand side of the extended path of the $(i+1)$st chain. We call such partition a *normal partition* of H, and we call each chain a *component chain* of H. (Clearly, a normal partition of every h-chain can be found in linear time.) $H(i)$ represents the ith, from left to right, component chain of H. In the following discussion we express each h-chain H by $H = H(1) \cup H(2) \cup \cdots \cup H(h)$.

Let M_h represent a maximum h-chain and $M_h(i)$ represent the ith component chain of M_h. The extended paths of the component chains of M_h, (i.e., $\mathcal{P}(M_h(1)), \mathcal{P}(M_h(2)), \ldots$, and $\mathcal{P}(M_h(h))$, partition the point set ρ into $2h+1$ subsets:

(a) h component chains of M_h;

(b) $R_h(1)$, the subset of the points on the left-hand side of $\mathcal{P}(M_h(1))$;

(c) $R_h(i)$, the subset of the points on the right-hand side of $\mathcal{P}(M_h(i-1))$ and on the left-hand side of $\mathcal{P}(M_h(i))$ for $2 \leq i \leq h$;

(d) $R_h(h+1)$, the subset of the points on the right-hand side of $\mathcal{P}(M_h(h))$.

As described in §2.1, our goal is to obtain a maximum $(h+1)$-chain from a maximum h-chain. To accomplish this task efficiently we need to establish some properties of a maximum $(h+1)$-chain. In particular, we will show that each component of a maximum $(h+1)$-chain is bounded by two components of a maximum h-chain. Such properties will be referred to as the *partition properties* (to be proved in Lemmas 2.1 and 2.2) and properties P1–P3 (to be proved in Lemma 2.3).

Consider a maximum h-chain M_h and an $(h+1)$-chain $F = F(1) \cup F(2) \cup \cdots \cup F(h+1)$ in point set ρ. We say F satisfies the partition property of M_h if

(a) $F(1)$ is a subset of $R_h(1) \cup M_h(1)$;

(b) $F(i)$ is a subset of $M_h(i-1) \cup R_h(i) \cup M_h(i)$ for $2 \le i \le h$;

(c) $F(h+1)$ is a subset of $M_h(h) \cup R_h(h+1)$.

Given a maximum h-chain M_h and a chain C, we can readily construct an $(h+1)$-chain $F' = M_h \cup C$ such that F' satisfies the partition property of M_h. An example is shown in Fig. 1.

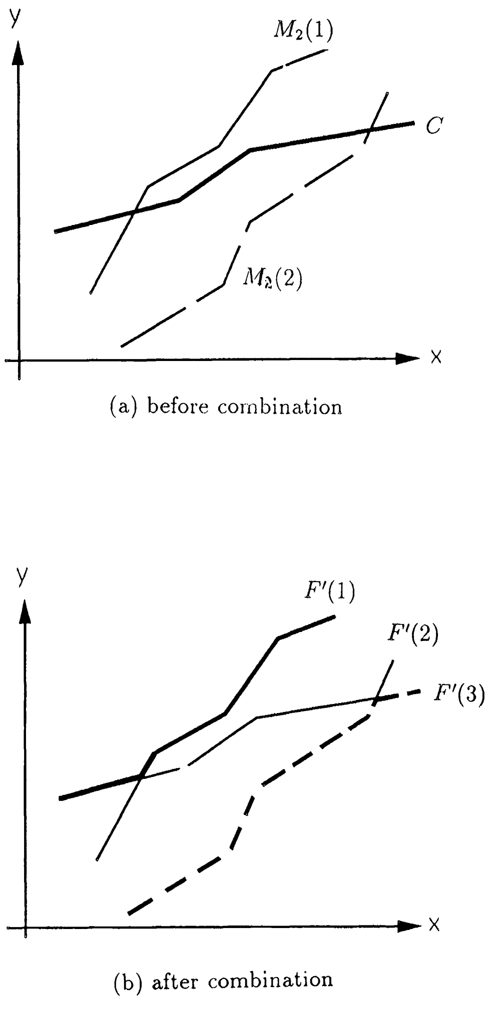


FIG. 1. *The combination of* M_2 *and* C: (a) *before combination and* (b) *after combination.*

LEMMA 2.1. *Given a maximum h-chain* M_h *and an* $(h+1)$*-chain* F *in point set* p*, there exists an* $(h+1)$*-chain* F' *such that* F' *satisfies the partition property of* M_h *and* $|F'| \ge |F|$.

Proof. Let $F(j)$ be a component chain of F. $M_h \cap F(j)$ may not be empty. The total number of points in the other h component chains of F is less than or equal to that of M_h. Clearly, we can construct an $(h+1)$-chain $F' = M_h \cup F(j)$ such that F' satisfy the partition property of M_h and $|F'| \ge |F|$. $\square$

The following lemma is a corollary of Lemma 2.1.

LEMMA 2.2. *Given a maximum h-chain M_h in point set ρ, there exists at least one maximum $(h+1)$-chain M_{h+1} such that M_{h+1} satisfies the partition property of M_h.*

Consider a chain C in the point set ρ. The highest point of C (i.e., the point with maximum y coordinate) is called the *top point* of C. Similarly, the lowest point of C is called the *bottom point* of C. Assume $C = \{P_{\pi_1}, P_{\pi_2}, \ldots, P_{\pi_m}\}$. Let $C' = \{P_{\pi_i}, P_{\pi_{i+1}}, \ldots, P_{\pi_{i+j}}\}$ be a subset of C such that there is no point in $C - C'$ that dominates some point in C' and is also dominated by some point in C'. Then we say C' is a *continuous portion* of C. Let C^1, C^2, and C^3 be three chains in ρ. Assume C^1 and C^2 have some common points. A point set D is called an *overlapped portion* of C^1 with C^2 if D has the following properties:

(O1) D is a continuous portion of C^1 and C^2 (hence D is also a subset of common points of C^1 and C^2).

(O2) No other point set with property O1 contains D as a subset (i.e., D is maximal).

Similarly, we define overlapped portions of C^2 with C^3. Notice that an overlapped portion of C^1 with C^2 may overlap with an overlapped portion of C^2 with C^3. With this definition there exists an $(h+1)$-chain M_{h+1} that satisfies the partition property of M_h. Furthermore, M_{h+1} has the following properties (to be proved below):

(P1) All bottom points of M_h are bottom points of M_{h+1}.

(P2) All top points of M_h are top points of M_{h+1}.

(P3) On each component chain $M_h(j)$ its portions overlapped with $M_{h+1}(j)$ and with $M_{h+1}(j+1)$ appear alternately from bottom to top.

LEMMA 2.3. *There exists an $(h+1)$-chain M_{h+1} that satisfies the partition property of M_h and attains properties* P1, P2, *and* P3.

Proof. Consider an $(h+1)$-chain M_{h+1} that satisfies the partition property of M_h. According to Lemma 2.1, such a chain exists. Now we have to show that M_{h+1} attains additional properties.

(1) *Properties* P1 *and* P2. Assume one bottom point P_{b_j} of component chains $M_h(j)$ is not a bottom point of any component chain of M_{h+1} (see Fig. 2). Let the lowest point of $M_h(j) \cap M_{h+1}$ be P_a. Without loss of generality, assume P_a is on $M_{h+1}(j+1)$. Let the points in $M_{h+1}(j+1)$ lower than P_a and not in M_h be $P_{a_1}, P_{a_2}, \ldots, P_{a_m}$, where $x_{a_1} < x_{a_2} < \cdots < x_{a_m}$. Let m' denote the number of the points in $M_h(j)$ below P_a. Because M_{h+1} is a maximum $(h+1)$-chain, $m \geq m'$. Because M_h is a maximum h-chain, $m \leq m'$. Thus $m = m'$. Hence we can delete points $P_{a_1}, P_{a_2}, \ldots, P_{a_m}$ from M_{h+1} and add the points that are on $M_h(j)$ and below P_a to M_{h+1}. In the new M_{h+1}, P_{b_j} is the bottom point of $M_{h+1}(j)$. Repeating this procedure, we can make all bottom points of M_h bottom points of M_{h+1}. Similarly, we can make all top points of M_h top points of M_{h+1}.

(2) *Property* P3. Assume property P3 is not satisfied for a maximum $(h+1)$-chain M_{h+1}. Let $M_h(j)$ be the leftmost component chain of M_h, such that property P3 is not satisfied (i.e., the portions of $M_h(j)$ overlapped with $M_{h+1}(j)$ and with $M_{h+1}(j+1)$ do not appear alternately). Consider the following two cases (in Case i the property is violated by $M_h(j)$ and $M_{h+1}(j)$, and in Case ii the property is violated by $M_h(j)$ and $M_{h+1}(j+1)$):

Case (i). Assume that σ_1 and σ_2 are two overlapped portions of $M_h(j)$ with $M_{h+1}(j)$ and that no portion of $M_h(j)$ overlapped with $M_{h+1}(j+1)$ is between σ_1 and σ_2 (see Fig. 3). That is, there are no common points of $M_h(j)$ and $M_{h+1}(j+1)$ that dominate points in σ_1 and are dominated by points in σ_2.

Let B_1 be the set of points in $M_h(j)$ and between σ_1 and σ_2. Let B_2 be the set of points in $M_{h+1}(j)$ between σ_1 and σ_2 not including the overlapped portion of $M_h(j-1)$ and $M_{h+1}(j)$. According to properties P1 and P2, the top point and the bottom point of $M_h(j-1)$ are in M_{h+1}. Also, according to the assumption, $M_h(j)$ is the leftmost component chain of M_h for

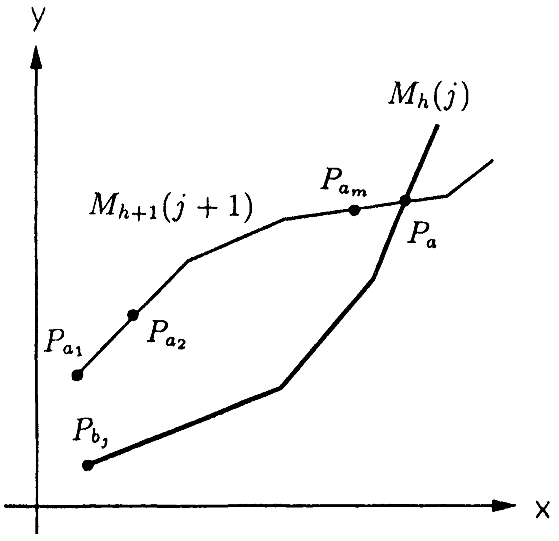


FIG. 2. *Proof of Lemma* 2.3, P1 *and* P2.

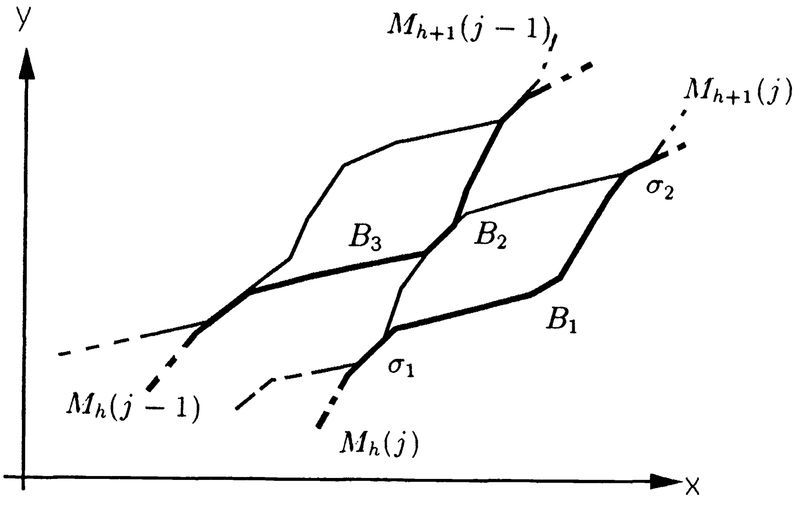


FIG. 3. *Proof of Lemma* 2.3, P3.

which P3 is not satisfied. Thus in $M_h(j-1)$ let B_3 be the portion between two portions of $M_h(j-1)$ overlapped with $M_{h+1}(j-1)$.

If we delete all points in B_2 from $M_{h+1}(j)$ and add all points in B_1 to $M_{h+1}(j)$, the union of σ_1, B_1, and σ_2 becomes a new overlapped portion of $M_h(j)$ and $M_{h+1}(j)$. Because M_h is a maximum h-chain, $|B_1| \geq |B_2|$. Thus the new M_{h+1} is still a maximum $(h+1)$-chain. However, B_3 is not in the new M_{h+1}; furthermore, $M_h(j-1)$ is the leftmost component chain of M_h. Therefore, property P3 is now satisfied. Repeating the above procedure from left to right, we can make all component chains of M_h satisfy property P3.

Case (ii). Assume that σ_1 and σ_2 are two portions of $M_h(j)$ overlapped with $M_{h+1}(j+1)$ and that no portion of $M_h(j)$ overlapped with $M_{h+1}(j)$ is between σ_1 and σ_2. Let B_1 be the set of points in $M_h(j)$ and between σ_1 and σ_2. Let B_2 be the set of points in $M_h(j+1)$ between σ_1 and σ_2. Let B_3 be the set of points in $M_{h+1}(j+1)$ between σ_1 and σ_2. As in Case i, we can make all component chains of M_h satisfy property P3. □

As explained in §2.1, we aim to construct a maximum $(h+1)$-chain if a maximum h-chain is available. We accomplish this task by deleting some points from the h-chain and adding new points to it. The collection of the h-chain, the added points, and the points deleted is called a *z-chain*, to be formalized below. Given a maximum h-chain M_h in ρ, we call the points in M_h *marked points* and the other points in ρ *unmarked points*. A path, called a *z-chain* of M_h (or simply a z-chain), is constructed as follows (see Fig. 4, where thin lines correspond to the chains and thick lines correspond to the z-chain). The path consists of an ordered set of points with the first point called the *starting point* and the last point called the *ending point*. Let P_i be a point on the path. Let P_j be the point next to P_i on the path. If P_j is an unmarked point, then P_j must dominate P_i. If P_j is marked, it either dominates or is dominated by P_i. If P_j dominates P_i, then we say P_j is an *up-point*; otherwise, P_j is a *down-point*. The starting point is defined to be an up-point. If P_i is an up-point and P_j is a down-point, we call P_i a *peak point*. If P_i is a down-point and P_j is an up-point, we call P_i a *valley point*. Each portion of points between a peak point and its following valley point in the path is called a *reverse chain*. We also define the *z-size* of a z-chain as the number of unmarked up-points minus the number of marked nonvalley down-points in the z-chain. A z-chain (of M_h) from point P_a to point P_b with maximum z-size among all z-chains with the same starting point P_a and ending point P_b is called a *maximum z-chain* (of M_h) from P_a to P_b. A maximum z-chain with maximum z-size among all possible maximum z-chains between all pairs of points in ρ is called a *global maximum z-chain* (of M_h).

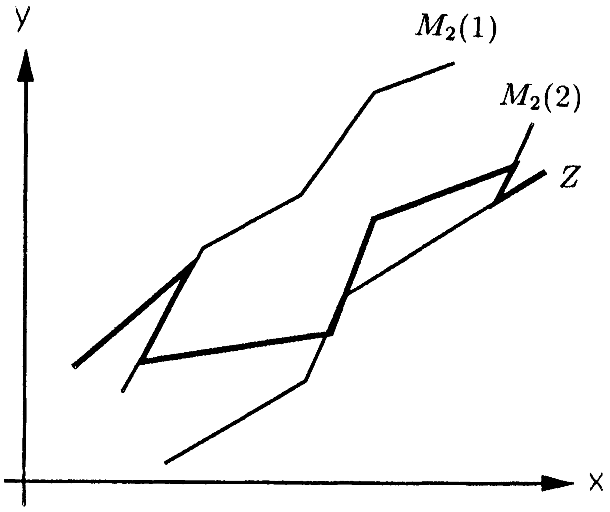


FIG. 4. *An example of a z-chain.*

Given a maximum h-chain M_h and a z-chain Z in ρ, if we change unmarked up-points to be marked and nonvalley down-points to be unmarked, the set of resultant marked points is called the *combination of M_h and Z*. The size of the combination of M_h and Z is the sum of the size of M_h and the z-size of Z.

LEMMA 2.4. *In a maximum z-chain no point is repeated.*

Proof. Assume that Z is a maximum z-chain and that point P_a in Z appears twice. Then there is a closed-loop z-chain Z_c whose z-size is larger than zero, for if its size is not positive it can be removed. Clearly, the combination of M_h and Z_c is an h-chain with size larger than that of M_h, which is a contradiction. □

LEMMA 2.5. *The combination of a maximum h-chain M_h and a global maximum z-chain is a maximum $(h + 1)$-chain.*

Proof. Clearly, the combination of a maximum h-chain and a z-chain is an $(h + 1)$-chain. In the following we will show that a maximum $(h + 1)$-chain can always be obtained from the combination of a maximum h-chain and a z-chain. Thus the combination of a maximum h-chain M_h and a global-maximum z-chain is a maximum $(h + 1)$-chain.

According to Lemma 2.2 there exists at least one max-$(h + 1)$-chain M_{h+1} that satisfies the partition property of M_h. We construct a z-chain as follows. First we choose the starting point of the z-chain. Consider the following two cases:

Case (i). If the bottom point P_b of component chain $M_{h+1}(j)$ is not a bottom point of M_h, we choose P_b as the starting point. Beginning from P_b, we let the points on $M_{h+1}(j)$ be up-points until we reach the top point of $M_{h+1}(j)$ or reach a point P_a that is a common point of $M_{h+1}(j)$ and a component chain of M_h, say, $M_h(j - 1)$.

Case (ii). If all bottom points of M_{h+1} are bottom points of M_h, we choose the point P_b' that is the bottom point of two component chains of M_{h+1} as the starting point. Without loss of generality, assume P_b' is the bottom point of $M_{h+1}(j)$, $M_{h+1}(j + 1)$, and $M_h(j)$. Also assume that the lowest point in $M_{h+1}(j)$, but not in $M_h(j)$, is lower than the lowest point in $M_{h+1}(j)$ but not in $M_h(j + 1)$ (see Fig. 5). Beginning from the starting point, we let the points in $M_{h+1}(j)$ be up-points until we reach the top point of $M_{h+1}(j)$ or reach a point P_a that is a common point of $M_{h+1}(j)$ and a component chain of M_h, say, $M_h(j - 1)$.

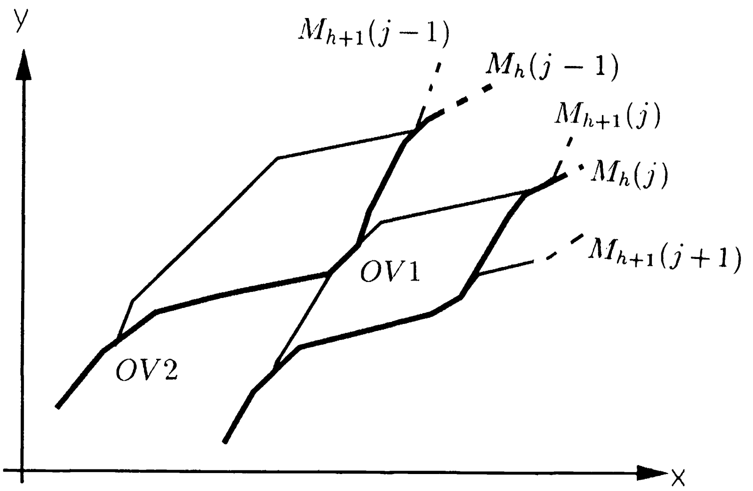


FIG. 5. *Proof of Lemma* 2.5.

Let the portion of $M_{h+1}(j)$ overlapped with $M_h(j - 1)$ be denoted by $OV1$. According to Properties P1 and P3, there is a portion of $M_{h+1}(j - 1)$ overlapped with $M_h(j - 1)$, denoted by $OV2$, below $OV1$. If the points in $OV2$ have been processed, we follow $M_{h+1}(j)$ up until we reach another overlapped portion. Otherwise, we follow $M_h(j - 1)$ down and make all points on $M_h(j - 1)$ and between $OV1$ and $OV2$ down-points; then we follow $M_{h+1}(j - 1)$ up until we reach another overlapped portion.

Each time we meet an overlapped portion we repeat the above procedure. Finally, we reach a top-point of M_{h+1} and obtain a z-chain Z. Obviously, the combination of Z and M_h is M_{h+1}. □

3. Algorithm MAX-THREE-CHAIN.

In this section we first discuss some properties of point dominance in the xy-plane. On the basis of these properties we present Algorithm MAXZ2 for finding a global-maximum z-chain of a given maximum two-chain. We then present Algorithm MAX-THREE-CHAIN for finding a maximum three-chain by using Algorithm MAXZ2.

3.1. Maximum level and minimum level.

Given a set of points ρ, the points that are not dominated by any other points are called the *maxima* of ρ. We can find the maxima of ρ where all maxima are at the same *level*, known as the *dominance hull* [6] of ρ. We then ignore the points at this level and recursively find other points at the same level for the remaining points. This process is repeated until all points in ρ have been assigned a level. We call the levels from bottom to top *levels* $1, 2, \ldots, s$, where level s contains points on the dominance hull of ρ [5].

In point set ρ, the points that do not dominate any other points are called the *minima* of ρ. With a process symmetric to the process for maximum levels we can partition point set ρ into, from bottom to top, *minimum levels* $1, 2, \ldots, s'$, where minimum level s' is the top minimum level. The maximum levels and minimum levels have properties stated in the following three lemmas.

LEMMA 3.1. *Each point at maximum level u, where $1 \leq u < s$, is dominated by at least one point at maximum level $u + 1$.*

Proof. Assume that point P_a is at level u. If P_a is not dominated by any point at level $u + 1$, then P_a should be at level $u + 1$. This is a contradiction. □

LEMMA 3.2. *Each point at minimum level u, where $1 < u \leq s'$, dominates at least one point at minimum level $u - 1$.*

The proof of Lemma 3.2 is similar to that of Lemma 3.1.

A chain C from a point P_a at maximum level u to P_b at maximum level v is called a *maximum-level solid chain* if C has one point at each maximum level from u to v. A *minimum-level solid chain* is similarly defined. The following lemma is based on Lemmas 3.1 and 3.2.

LEMMA 3.3. *In a point set the number of minimum levels and the number of maximum levels are equal (i.e., $s = s'$).*

Consider the following procedure: Beginning from the leftmost point of maximum level 1, choose one point at each maximum level such that each chosen point at maximum level u is the leftmost point at that level dominating the point chosen at maximum level $u - 1$, where $u > 1$. All chosen points form a maximum-level solid chain. This maximum-level solid chain is denoted by M_l. Symmetrically, beginning from the rightmost point of maximum level 1, choose one point at each maximum level such that each chosen point at maximum level u is the rightmost point at that level dominating the point chosen at maximum level $u - 1$, where $u > 1$. All chosen points form a maximum-level solid chain, which is denoted by M_r. Find the extended paths of M_l and M_r. The subset of the points on and between the two paths is called the *central subset* of ρ. Clearly, points of every maximum chain of ρ are in the central subset of ρ.

LEMMA 3.4. *If two points P_a and P_b are at the same maximum level and P_a is in the central subset of ρ and at minimum level l_a, then P_b is at a minimum level l_b, where $l_a \geq l_b$.*

Lemma 3.4 can be proved from the definitions.

3.2. Basic z-chains.

A z-chain can be partitioned into a collection of simpler shapes. This is a nice feature, since it simplifies the problem of obtaining a maximum z-chain. Consider the following three types of (sub-) z-chains:

(1) An A-*shaped z-chain* consists of two parts (see Fig. 6(a)): (a) a chain from the first point P_a of the z-chain to a marked point P_t, and (b) a reverse chain from P_t to the last point P_b of the z-chain.

(2) A V-*shaped z-chain* also consists of two parts (see Fig. 6(b)): (a) a reverse chain from a marked point P_a to a marked point P_v, and (b) a chain from P_v to the last point P_b of the z-chain.

(3) A C-*shaped z-chain* consists of three parts (see Fig. 6(c)): (a) a reverse chain from a marked point P_a to a marked point P_v, (b) a chain from P_v to a marked point P_t, and (c) a reverse chain from P_t to the last point P_b of the z-chain. Notice that all points in the chain from P_v to P_t in (b), except for P_v and P_t, are unmarked points.

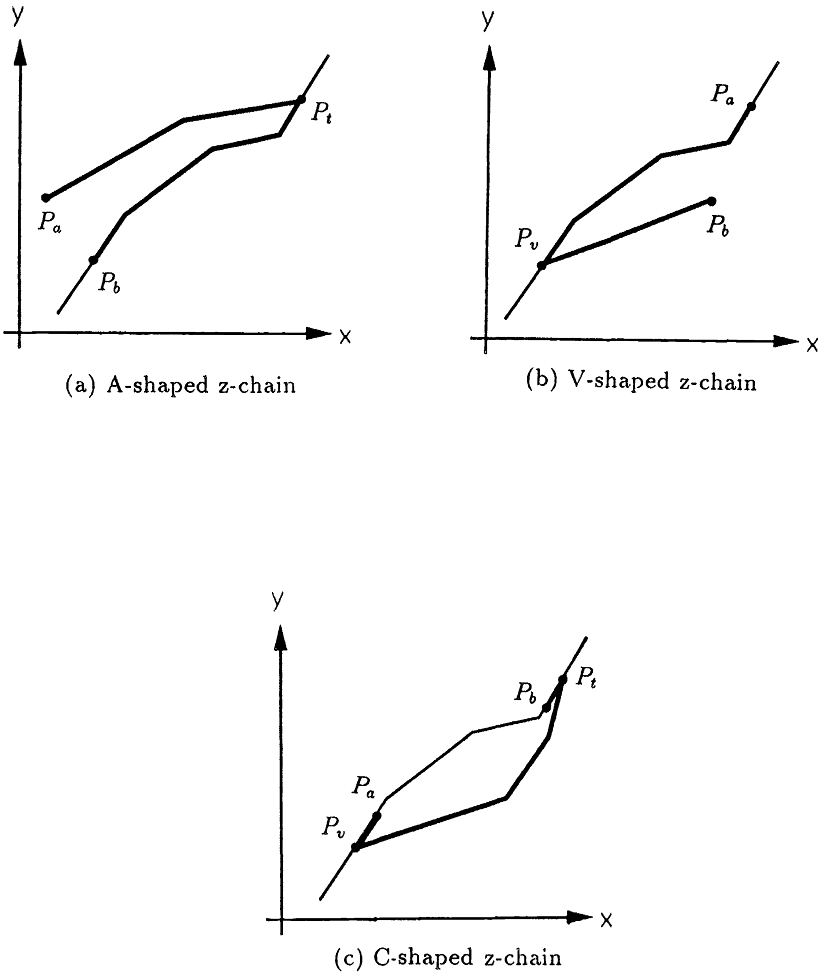


FIG. 6. *Three types of z-chains*: (a) *A-shaped z-chain*; (b) *V-shaped z-chain*; *and* (c) *C-shaped z-chain*.

Given a pair of points P_a and P_b, according to the definition there may exist several A-shaped z-chains from P_a to P_b. For simplicity, hereafter we use the term A-*shaped z-chain* to imply an A-shaped z-chain with maximum z-size among all such z-chains from P_a to P_b. For a V-shaped z-chain the situation is similar.

In the rest of this subsection we will elaborate on the properties of basic z-chains. Assume that we have already found maximum levels, minimum levels, and a maximum chain M_1. Let each point P_a in ρ at maximum level u and minimum level v be assigned two values, $L(a) = u$ and $l(a) = v$. We have the following lemma.

LEMMA 3.5. *For any point P_a and a marked point P_b, if P_a is not dominated by P_b, then an* A*-shaped z-chain from point P_a to P_b has z-size equal to $L(b) - L(a)$.*

Proof. For convenience we add two fictitious points $P_{n+1} = (n+1, n+1)$ and $P_0 = (0, 0)$ (i.e., P_{n+1} and P_0 do not make any contribution to the cardinality of any chain or z-chain) to ρ. Also, let P_{n+1} and P_0 be marked points. Note that P_0 is dominated by every point and that P_{n+1} dominates every point.

Because P_a is not dominated by P_b, we always can find a marked point P_c (e.g., P_{n+1}) such that (i) $L(c) > L(b)$, $L(c) > L(a)$, and (ii) there exists a maximum level solid chain C from point P_a to P_c (see Lemma 3.1). Thus chain C has $L(c) - L(a) + 1$ points.

The reverse chain from P_c to P_b has $L(c)-L(b)+1$ points because M_1 is also a maximum-level solid chain. According to the definitions the z-chain from P_a to P_c and then to P_b is an A-shaped z-chain with z-size $L(b)-L(a)$. It is easy to show that we cannot find any A-shaped z-chain from P_a to P_b with a larger z-size. □

The following lemma is similarly proved.

LEMMA 3.6. *For a marked point P_a and any point P_b, if P_a is not dominated by P_b, then a* V*-shaped z-chain from P_a to P_b has z-size equal to $l(b) - l(a)$.*

Given a pair of points P_a and P_b, we use $MZ(P_a, P_b)$ to denote a maximum z-chain from P_a to P_b.

LEMMA 3.7. *For an arbitrary point P_a and a marked point P_b, if P_a is dominated by P_b, then either* (1) *$MZ(P_a, P_b)$ is a chain from P_a to P_b or* (2) *there exists a point P_d at maximum level $L(b)$ such that $MZ(P_a, P_b)$ is an* A*-shaped z-chain from P_a through P_d to P_b.*

Proof. Because P_b is marked, there are only two possible types of z-chains from P_a to P_b: a chain or an A-shaped z-chain. In the following we show that if $MZ(P_a, P_b)$ is an A-shaped z-chain, then $MZ(P_a, P_b)$ should have a point at maximum level $L(b)$.

Let P_{b_1} be the highest point (i.e., with the largest y-coordinate) in $MZ(P_a, P_b)$ dominated by P_b. Let the number of points in $MZ(P_a, P_b)$ from P_a to P_{b_1} be m; then $MZ(P_a, P_b)$ has z-size at least equal to m. Let P_{b_2} be the lowest point in $MZ(P_a, P_b)$ not dominated by P_b. If P_{b_2} is at a maximum level $u > L(b)$, then the z-size of $MZ(P_a, P_b)$ is $m + L(b) - u < m$, a contradiction. Thus P_{b_2} is at a maximum level $u \leq L(b)$ and $MZ(P_a, P_b)$ has a point at maximum level $L(b)$. □

The following lemma is similarly proved.

LEMMA 3.8. *For a marked point P_a and a point P_b, if P_a is dominated by P_b, then either* (1) *$MZ(P_a, P_b)$ is a chain from P_a to P_b or* (2) *there exists a point P_d at minimum level $l(a)$ such that $MZ(P_a, P_b)$ is a* V*-shaped z-chain from P_a through P_d to P_b.*

3.3. Algorithms MAXZ2 and MAX-THREE-CHAIN. We first discuss Algorithm MAXZ2, which finds a global maximum z-chain of a given maximum two-chain M_2. Then we show how to combine a global maximum z-chain with a maximum two-chain to obtain a maximum three-chain.

Consider a given global maximum z-chain Z of M_2 starting from a point P_S, as shown in Fig. 7. Without loss of generality, assume P_S is in $R_1(1)$. For simplicity we add fictitious points $P_0 = (0, 0)$ and $P_{n+1} = (n+1, n+1)$ to ρ. (However, hereafter each time we partition a point subset into levels, we do not take P_0 and P_{n+1} into consideration.) Then Z starts from P_0 and terminates at P_{n+1}. For each point P_i in ρ let $z(i)$ denote the z-size of the maximum z-chain from P_0 to P_i. Thus, if point P_i is in Z, $z(i)$ is equal to the z-size of Z from point P_S to P_i. Z may intersect M_2 many times. Let $P_{t_\ell^j}$ denote the jth peak point of Z on $M_2(\ell)$, where $\ell = 1$ or 2. Let $P_{v_\ell^j}$ denote the jth valley point of Z on $M_2(\ell)$.

If we say that A-shaped and V-shaped z-chains are degenerate types of C-shaped z-chains, then a maximum z-chain of M_2 is partitioned by M_2 into chains and C-shaped z-chains. For example, Z is partitioned into the following sets (see Fig. 8):

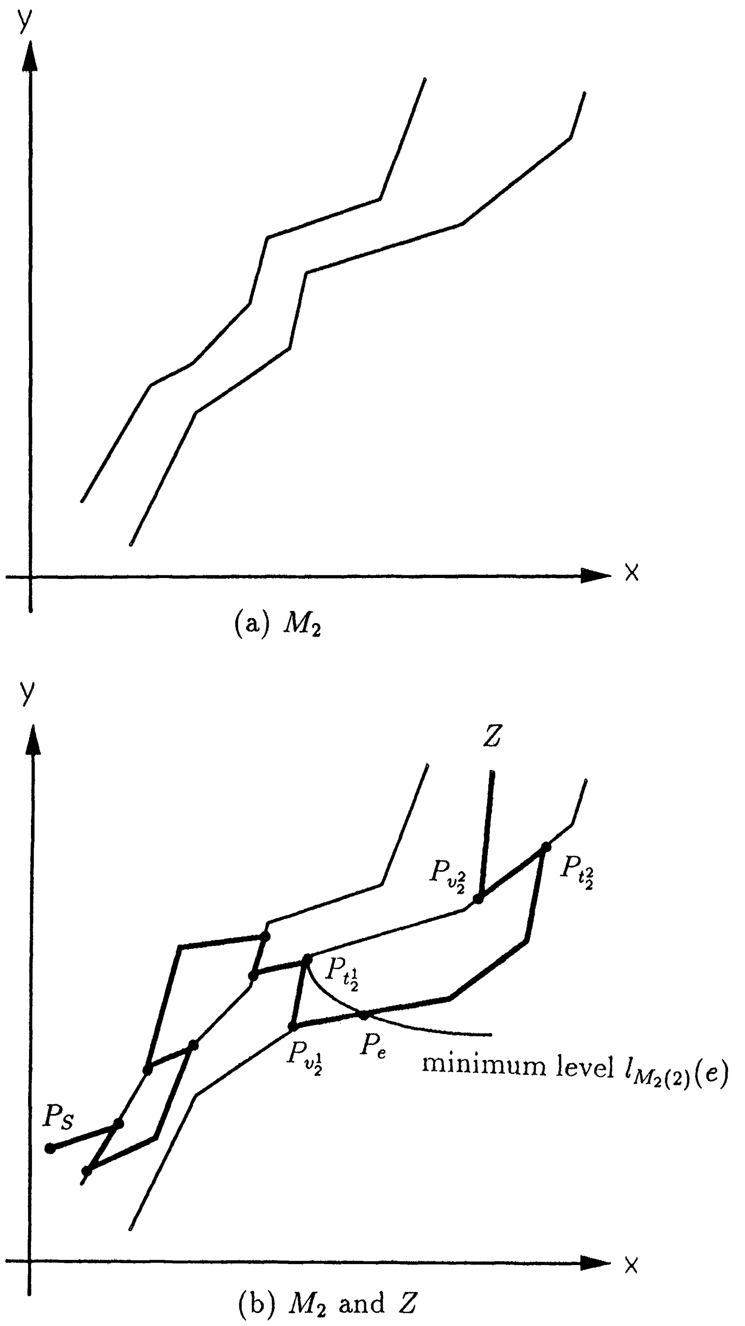


FIG. 7. *M_2 and global-maximum z-chain* Z: (a) M_2 *and* (b) M_2 *and* Z.

(DZ1) a set of chains in $R_2(2)$;
(DZ2) a set of chains and C-shaped z-chains in $R_2(1) \cup M_2(1)$;
(DZ3) a set of chains and C-shaped z-chains in $M_2(2) \cup R_2(3)$.
Consider two points P_a and P_b in Z. Assume that $z(a)$ has already been found. If

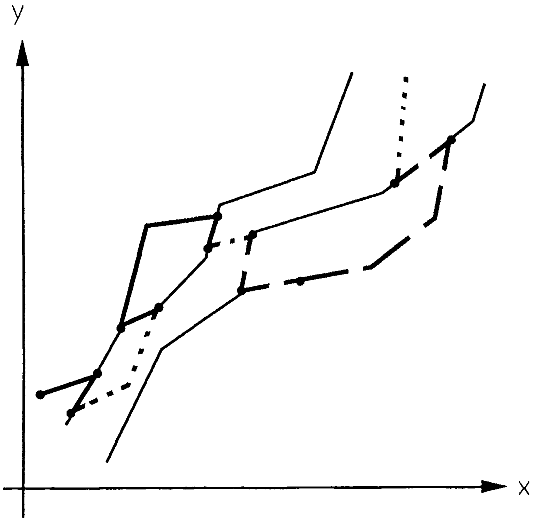


FIG. 8. *Partition of Z.*

$MZ(P_a, P_b)$ is a portion of Z, $z(b)$ is equal to $z(a)$ plus the z-size of $MZ(P_a, P_b)$. If P_a and P_b are both in $R_2(1)$ (or both in $R_2(3)$) and if $MZ(P_a, P_b)$ does not intersect M_2, then $MZ(P_a, P_b)$ is a chain. If P_a and P_b are in $M_2(1) \cup R_2(2) \cup M_2(2)$, according to the above discussion there are two possibilities for $MR(P_a, P_b)$: a chain or a C-shaped z-chain. Hence we can construct algorithm MAXZ2 as follows.

Each time we process a point P_i we do the following:

(ZP1) *Assume P_i is in a chain stated in* DZ1. We find a point P_j dominated by P_i with maximum z value. Let $z_1(i) = z(j) + 1$ if P_i is unmarked; otherwise, let $z_1(i) = z(j)$.

(ZP2) *If P_i is a marked point, assume P_i is the ending point of a* C*-shaped z-chain.* We have to find a marked point P_j in the same component chain with P_i and dominated by P_i, such that $z(j)$ plus the z-size of $MZ(P_j, P_i)$ is maximum among all possible P_j's. We denote such a value by $z_2(i)$. Notice that all unmarked points in such C-shaped z-chains are in $R_2(1)$ (or $R_2(3)$). Else, if P_i is unmarked, it is handled as in ZP1.

Then let $z(i) = \max(z_1(i), z_2(i))$. After all points are processed, the point, except for P_{n+1}, with the maximum z value in ρ is the last point of a global-maximum z-chain. We trace back every point in Z starting from the last point of Z.

Algorithms in [2] or [5] can be used to handle case ZP1. However, for ZP2 we need some detailed discussion. We use the C-shaped z-chain from $P_{t_2^1}$ to $P_{v_2^2}$ in Fig. 7 as an example to explain how to find value $z(v_2^2)$ of $P_{v_2^2}$ if it is assumed that value $z(t_2^1)$ of $P_{t_2^1}$ is known.

Let $\rho_{M_2(1)}$ denote the point subset $\rho - M_2(2)$, and let $\rho_{M_2(2)}$ denote the point subset $\rho - M_2(1)$. Obviously, the two point subsets have common points. We find maximum levels and minimum levels for both $\rho_{M_2(1)}$ and $\rho_{M_2(2)}$. It is clear that $M_2(\ell)$, where $\ell = 1$ or 2, is a maximum chain of point set $\rho_{M_2(\ell)}$; otherwise, M_2 is not a maximum two-chain. To each point P_i in $R_2(1) \cup M_2(1)$, values $L_{M_2(1)}(i)$ and $l_{M_2(1)}(i)$ are assigned to denote that P_i is at maximum level $L_{M_2(1)}(i)$ and minimum level $l_{M_2(1)}(i)$ of point set $\rho_{M_2(1)}$. Similarly, to each point P_j in $M_2(2) \cup R_2(3)$ values $L_{M_2(2)}(j)$ and $l_{M_2(2)}(j)$ are assigned. In the C-shaped

z-chain from $P_{t_2^1}$ to $P_{v_2^2}$ in Fig. 7, according to Lemma 3.8 there is a point P_e at the minimum level $l_{M_2(2)}(t_2^1)$ of $\rho_{M_2(2)}$ such that $MZ(P_{t_2^1}, P_e)$ is a V-shaped z-chain. Clearly, the z-size of $MZ(P_{t_2^1}, P_e)$ is 0 and $z(e) = z(t_2^1)$ (Lemma 3.6). Consider the following two cases:

(C1) *P_e and $P_{v_2^2}$ cross each other.* P_e also crosses $P_{t_2^1}$. According to the proof of Lemma 3.5 there is an A-shaped z-chain from P_e to $P_{v_2^2}$ and then to each point P_i in $M_2(2)$ between $P_{v_2^2}$ and $P_{t_2^1}$. For each point P_i in $M_2(2)$ between $P_{t_2^1}$ and $P_{v_2^2}$, $z_2(i)= z(e) + L_{M_2(2)}(i) - L_{M_2(2)}(e)$. Because $z_2(t_2^1)= z(e) + L_{M_2(2)}(t_2^1) - L_{M_2(2)}(e)$, we have $z_2(i) = z_2(t_2^1) + d + 1$, where d is the number of points in $M_2(2)$ between $P_{t_2^1}$ and P_i. Notice that $z(t_2^1) = z_1(t_2^1) \geq z_2(t_2^1)$. However, there are two problems: (i) if there are more than one point at minimum level $l_{M_2(2)}(t_2^1)$ of $\rho_{M_2(2)}$, we do not know which point is P_e; (ii) the above $z_2(i)$ equations may not be true for points in $M_2(2)$ higher than $P_{t_2^2}$. Thus we assign each point P_i a proper $z_2(i)$ value as follows. For each point P_g in $R_2(3)$ we denote the intersection point of $M_2(2)$ with the maximum-level solid chain from P_g to $M_2(2)$ by $I(P_g, M_2(2))$. We construct a height-balanced binary priority tree data structure T_1. Each leaf $\mathcal{L}$ of T_1 represents a point P_g in $R_2(3)$ and is assigned a weight $W(\mathcal{L}(g))$. The root $\mathcal{R}$ of T_1 is also assigned a weight $W(\mathcal{R})$. When P_i is processed, the sum of the weights $W(\mathcal{L}(g))$ and $W(\mathcal{R})$, denoted by $E(g)$, represents the sum of $z(g)$ and the z-size of the A-shaped z-chain from P_g through $I(P_a, M_2(2))$ to P_i only if $I(P_a, M_2(2))$ is higher than P_i. Initially, T_1 has no leaf and $W(\mathcal{R}) = 0$. When we process P_i, we first increase the weight of the root by 1. For each point P_g at minimum level $l_{M_2(2)}(i)$ we calculate each $z(g) + L_{M_2(2)}(i) - L_{M_2(2)}(g) - W(\mathcal{R})$ value, create a leaf $\mathcal{L}$ in T_1 to represent P_g, and store the value. Then in $O(\log n)$ time we can find the leaf in T_1 with maximum E value. Let P_a be the point represented by the leaf. If $I(P_a, M_2(2))$ is a point higher than P_i, then $z_2(i) = E(a)$; otherwise, delete the leaf representing P_a and find another maximum E value again.

(C2) *P_e is dominated by $P_{v_2^2}$.* According to Lemma 3.7 there is a point P_f (in Z) at maximum level $L(v_2^2)$ of $\rho_{M_2(2)}$. Because $P_{v_2^2}$ is in the central region of $\rho_{M_2(2)}$, from Lemma 3.4, P_f is at a minimum level of $\rho_{M_2(2)}$ lower than that of $P_{v_2^2}$. Thus according to C1 when point $P_{v_2^2}$ is processed, a leaf corresponding to point P_f has already been created in T. Furthermore, $MZ(P_e, P_f)$ is a chain. Therefore, we can obtain $z(f)$ and then $z(v_2^2)$.

On the basis of the above discussion we introduce a type of level called a *master level*, as follows. First we find minimum levels of $M_2(1) \cup R_2(2) \cup M_2(2)$. Each point at minimum level u of $M_2(1) \cup R_2(2) \cup M_2(2)$ is assigned to master level u. For each point P_i in $M_2(1)$ at master level u we assign all points in $R_2(1)$ at minimum level $l_{M_2(1)}(i)$ to master level u. Points in $R_2(3)$ are similarly assigned. Then Algorithm MAXZ2 has the following steps:

(XZ1) Assign each point P_i in $R_2(1) \cup M_2(1)$ values $L_{M_2(1)}(i)$ and $l_{M_2(1)}(i)$. Assign each point P_j in $M_2(2) \cup R_2(3)$ values $L_{M_2(2)}(j)$ and $l_{M_2(2)}(j)$.

(XZ2) Assign points in ρ to master levels. Assume there are m master levels. Let master level 1 be the bottom level.

(XZ3) For each point P_f in $R_2(1)$ find $I(P_f, M_2(1))$. If there are several possible intersection points, take the lowest one. Similarly, for each point P_g in $R_2(3)$ find $I(P_g, M_2(2))$.

(XZ4) Assign $z(0) = 0$ for point P_0.

(XZ5) For $u = 1$ to m do the following (for each master level u):

(XZ5.1) Find z_1 values for all points at master level u.

(XZ5.2) Find z_2 values for all marked points at master level u.

(XZ5.3) For each marked point P_i at master level u let $z(i)=\max(z_1(i), z_2(2))$. For each unmarked point P_i at master level u let $z(i)=z_1(i)$.

(XZ6) Carry out the process to the top master level; the point with maximum z value is the last point of a maximum z-chain. By tracing back from this point a maximum z-chain can be found. (To trace every point in the maximum z-chain from its last point we have to add some pointers in steps XZ5.1, XZ5.2, and XZ5.3. However, for simplicity we do not discuss these pointers in this paper.)

Finally, Algorithm MAX-THREE-CHAIN has the following steps:

(MKC1) Call Algorithm MAX-TWO-CHAIN (see [5]) to find a maximum two-chain M_2.
(MKC2) Find a global-maximum z-chain of M_2.
(MKC3) Combine the global maximum z-chain and M_3 to obtain a maximum three-chain.

According to previous discussion we have the following lemma:

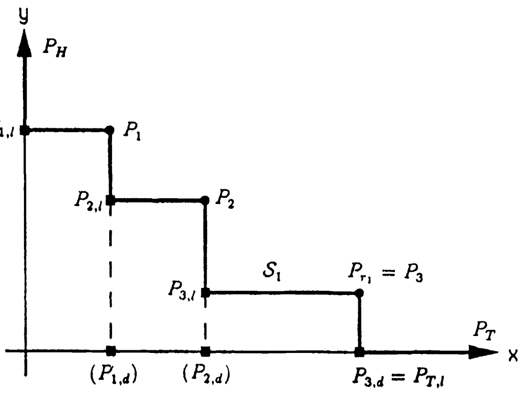
LEMMA 3.9. *Algorithm* MAX-THREE-CHAIN *finds a maximum three-chain of point set* ρ.

4. Implementation and time complexity. In steps XZ1 and XZ2 of Algorithm MAXZ2 we have to partition point set ρ into levels. By using an algorithm in [5] we can partition point set ρ into s maximum levels. With the staircase data structure in [5] step XZ5.1 takes $O(\log n)$ time for each point. For completeness we briefly describe the algorithm and the data structure again in the following two subsections. Then we discuss the time complexity of Algorithm MAX-THREE-CHAIN.

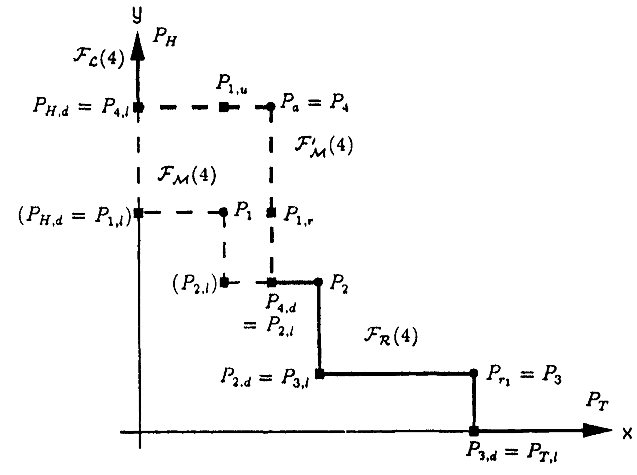
4.1. Implementation of Algorithm MAX-LEVEL. Recall that points in ρ are assumed to be sorted in ascending order of x-coordinates such that $x_1 < x_2 < \cdots < x_n$. We use a plane sweep technique [6] to scan the points from right to left and perform the following operations. Suppose we have processed points $P_n, P_{n-1}, \ldots, P_{i+1}$ and have obtained l lists of points that belong to the same levels. For each list we keep the y-coordinate of the last scanned point. These y-values are maintained as a height-balanced tree T_2. In processing P_i we use T_2 to first identify the list $\mathcal{L}(k)$ whose associated y-value is immediately below y_i. We then insert P_i into list $\mathcal{L}(k)$ and replace the associated y-value by y_i. If y_i is smaller than all y-values in T_2, the $(l+1)$st list is created and its associated y-value is set to y_i. At the end of the process the number s of the lists created is the total number of levels. We then reindex the lists, so that the innermost level (created last) has index 1 and the outermost one has index s. We call the above algorithm MAX-LEVEL.

4.2. Staircase. Let ρ_u represent the set of points at uth master level, where $u = 1, 2, \ldots, s$. To simplify the discussion we *assume* that $|\rho_u| = r_u$ and the points in ρ are reindexed as $\rho_1 = \{P_1, P_2, \ldots, P_{r_1}\}$, $\rho_2 = \{P_{r_1+1}, \ldots, P_{r_1+r_2}\}, \ldots, \rho_s = \{P_{r_1+\cdots+r_{s-1}+1}, \ldots, P_{r_1+\cdots+r_s}\}$, so that $x_1 < x_2 < \cdots < x_{r_1}$, $x_{r_1+1} < x_{r_1+2} < \cdots < x_{r_1+r_2}$, etc. Consider the points in ρ_1. Let $P_{i,d} = (x_i, 0)$ be the *downward vertical intersection* (d-intersection for short) of P_i, $i = 1, 2, \ldots, r_1$. Let $P_{1,l} = (0, y_1)$, and let $P_{i,l} = (x_{i-1}, y_i)$ be the *leftward horizontal intersection* (l-intersection for short) of the left point P_1 and other points P_i in ρ, respectively, on the y-axis and the vertical line segment $\overline{P_{i-1}, P_{i-1,d}}$, $i = 2, 3, \ldots, r_1$ (see Fig. 9(a)). The rectilinear path $(P_{1,l} P_1 P_{2,l} P_2 \cdots P_{i,l} P_i P_{i,d})$, abbreviated as $(P_{1,l} \cdots P_{i,d})$, is referred to as the *staircase* $\mathcal{S}(P_i)$, $i = 1, 2, \ldots, r_1$. For convenience we augment the staircase by two points, $P_H = (0, \infty)$ and $P_T = (\infty, 0)$, and denote the staircase as $(P_H P_{1,l} P_1 \cdots P_i P_{T,l} P_T)$, where $P_{T,l} := P_{i,d}$. Furthermore, we adopt the convention that each point P_j in $\rho \cup \{P_H, P_T\}$ on a staircase has both d- and l-intersections on the staircase except P_H, which has only d-intersection $P_{H,d} := P_{1,l}$, and P_T, which has only l-intersection $P_{T,l} := P_{i,d}$. That is, we shift each $P_{j,d}$ up to the staircase and let new $P_{j,d} := P_{j+1,l}$, where $j = 1, 2, \ldots, i-1$. Staircase $\mathcal{S}(P_{r_1})$ is called the staircase of level 1, denoted $\mathcal{S}_1$, and is shown in Fig. 9(a).

Consider now the points in ρ_2, i.e., $P_{r_1+1}, P_{r_1+2}, \ldots, P_{r_1+r_2}$. Imagine dropping downward a vertical line V_i from each point P_i, $r_1 < i \le r_1 + r_2$. The point in $\mathcal{S}_1$ that is hit by V_i is



(a) Staircase $\mathcal{S}_1$ (where $r_1 = 3$)



(b) Staircase $\mathcal{S}(P_4)$ (where $r_1 = 3$)

FIG. 9. *Example of a staircase*: (a) *staircase* $\mathcal{S}_1$ (*where* $r_1 = 3$) *and staircase* $\mathcal{S}(P_4)$ (*where* $r_1 = 3$).

the d-intersection, denoted by $P_{i,d}$, of P_i on $\mathcal{S}_1$ (see Fig. 9(b)). For $P_i \in \rho_2$, the l-intersection on $\mathcal{S}_1$, $P_{i,l}$, is defined in the same way as for the points in ρ_1. Let P_a be the left point in ρ_2. The l- and d-intersections, $P_{a,l}$ and $P_{a,d}$, respectively, partition $\mathcal{S}_1$ into three *substaircases*: $\mathcal{F}_{\mathcal{L}}(P_a)$, $\mathcal{F}_{\mathcal{M}}(P_a)$, and $\mathcal{F}_{\mathcal{R}}(P_a)$, where $\mathcal{F}_{\mathcal{L}}(P_a) = (P_H P_{a,l})$ is the *leading portion* of $\mathcal{S}_1$, $\mathcal{F}_{\mathcal{M}}(P_a) = (P_{a,l} \cdots P_{a,d})$, is the *middle portion* of $\mathcal{S}_1$, and $\mathcal{F}_{\mathcal{R}}(P_a) = (P_{a,d} \cdots P_{T,l} P_T)$ is the *trailing portion* of $\mathcal{S}_1$. Let $P_{k,l}$ and P_k be two consecutive points in $\mathcal{S}_1$ such that $x_{k,l} < x_a < x_k$. The *upward vertical projection* (u-projection for short) of all the points excluding the endpoints $P_{a,l}$ and $P_{a,d}$ in $\mathcal{F}_{\mathcal{M}}(P_a)$ on the horizontal line segment $\overline{P_{a,l}, P_a}$

gives rise to the u-projections $P_{1,u}, P_{2,u}, \ldots, P_{k-1,u}$. Similarly, the *rightward horizontal projection* (r-projection for short) of all the points excluding the two endpoints in $\mathcal{F}_{\mathcal{M}}(P_a)$ on the vertical line segment $\overline{P_a, P_{a,d}}$ yields r-projections $P_{1,r}, P_{2,r}, \ldots, P_{k-1,r}$. The staircase $\mathcal{S}(P_a)$ associated with P_a is obtained from $\mathcal{S}_1$ as the concatenation of $\mathcal{F}_{\mathcal{L}}(P_a)$, $\mathcal{F}'_{\mathcal{M}}(P_a)$ and $\mathcal{F}_{\mathcal{R}}(P_a)$, where $\mathcal{F}'_{\mathcal{M}}(P_a)$ is the substaircase $(P_{a,l}P_{1,u} \cdots P_{k-1,u}P_aP_{1,r} \cdots P_{k-1,r}P_{a,d})$ derived from projecting $\mathcal{F}_{\mathcal{M}}(P_a)$ onto $\overline{P_{a,l}, P_a}$ and $\overline{P_a, P_{a,d}}$. Thus $\mathcal{S}(P_a)$ contains not only point P_a and its l- and d-intersections $P_{a,l}$ and $P_{a,d}$ but also the leading and trailing portions of $\mathcal{S}_1$ and projections of the points in the middle portion of $\mathcal{S}_1$.

Let P_a be the left point in ρ_j, $j > 1$. In general, the staircase $\mathcal{S}(P_i)$ of any point $P_i \in \rho_j$ is a rectilinear path of the form $(P_H\ P_{a,l}\ \mathcal{U}_a\ P_a\ \mathcal{R}_a\ P_{a+1,l}\ \mathcal{U}_{a+1}\ P_{a+1}\ \mathcal{R}_{a+1} \cdots P_{i,l}\ \mathcal{U}_i\ P_i\ \mathcal{R}_i\ P_{i,d}\ \mathcal{F}_{\mathcal{R}}(P_i))$, where the $\mathcal{U}$'s, $\mathcal{R}$'s, and $\mathcal{F}_{\mathcal{R}}(P_i)$ are, respectively, u-projections, r-projections, and the trailing portion of the previous staircase $\mathcal{S}(P_{i-1})$. Note that $\mathcal{F}_{\mathcal{L}}(P_i) = (P_H\ P_{a,l}\ \mathcal{U}_a\ P_a\ \mathcal{R}_a\ P_{a+1,l}\ \mathcal{U}_{a+1}\ P_{a+1}\ \mathcal{R}_{a+1} \cdots P_{i,l})$ and $\mathcal{F}_{\mathcal{M}}(P_i) = (P_{i,l}\ \mathcal{U}_i\ P_i\ \mathcal{R}_i\ P_{i,d})$. Now let us consider how to obtain $\mathcal{S}(P_{i+1})$ from $\mathcal{S}(P_i)$.

We first find the l-intersection $P_{i+1,l}$ on VL $= \overline{P_i, P_{i,d}}$ and the d-intersection $P_{i+1,d}$ on $\mathcal{S}(P_i)$. Let R' and R'' be two consecutive points on VL containing $P_{i+1,l}$, and let Q' and Q'' be two consecutive points on $\mathcal{S}(P_i)$ containing $P_{i+1,d}$. Thus $\mathcal{F}_{\mathcal{M}}(P_{i+1})$ is the substaircase $(R'' \cdots Q')$ of $\mathcal{S}(P_i)$, and the new $\mathcal{F}'_{\mathcal{M}}(P_{i+1})$ is obtained by projecting all points (including projections) in $\mathcal{F}_{\mathcal{M}}(P_{i+1})$ vertically upward and horizontally rightward onto $\overline{P_{i+1,l}, P_{i+1}}$, $\overline{P_{i+1}, P_{i+1,d}}$, respectively. Note that points on the same vertical line segment have the same u-projection and that points on the same horizontal line segment have the same r-projection. The staircase $\mathcal{S}(P_{i+1})$ is the concatenation of the leading portion $(P_H \cdots R')$ of $\mathcal{S}(P_i)$, $\mathcal{F}'_{\mathcal{M}}(P_{i+1})$, and the trailing portion $(Q'' \cdots P_T)$ of $\mathcal{S}(P_i)$. According to the convention adopted we shift certain points each time a new staircase is obtained. Specifically, let $P_f \in \rho \cup \{P_H, P_T\}$ be the rightmost point on $\mathcal{S}(P_i)$ whose y-coordinate is greater than y_{i+1}, and let $P_g \in \rho \cup \{P_H, P_T\}$ be the leftmost point on $\mathcal{S}(P_i)$ whose x-coordinate is greater than x_{i+1}. We let $P_{i+1,l}$ and $P_{i+1,d}$ be the new $P_{f,d}$ and $P_{g,l}$, respectively; that is, we shift $P_{f,d}$ and $P_{g,l}$ from their original positions to $P_{i+1,l}$ and $P_{i+1,d}$, respectively. For example, in Fig. 9(b), when $\mathcal{S}(P_4)$ is obtained $P_{2,l}$ is shifted to $P_{4,d}$ and $P_{H,d}$ is shifted to $P_{4,l}$.

4.3. Time complexity.

We first discuss the time complexity of Algorithm MAXZ2. In steps XZ1 and XZ2 we have to partition point set ρ into levels. By using Algorithm MAX-LEVEL and its modification we can complete step XZ1 and XZ2 in $O(n \log n)$ time, where $n = |\rho|$. With the information of maximum levels of $\rho_{M_2(1)}$ and $\rho_{M_2(2)}$, step XZ3 takes linear time. With the staircase data structure step XZ5.1 takes $O(\log n)$ time for each point [5]. Step XZ5.2 takes $O(n \log n)$ time for Algorithm MAXZ2. Thus Algorithm MAXZ2 takes $O(n \log n)$ time.

In step MKC1 of Algorithm MAX-THREE-CHAIN, Algorithm MAX-TWO-CHAIN is called and takes $O(n \log n)$ time. In step MKC3 the combination of global maximum z-chain and M_2 takes linear time. Hence Algorithm MAX-THREE-CHAIN runs in $O(n \log n)$ time.

In [5] it is shown that the maximum two-chain problem has an $\Omega(n \log n)$ lower bound. We extend an $\Omega(n \log n)$ lower bound for the maximum three-chain problem as follows (as done in [5] for the maximum two-chain problem). Given a point set ρ we can construct in linear time a chain $\bar{\rho}$ in which each point is crossing with all points in ρ (see Fig. 10). A maximum three-chain of $\rho \cup \bar{\rho}$ should include $\bar{\rho}$ and a maximum two-chain of ρ. So we can find a maximum two-chain of ρ by applying any maximum three-chain algorithm to $\rho \cup \bar{\rho}$. Thus the following theorem is established.

THEOREM 4.1. *Algorithm* MAX-THREE-CHAIN *finds a maximum three-chain of a point set ρ and runs in $\Theta(n \log n)$ time, where $n = |\rho|$.*

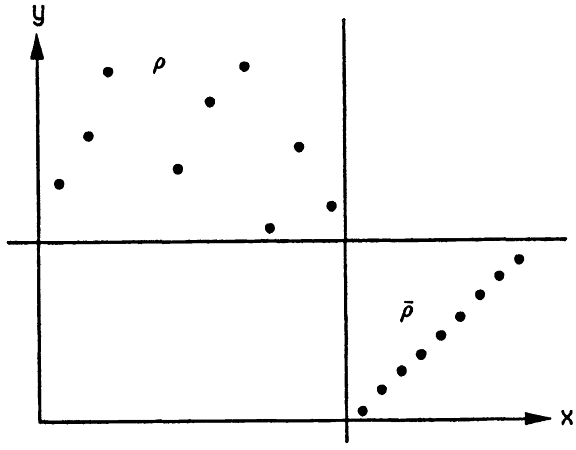


FIG. 10. *p and $\bar{p}$.*

5. Discussion. In algorithm MAXZ2 we *mainly* process points in $M_2(1) \cup R_2(2) \cup M_2(2)$. The purposes of processing the points in $R_2(1)$ and $R_2(3)$ are (1) to help to determine z_2 values of marked points and (2) to handle the case for which the starting point and the ending point may be in $R_2(1)$ or $R_2(3)$. In Case (1) we use C-shaped z-chain and z_2 values to pass the possible contribution of $R_2(1)$ and $R_2(3)$ on Z to the processing of $M_2(1) \cup R_2(2) \cup M_2(2)$.

We can extend Algorithm MAXZ2 to MAXZh, where $h > 2$, which finds a maximum z-chain of a given maximum h-chain as follows. Consider P_a and P_b in the same component chain $M_h(j)$, and P_a dominates P_b. A *right general* C*-shaped z-chain* is a z-chain starting from point P_a to P_b all of whose points except for P_a and P_b are in $R_h(j+1) \cup M_h(j+1) \cup R_h(j+2) \cup \cdots \cup R_h(h+1)$. A *left general* C*-shaped z-chain* is symmetrically defined. We preprocess the points to find all such z-chains. Then we *mainly* process points in each $M_h(i-1) \cup R_h(i) \cup M_h(i)$ for $i = 2$ to h in a manner similar to Algorithm MAXZ2. We use right general C-shaped z-chains to pass the possible contribution of the points on the right-hand side of $\mathcal{P}(M_h(i))$ to the processing of $M_h(i-1) \cup R_h(i) \cup M_h(i)$. Left general C-shaped z-chains are also used for a similar reason.

Recall that the dominance property of marked points in z-chains may be reversed (i.e., the order of marked points in z-chains may be different from their order in chains). Hence in the process of finding a global maximum z-chain we say that the dominance property between each pair of points on the different sides of a component chain (i.e., in different $R_h(i)$ regions) has been destroyed. Basically, the dominance property provides the efficiency in finding the maximum chain, the maximum two-chain, and the maximum three-chain; that is, it enables us to solve the problems in $O(n \log n)$ time. In Algorithm MAXZ2 for the maximum three-chain problem we *mainly* process points in region $M_2(1) \cup R_2(2) \cup M_2(2)$ and use a C-shaped z-chain to combine results from $R_2(1)$ and $R_2(3)$. Because the dominance property in each region still holds, we can process points master level by master level and use the dominance property to reduce the time complexity.

For $k > 3$, for each region $M_h(i-1) \cup R_h(i) \cup M_h(i)$ there may be several regions on the left- (or right-hand) hand side of $\mathcal{P}(M_h(i-1))$ (or $\mathcal{P}(M_h(i))$.) Because the dominance property between regions has been destroyed, we cannot use dominance property to reduce the time complexity of finding left (or right) general C-shaped z-chains. The number of starting point and ending point pairs of general C-shaped z-chains is $O(n^2)$. The time complexity of MAXZh is $O(n^2)$. The time complexity of Algorithm MAX-k-CHAIN, which finds a maximum k-chain by using Algorithm MAXZh, is thus $O(n^2)$ for any fixed k. This time is equivalent to that obtained in [4] (or in [7]). In conclusion, we believe that an improved algorithm for the maximum k-chain problem with $k > 3$ needs concepts beyond the z-chain properties.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974.

[2] M. J. Atallah and S. R. Kosaraju, *An efficient algorithm for maxdominance, with applications*, Algorithmica, 4 (1989), pp. 221–236.

[3] J. Cong and C. L. Liu, *On the k-layer planar subset and via minimization problems,* IEEE Trans. Computer-Aided Design, 10 (1991), pp. 972–981.

[4] F. Gavril, *Algorithms for maximum k-colorings and k-coverings of transitive graphs*, Networks, 17 (1987), pp. 465–470.

[5] R. D. Lou, M. Sarrafzadeh, and D. T. Lee, *An optimal algorithm for the maximum two-chain problem,* in Proc. 1st SIAM–ACM Conference on Discrete Algorithms (SODA), Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990; SIAM J. Discrete Math., 5 (1992), pp. 284–304.

[6] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[7] C. S. Rim, T. Kashiwabara, and K. Nakajima, *Exact algorithms for multilayer topological via minimization*, IEEE Trans. Computer-Aided Design, 8 (1989), pp. 1165–1184.

[8] M. Sarrafzadeh and D. T. Lee, *New approach to topological via minimization*, IEEE Trans. Computer-Aided Design, 8 (1989), pp. 890–900.

[9] ———, *Topological via minimization revisited*, IEEE Trans. Comput., 40 (1991), pp. 1307–1312.

[10] M. Sarrafzadeh and R. D. Lou, *Maximum k-coverings of weighted transitive graphs with applications*, Algorithmica, 9 (1993), pp. 84–100.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 994–1005, October 1993

RANDOM-SELF-REDUCIBILITY OF COMPLETE SETS*

JOAN FEIGENBAUM† AND LANCE FORTNOW‡

Abstract. This paper generalizes the previous formal definitions of random-self-reducibility. It is shown that, even under a very general definition, sets that are complete for any level of the polynomial hierarchy are not nonadaptively random-self-reducible, unless the hierarchy collapses. In particular, NP-complete sets are not nonadaptively random-self-reducible, unless the hierarchy collapses at the third level. By contrast, we show that sets complete for the classes PP and $\mathrm{MOD}_m\mathrm{P}$ are random-self-reducible.

Key words. random-self-reductions, complexity classes, interactive proof systems, program checkers

AMS subject classifications. 68Q05, 68Q15

1. Introduction. Informally, a function f is *random-self-reducible* if the evaluation of f at any given instance x can be reduced in polynomial time to the evaluation of f at one or more *random* instances y_i.

Random-self-reducible functions have many applications, including:

Average-case complexity. A random-self-reduction maps an arbitrary, worst-case instance x in the domain of f to a set of random instances $y_1, \ldots, y_k$ in such a way that $f(x)$ can be computed in polynomial time, given x, $f(y_1), \ldots, f(y_k)$, and the coin-toss sequence used in the mapping. Thus the average-case complexity of f, where the average is taken with respect to the induced distribution on instances y_i, is the same, up to polynomial factors, as the worst-case randomized complexity of f. An important special case is that in which each random instance y_i is uniformly distributed over all elements in $Dom(f)$ that have length $|x|$. In this case f is as hard on average as it is in the worst case. For example, it follows from a result in [21] that the permanent of integer matrices (PERM) function is random-self-reducible. The PERM function is also #P-complete (cf. [29]); thus if PERM could be computed efficiently *on average* (with respect to the target distribution of the reduction), then *every* function in #P could, with a randomized algorithm, be computed efficiently in the *worst case*. Furthermore, the random-self-reduction for PERM is very simple, whereas standard average-case hardness proofs are often complicated.

Lower bounds. The random-self-reducibility of the parity function is used in [3] to obtain a simple proof that a random oracle separates the polynomial hierarchy (PH) from PSPACE. (An earlier proof of this result in [14] does not use random-self-reducibility.)

Interactive proof systems and program checkers. Random-self-reductions are crucial ingredients in many of the original examples of interactive proof systems and program checkers (cf. [11], [18]). Intuitively, this occurs because the verifier/checker interrogates the prover/program by comparing its output on the specific input of interest to its outputs on other correlated random instances. Several variations of this relationship between random-self-reducibility and proof systems/checkers are stated formally in [12], [22], [28]. These ideas play a crucial role in the characterization of the language-recognition power of interactive

*Received by the editors January 29, 1991; accepted for publication (in revised form) July 7, 1992. This work was supported in part by National Science Foundation grant CCR 90-09936. This is a revised version of the paper "On the Random-Self-Reducibility of Complete Sets," appearing in Proceedings of the 6th Annual Structure in Complexity Theory Conference, June 30–July 3, 1991, Chicago, Illinois. ©1991 by IEEE Computer Society.

†AT&T Bell Laboratories, Room 2C473, 600 Mountain Avenue, P.O. Box 636, Murray Hill, New Jersey 07974-0636 (`jf@research.att.com`).

‡Department of Computer Science, University of Chicago, 1100 East 58th Street, Chicago, Illinois 60637 (`fortnow@cs.uchicago.edu`).

proof systems (cf. [5], [22], [25]). Currently, one of the most important open questions about checkability is whether NP-complete sets are checkable. The main result that we present in §3 implies that if NP-complete sets are checkable, their checkers must use radically different techniques from those used by the existing checkers.

Cryptographic protocols. The fact that certain number-theoretic functions are random-self-reducible (and hence hard on average if they are hard at all) is used extensively in the theory of cryptography, e.g., to achieve *probabilistic encryption* (cf. [17]) and *cryptographically strong pseudorandom number generation* (cf. [13]). Random-self-reductions also provide natural examples of *instance-hiding schemes* (cf. [1], [7], [8]), in which a weak, private computing device uses the resources of a powerful, shared computing device without revealing its private data.

Although random-self-reducibility had been used for a long time in the design and analysis of cryptographic protocols (cf., e.g., [17], [13]), it was first defined formally and studied from a complexity-theoretic point of view by Abadi, Feigenbaum, and Kilian [1]; they considered reductions that map the given instance x to *one* random instance y. It is a corollary of the main result in [1] that no NP-hard function is random-self-reducible in this sense unless the polynomial hierarchy collapses at the third level.

Random-self-reductions that produce several correlated random instances $y_1, \dots, y_k$ were defined formally by Feigenbaum, Kannan, and Nisan [15]; however, they considered only reductions that produce y_i's that are uniformly distributed over $\{0, 1\}^{|x|}$. Their main result is that self-reductions that map x to two instances y_1 and y_2, each of which is uniformly distributed over $\{0, 1\}^{|x|}$, do not exist for NP-hard functions unless the polynomial hierarchy collapses at the third level.

The related idea of mapping an instance x in the domain of f to one or more random instances $y_1, \dots, y_k$ in the domain of a different function g is studied in [1], [7], [8]. For the case of one random y, a negative result for NP-hard functions is obtained in [1]. If multiple random y_i's are allowed, then *every* function f can be *locally randomly reduced* to a related function g; see [7], [8] for a thorough discussion.

In this paper we continue the study of random-self-reductions from a complexity-theoretic point of view. We further generalize the formal definition of random-self-reducibility that is studied in [15]. Specifically, we look at reductions that map a given instance x to a sequence of random instances $y_1, \dots, y_k$, with the property that the induced distribution on each y_i depends only on the length of x. We consider both *nonadaptive k-random-self-reductions*, in which the k random instances are produced in one pass, and *adaptive k-random-self-reductions*, in which the instance y_i may depend not only on x and the coin-toss sequence used in the reduction but also on $f(y_1), \dots, f(y_{i-1})$. Our main results are as follows:

• If S is complete for Σ_i^p for any $i \geq 1$ and if χ_S is nonadaptively $k(n)$-random-self-reducible, for any polynomially bounded function k, then the polynomial hierarchy collapses at the $(i+2)$nd level. In particular, if the characteristic function for any NP-complete set has a nonadaptive random-self-reduction, then the polynomial hierarchy collapses at the third level. This strengthens the main result in [15].

• If S is complete for PP or for $\mathrm{MOD}_m\mathrm{P}$ for any $m > 1$, then χ_S is adaptively $k(n)$-random-self-reducible for some polynomially bounded function k. Setting $m = 2$, we get that $\oplus$P-complete sets are random-self-reducible.

The rest of this paper is organized as follows. In §2, we define our terms precisely and recall known results that we will use. Section 3 contains our main negative result about complete sets in the polynomial hierarchy. Section 4 contains the proofs that complete sets for PP and $\mathrm{MOD}_m\mathrm{P}$ are random-self-reducible. Open problems are stated in §5.

2. Preliminaries.

Throughout this paper, f is a function from $\{0, 1\}^*$ to $\{0, 1\}^*$, and x is an arbitrary input for which we would like to determine $f(x)$. We use r to denote a sequence of fair coin tosses; if $|x| = n$, then $|r| = w(n)$, where w is a polynomially bounded function of n. The number of random queries produced by a reduction, denoted $k(n)$, is also a polynomially bounded function of n.

DEFINITION 2.1. A function f is *nonadaptively $k(n)$-random-self-reducible* (abbreviated "nonadaptively k-rsr") if there are polynomial-time computable functions σ and ϕ with the following properties.

(1) For all n and all $x \in \{0, 1\}^n$

$$f(x) = \phi(x, r, f(\sigma(1, x, r)), \ldots, f(\sigma(k, x, r)))$$

for at least 3/4 of all r's in $\{0, 1\}^{w(n)}$.

(2) For all n, all $\{x_1, x_2\} \subset \{0, 1\}^n$, and all i, $1 \le i \le k$, if r is chosen uniformly at random, then $\sigma(i, x_1, r)$ and $\sigma(i, x_2, r)$ are identically distributed.

Feigenbaum, Kannan, and Nisan [15] use the term *$k(n)$-random-self-reduction* to describe a special case of Definition 2.1, i.e., the case in which each of the random variables $\sigma(i, x, r)$ is distributed uniformly over $\{0, 1\}^n$.

Next we generalize Definition 2.1 to allow a multiround, adaptive strategy for choosing random queries.

DEFINITION 2.2. The function f is *adaptively $k(n)$-random-self-reducible* (abbreviated "adaptively k-rsr") if there is a probabilistic, polynomial-time oracle machine ϕ that, on input x of length n, produces $k(n)$ *rounds* of f-oracle queries. The query $y_i(x, r)$ produced in round i may depend on all queries and answers in rounds 1 through $i - 1$.

The reduction ϕ must have the following properties:

(1) For all x it outputs the correct answer $f(x)$ for at least 3/4 of all $r \in \{0, 1\}^{w(n)}$.

(2) For all n and all i, $1 \le i \le k$, if $|x_1| = |x_2| = n$ and r is chosen uniformly from $\{0, 1\}^{w(n)}$, then the random variables $y_i(x_1, r)$ and $y_i(x_2, r)$ are identically distributed. Note that condition (2) is not required to hold for $y_i(x, r)$ if wrong answers are given in earlier rounds.

We say that a function f is *poly-rsr* (or simply *rsr*) if there is some polynomially bounded function k such that f is either nonadaptively or adaptively k-rsr. The reductions themselves are also referred to as poly-rsr's or rsr's. A set S is poly-rsr if its characteristic function χ_S is poly-rsr.

Locally random reductions (lrr's) are a generalization of nonadaptive random-self-reductions. In a (t, k)-lrr from f to g, an instance x in the domain of f is mapped (nonadaptively) to k instances $y_1, \ldots, y_k$ in the domain of a different function g. For any $\{i_1, \ldots, i_t\} \subseteq \{1, \ldots, k\}$ the distribution induced on target queries $y_{i_1}, \ldots, y_{i_t}$ is the same for input instances x_1 and x_2 if $|x_1| = |x_2|$. Thus a nonadaptive k-rsr for f is a $(1, k)$-lrr from f to f. *Instance-hiding schemes* (ihs's) provide a further generalization of this notion. In a t-private, k-oracle ihs for f, the querier may use a multiround adaptive strategy to query k physically separated, arbitrarily powerful oracles; the oracles may also use an adaptive strategy and may flip coins. The *view* of any set of at most t of the oracles (i.e., the transcript of queries and answers together with the coin flips of the oracles) depends only on the length of the input instance. One-oracle ihs's were studied by Abadi, Feigenbaum, and Kilian [1], who showed that NP-hard functions do not have one-oracle ihs's unless the polynomial hierarchy collapses at the third level. The question of whether multioracle ihs's exist was posed by Rivest [1] and was answered by Beaver and Feigenbaum [7]. Every function f has a 1-private, $(n + 1)$-oracle ihs. In fact, the general ihs construction of [7] uses only one round of queries and does not require the oracles to flip coins; so, in current terminology, it is a $(1, n + 1)$-lrr. The term lrr was subsequently introduced and formally defined by Beaver, Feigenbaum, Kilian, and Rogaway [8], who also gave an improvement of the Beaver–Feigenbaum construction: For every polynomially bounded $t = t(n)$ and every function f there is a function g such that f is $(t, (tn/\log n) + 1)$-lrr to g.

The gist of Definitions 2.1 and 2.2 is that, for any fixed value of i, the distribution of random queries to the ith oracle depends only on the length n of the input x. In keeping with the terminology in [1], [7], we say that an rsr "leaks at most n to each oracle." In cryptographic applications it is often natural to consider reductions that leak at most some other function L; Definitions 2.1 and 2.2 have natural generalizations that fit these applications; see [1], [7] for details.

In several proofs we will use the following Chernoff bounds on the binomial distribution, which are taken directly from [26, Lecture 4, p. 29].

FACT 2.3. *Let $Y_1, \dots, Y_n$ be independent, with* $\Pr(Y_j = 1) = p_j$ *and* $\Pr(Y_j = 0) = 1 - p_j$, *and normalize by setting $X_j = Y_j - p_j$. Set $p = (p_1 + \dots + p_n)/n$ and $X = X_1 + \dots + X_n$. Then*

$$\Pr(X > a) < e^{-2a^2/n} \tag{1}$$

and

$$\Pr(X < -a) < e^{-a^2/2pn}. \tag{2}$$

LEMMA 2.4. *If a function f is nonadaptively (respectively, adaptively) $k(n)$-rsr, then f is nonadaptively (respectively, adaptively) $24t(n)k(n)$-rsr, where condition (1) of Definitions 2.1 and 2.2 holds for at least $1 - 2^{-t(n)}$ of the r's in $\{0, 1\}^{24t(n)w(n)}$.*

Proof. Let $r = r_1 \dots r_{24t(n)}$, with each $r_i \in \{0, 1\}^{w(n)}$. Define $\sigma_{ij} = \sigma(i, x, r_j)$ for $1 \le i \le k$ and $1 \le j \le 24t(n)$. Let $\phi_j = \phi(x, r_j, f(\sigma_{1j}), \dots, f(\sigma_{kj}))$. Let the new ϕ choose the plurality of the ϕ_j's, handling ties arbitrarily. Now apply inequality (2) from Fact 2.3 with $p = 3/4$, $n = 24t$, $a = -n/4$, and $Y_j = 1$ if and only if $\phi_j = f(x)$. □

We now recall some definitions and known results that will be used in §§3 and 4.

Let $f : \{0, 1\}^* \to \{0, 1\}$ be an arbitrary Boolean function, and let $f_n : \{0, 1\}^n \to \{0, 1\}$ be the restriction of f to inputs $(x_1, \dots, x_n) \in \{0, 1\}^n$. For any finite field K_n there is a unique multilinear polynomial $g_n \in K_n[X_1, \dots, X_n]$ that *represents* f_n over K_n, i.e., g_n agrees with f_n on all inputs $(x_1, \dots, x_n)$ in $\{0, 1\}^n$. In the context of random-self-reducibility we always take K_n to be a field of size at least $n + 1$. The polynomial g_n has a standard explicit formula; we give the formula here and discuss some computational aspects of it in §4 below. Let $x = (x_1, \dots, x_n)$ be an arbitrary element of K_n^n, and let $y = (y_1, \dots, y_n)$ be an arbitrary element of $\{0, 1\}^n$. Then

$$g_n(x) = \sum_{y \in \{0,1\}^n} \delta_y(x) f_n(y), \tag{3}$$

and

$$\delta_y(x) = \prod_{i=1}^{n} (x_i - (1 - y_i))(-1)^{(1-y_i)}. \tag{4}$$

For $x \in \{0, 1\}^n$ the monomial $\delta_y(x)$ is 1 if $y = x$ and is 0 otherwise. We call g_n the *arithmetization of f_n over K_n*, and we call $g = \{g_n\}_{n \ge 1}$ the *arithmetization of f over $\{K_n\}_{n \ge 1}$*.

FACT 2.5 (the low-degree polynomial trick). *If $g_n \in K_n[X_1, \dots, X_n]$ has degree d_n and $|K_n| > d_n$, then $g = \{g_n\}_{n \ge 1}$ is nonadaptively $(d_n + 1)$-rsr. In particular, if g_n is the arithmetization over K_n of a Boolean function f_n, then g is nonadaptively $(n + 1)$-rsr.*

Proof. Let $\alpha_1, \ldots, \alpha_{d_n+1}$ be distinct elements of K_n. Choose coefficients $c_1, \ldots, c_n$ independently and uniformly at random from K_n, and let $\sigma(i, (x_1, \ldots, x_n), r) = (c_1\alpha_i + x_1, \ldots, c_n\alpha_i + x_n)$ for $1 \le i \le d_n + 1$. Let

$$G(Z) = g_n(c_1 Z + x_1, \ldots, c_n Z + x_n).$$

Then G is a one-variable polynomial of degree at most d_n that satisfies

$$G(0) = g_n(x_1, \ldots, x_n).$$

The function ϕ of the rsr interpolates the $d_n + 1$ values $(\alpha_1, G(\alpha_1))$, $(\alpha_2, G(\alpha_2))$, $\ldots$, $(\alpha_{d_n+1}, G(\alpha_{d_n+1}))$ to recover the polynomial G and outputs the constant term. □

The random-self-reducibility of multivariate polynomials is the key to some of the results stated above. Beaver and Feigenbaum's general construction of multioracle ihs's [7], which they described in terms of arithmetic circuits, can be described in current terminology as follows: Every Boolean function is $(1, n+1)$-lrr to its arithmetization over $\{K_n\}_{n\ge1}$, where K_n is any finite field of size greater than n. Lipton [21] later used the same construction to show that multivariate polynomials are, in his terms, *randomly testable*; his was the first paper to state the construction in terms of polynomials instead of arithmetic circuits. In current terminology Lipton's observation is that multivariate polynomials are nonadaptively rsr, provided the degree is polynomially bounded in the number of variables. Lipton also pointed out that the function that computes the permanent of a matrix over a finite field is a low-degree multivariate polynomial and is thus randomly testable. Finally, Beaver, Feigenbaum, Kilian, and Rogaway [8] showed how to represent f_n as a degree-$(n/\log n)$ polynomial h_n over K_n by performing a simple change of variables, thus obtaining a $(1, (n/\log n) + 1)$-lrr from f to h.

DEFINITION 2.6. A complexity class C is *#P-robust* if $\mathrm{FP}^{\mathrm{C}} = \#\mathrm{P}^{\mathrm{C}}$, where FP denotes the class of all polynomial-time computable functions.

In §4, we will use the following generalized version of #P.

DEFINITION 2.7 (cf. [16]). A function $f : \{0,1\}^* \to \mathcal{Z}$ is in the complexity class Gap-P if there is an NP machine M such that, for all x, $f(x)$ is the difference between the number of accepting computations of M on input x and the number of rejecting computations of M on input x. Equivalently, a function f is in Gap-P if it is the difference of two #P functions.

By analogy with Definition 2.6, we have the following definition.

DEFINITION 2.8. A complexity class C is *Gap-P-robust* if $\mathrm{FP}^C = \text{Gap-P}^C$.

FACT 2.9. *A complexity class C is* Gap-P-*robust if and only if it is* #P-*robust.*

Let $\langle\cdot,\cdot\rangle$ be a one-to-one, onto, polynomial-time-computable, polynomial-time invertible pairing function from $\{0,1\}^* \times \{0,1\}^*$ to $\{0,1\}^*$. Gap-P has the following closure properties.

FACT 2.10 (cf. [16]). *If a function* $f(\langle x, y\rangle) \in$ Gap-P, *then the following functions are also in* Gap-P *for any polynomial* p:

1. $g(\langle x, y\rangle) = -f(\langle x, y\rangle)$;
2. $g(x) = \sum_{|y|\le p(|x|)} f(\langle x, y\rangle)$;
3. $g(x) = \prod_{1\le y\le p(|x|)} f(\langle x, y\rangle)$.

In particular, Gap-P is closed under subtraction.

DEFINITION 2.11. Let m be a positive integer greater than 1. A set S is in $\mathrm{MOD}_m\mathrm{P}$ if there is an NP machine M with the following property: If $x \in S$, then the number of accepting computations of M on input x is not equal to 0 mod m; if $x \notin S$, then the number of accepting computations of M on input x is equal to 0 mod m.

Thus the class ⊕P, defined in [23], is $\mathrm{MOD}_2\mathrm{P}$ in the notation used here.

FACT 2.12 (cf. [9], [20]). *If m_1 and m_2 are relatively prime, then* $S \in \mathrm{MOD}_{m_1m_2}\mathrm{P}$ *if and only if there are sets* $S_1 \in \mathrm{MOD}_{m_1}\mathrm{P}$ *and* $S_2 \in \mathrm{MOD}_{m_2}\mathrm{P}$ *such that* $S = S_1 \cup S_2$.

FACT 2.13 (cf. [9], [20]). *If $p_1^{e_1} \cdots p_t^{e_t}$ is the prime factorization of m, then* $\mathrm{MOD}_m\mathrm{P} = \mathrm{MOD}_{p_1 \cdots p_t}\mathrm{P}$.

We use the following class of straight-line programs of multivariate polynomials over $\mathcal{Z}$ to prove that sets complete for $\mathrm{MOD}_m\mathrm{P}$ are rsr.

DEFINITION 2.14 (cf. [4]). A *positive retarded arithmetic program with binary substitutions* (PRAB) is a sequence $P = \{p_1, \ldots, p_s\}$ of instructions such that, for every k, one of the following holds:

(1) p_k is one of the constant polynomials 0 or 1;

(2) $p_k = x_i$ for some $i \leq k$;

(3) $p_k = 1 - x_i$ for some $i \leq k$;

(4) $p_k = p_i + p_j$ for some $i, j < k$;

(5) $p_k = p_i p_j$ for some $i + j \leq k$;

(6) $p_k = p_j(x_i = 0)$ or $p_j(x_i = 1)$ for some $i, j < k$. Here $p_j(x_i = \epsilon)$ refers to the polynomial obtained from p_j by replacing the variable x_i by the value ϵ.

We say that the program P *computes the polynomial* p_s.

DEFINITION 2.15 (cf. [4]). A sequence $P_1, P_2, \ldots$ of PRABs is *uniform* if there is a deterministic polynomial-time machine that, on input 1^n, outputs the instruction sequence P_n.

FACT 2.16 (cf. [4]). *A set S is in* $\mathrm{MOD}_m\mathrm{P}$ *if and only if there is a uniform sequence* $\{P_n\}_{n \geq 1}$ *of* PRAB*s such that for every* $x \in \{0, 1\}^*$

$$\chi_S(x) \equiv P_{|x|}(x) \bmod m.$$

We use AM^{poly} to denote the class of sets accepted by bounded-round Arthur–Merlin games (cf. [6]) in which Arthur is given polynomial-length advice in addition to probabilistic polynomial time. Note that this class is not necessarily the same as $\mathrm{AM}/poly$, because AM^{poly} requires proper probabilities of acceptance only when the advice is correct. Because the main results of [6], [19] relativize, we have:

FACT 2.17. $\mathrm{AM}^{poly} = \mathrm{NP}/poly$.

Finally, we use the following known relationship between levels of the polynomial hierarchy and the corresponding nonuniform classes.

FACT 2.18 (cf. [30]). *If $\Sigma_i^P \subseteq \Pi_i^P/poly$, then the polynomial hierarchy collapses to Σ_{i+2}^P. This fact relativizes: For any O, if $\Sigma_i^{P,O} \subseteq \Pi_i^{P,O}/poly$, then* $\mathrm{PH}^O \subseteq \Sigma_{i+2}^{P,O}$.

3. Complete sets in the polynomial hierarchy.

THEOREM 3.1. *If S is in* NP *and is nonadaptively poly-rsr, then $\overline{S}$ is in* AM^{poly}.

Proof. Let σ, ϕ be a nonadaptive k-rsr for S, where $k = k(n)$ is a polynomially bounded function. By Lemma 2.4, we can assume ϕ gives an incorrect value for the characteristic function of S with probability at most 2^{-n}.

Consider instances x of length n. The verifier's advice is the k-tuple $(p_1, \ldots, p_k)$, where p_i is the probability that a target instance $\sigma(i, x, r)$ is in S. The probability is computed over all coin-toss sequences r.

We denote by $Trans(x, r)$ the *transcript* of the reduction σ, ϕ on input x and random string r. That is, if $y_i = \sigma(i, x, r)$ and $b_i = \chi_S(y_i)$, then $Trans(x, r) = (y_1, b_1, \ldots, y_k, b_k)$. Fix a specific NP machine M that accepts S. Let $ATrans(x, r)$, an *augmented transcript*, be $(y_1, b_1, w_1, \ldots, y_k, b_k, w_k)$, where y_i and b_i are as before, $w_i =$ NIL if $b_i = 0$, and w_i is a witness, with respect to M, that $y_i \in S$ if $b_i = 1$.

The following is an AM^{poly} protocol for $\overline{S}$. Let $m = 9k^3$.

Interactive proof system for $\overline{\mathbf{S}}$. The quantifiers "for all $1 \leq i \leq k$" and "for all $1 \leq j \leq m$" are implicit whenever the subscripts i and j are used. For each $j \neq j'$, r_j is independent of $r_{j'}$.

V: Choose r_j.
$V \to P$: $\{r_j\}$.
$P \to V$: A claimed value $(y_{1,j}, b_{1,j}, w_{1,j}, \ldots, y_{k,j}, b_{k,j}, w_{k,j})$ for $ATrans(x, r_j)$.
V: Accept if and only if
(1) $\phi(x, r_j, b_{1,j}, \ldots, b_{k,j}) = 0$,
(2) More than $p_i m - 2\sqrt{km}$ of the $y_{i,j}$'s are in S according to P, and
(3) If $w_{i,j} \neq$ NIL, then it is a correct witness that $y_{i,j} \in S$.

Suppose that x is not in S and P is honest. Then acceptance condition (1) is met with probability at least $1 - m/2^n > 11/12$ for all $n > \log 12m$. Condition (3) is, of course, always met if P is honest. We need to show only that condition (2) is met with probability at least 3/4 to have all three conditions met with probability at least 2/3. Let $Z_{i,j}$ be an indicator variable that is 1 if $y_{i,j}$ is in S and 0 otherwise, and let $Z_i = \sum_{j=1}^{m} Z_{i,j}$. Acceptance condition (2) is met if $Z_i > p_i m - 2\sqrt{km}$ for all i. Because $r_1, \ldots, r_m$ are pairwise independent, so are $Z_{i,1}, \ldots, Z_{i,m}$. Clearly, $E(Z_i) = p_i m$ and $Var(Z_i) = p_i(1 - p_i)m < m$. So Chebyshev's inequality suffices to show that

$$\begin{aligned}\Pr(Z_i \leq p_i m - 2\sqrt{km}) &\leq \Pr(|Z_i - p_i m| \geq 2\sqrt{km}) \\ &= \Pr(|Z_i - E(Z_i)| \geq 2\sqrt{km}) \\ &\leq \frac{\mathrm{Var}(Z_i)}{4km} < \frac{1}{4k}\end{aligned}$$

for each i. Thus the probability that at least one Z_i is too small (i.e., the probability that condition (2) is not met) is at most 1/4.

Now suppose that x is in S. We wish to show that the probability that V accepts is at most 1/3. If V accepts, condition (1) is satisfied, and so either

(a) Given correct answers $b_{i,j}$, ϕ says x is not in S for some j, or

(b) P^* must have lied about $b_{i,j}$ for at least one $y_{i,j}$ for each j.

(The optimal cheating prover would never violate condition (3).) Event (a) can happen only with probability at most $m/2^n < 1/12$ for all $n > \log 12m$. Thus we need only show that event (b) can happen with probability at most 1/4.

If P^* tells a total of m lies, there must be an i for which P^* claims that at least m/k of the $y_{i,j}$'s that are in S are not in S. It suffices to show that, for each i, P^* can do this with probability at most $1/4k$ and still satisfy acceptance condition (2). The probability that P^* can tell m/k lies and still claim that more than $p_i m - 2\sqrt{km}$ of the $y_{i,j}$'s are in S is just the probability that more than $p_i m + m/k - 2\sqrt{km}$ of the $y_{i,j}$'s are in S. This probability is at most $\exp^{(-2(m/k^2 - 4\sqrt{m/k} + 4k))} = e^{-2k} < 1/4k$ for $m = 9k^3$ and all positive integers k. We obtain this bound by using inequality (1) from Fact 2.3, with $a = m/k - 2\sqrt{km}$ and $n = m$. □

The technique of showing that a particular type of random-self-reduction for S implies a type of interactive proof system for $\overline{S}$ was first used by Feigenbaum, Kannan, and Nisan [15, Thm. 4.4]. There it is shown that if S has what they call a "one-sided 1-rsr," then $\overline{S} \in \mathrm{AM}^{poly}$. These reductions are much more restricted than the type of rsr's considered here; a precise definition can be found in [15].

In fact, the following stronger statement can be made. This observation is due to Szegedy.

COROLLARY 3.2. *If S is in* NP *and is nonadaptively poly-rsr, then $\overline{S}$ is in* $\mathrm{AM}^{\log}$.

Proof. Let σ, ϕ be a nonadaptive rsr for S. Define a new nonadaptive rsr σ', ϕ' as follows. On input x and random string r', first choose a uniformly random permutation π on $\{1, \ldots, k\}$. Let r be the unused portion of r'. For $1 \leq i \leq k$, let $\sigma'(i, x, r') \equiv \sigma(\pi(i), x, r)$. Let $\phi'(x, r', b_1, \ldots, b_k) \equiv \phi(x, r, b_{\pi^{-1}(1)}, \ldots, b_{\pi^{-1}(k)})$. Now all of the random variables

$\sigma'(1, x, r'), \ldots, \sigma'(k, x, r')$ are identically distributed. The proof system for $\overline{S}$ is essentially the same as the one given in the proof of Theorem 3.1. The probabilities $p_1, \ldots, p_k$ are all equal because $\sigma'(1, x, r'), \ldots, \sigma'(k, x, r')$ are identically distributed. Let $p = p_1 = \cdots = p_k$. The integer $\lceil pm - 2\sqrt{km} \rceil$ is sufficient advice for the verifier on inputs of length n, and it can be written down in $O(\log n)$ bits. $\square$

COROLLARY 3.3. *If any* NP-*complete set is nonadaptively poly-rsr, then the polynomial hierarchy collapses at the third level.*

Proof. This follows directly from Theorem 3.1 and Facts 2.17 and 2.18. $\square$

COROLLARY 3.4. *If S is complete for Σ_i^p or Π_i^p, $i \geq 1$, and S is nonadaptively poly-rsr, then the polynomial hierarchy collapses at the* $(i+2)$nd *level.*

Proof. The proofs of Theorem 3.1, Fact 2.18 and thus Corollary 3.3 relativize. If we relativize them with respect to an oracle O such that O is Σ_{i-1}^p-complete, we get Corollary 3.4. $\square$

We end this section with a partial negative result about adaptive rsr's for NP-complete sets.

THEOREM 3.5. *If S is* NP-*complete and is adaptively $O(\log n)$-rsr, then the polynomial hierarchy collapses at the third level.*

Proof (sketch). We give the structure of the proof in some detail but omit the probability calculations. All of the calculations involve Chernoff bounds and are analogous to the ones used in the proof of Theorem 3.1.

As in the nonadaptive case, we will show that the hypothesis implies that $\overline{S} \in \mathrm{AM}^{poly}$. Suppose that S is adaptively k-rsr, where $k(n) = O(\log n)$. In the augmented transcript $(y_1, b_1, w_1, \ldots, y_k, b_k, w_k)$ of an adaptive reduction, y_i denotes the oracle query asked in the ith round.

The AM^{poly} proof system for $\overline{S}$ is almost identical to the one in Theorem 3.1. The only differences are that we may have to use a different polynomial for m and that the verifier, in acceptance condition (3), must also check whether $y_{i,j}$ is indeed the query that the reduction would produce in round i if the input string is x, the random string is r_j, and the answers in rounds 1 through $i-1$ are $b_{1,j}$ through $b_{i-1,j}$. However, it is trickier to show that the proof system is correct in the adaptive case. The problem arises when x is in S and P^* must lie about at least one $b_{i,j}$ for each j. In the nonadaptive case the only way that P^* can lie is to claim that $y_{i,j}$ is not in S when it really is. In the adaptive case this is not necessarily true.

Consider a $k \times m$ matrix whose (i, j)th entry is P^*'s claimed value for $b_{i,j}$. In order to convince V to accept an input x that is really in S, P^* must claim that the number of 1's in row i is greater than $p_i m - 2\sqrt{km}$, and P^* must *spoil* every column, i.e., P^* must lie about at least one $b_{i,j}$ for each j. The problem is that there is a trade-off between these two requirements that could work to P^*'s advantage. Suppose that $b_{i_0,j}$ is the first lie that P^* tells in column j. (If this is so, we say that "column j is spoiled in row i_0.") For this first lie, it must be the case that $y_{i,j}$ is in S but P^* claims it is not. However, this incorrect $b_{i_0,j}$ is used in the subsequent computation of queries $y_{i,j}$; this may produce $y_{i,j}$'s, $i > i_0$, that are in S where the correct value of $b_{i_0,j}$ would have produced $y_{i,j}$'s that are not in S.

For $k = O(\log n)$, we can still choose $m = poly(n)$ so that the proof system works. We make the following worst-case assumption in our argument: If column j is spoiled in row i_0, then the rest of the column, i.e., all $y_{i,j}$ with $i > i_0$, consists entirely of elements of S. We now estimate now many columns can be spoiled in each row.

The expected number of $y_{1,j}$'s that are in S is $p_1 m$; so, with very high probability, for these particular choices $r_1, \ldots, r_m$, the actual number is less than $p_1 m + 2\sqrt{km}$. P^* must claim that more than $p_1 m - 2\sqrt{km}$ are in S; so, with high probability, P^* may spoil at most $4\sqrt{km}$ columns in row 1 without getting caught.

What happens in row 2? With high probability, the actual number of $y_{2,j}$'s in S is less than $p_2m + 2\sqrt{km}$. We assume that the columns spoiled in row 1 contribute $4\sqrt{km}$ additional $y_{2,j}$'s in S. P^* must claim that more than $p_2m - 2\sqrt{km}$ are in S; so, with high probability, P^* may spoil at most $8\sqrt{km}$ new columns in row 2.

In row 3, the actual number of $y_{3,j}$'s in S is less than $p_3m + 2\sqrt{km}$, with high probability. Columns spoiled in rows 1 and 2 contribute at most $12\sqrt{km}$ additional $y_{3,j}$'s in S. Thus P^* may, with high probability, spoil at most $16\sqrt{km}$ new columns in row 3 and still claim that $p_3m - 2\sqrt{km}$ of the $y_{3,j}$'s are in S.

Continuing in this manner, we see that, with high probability, at most $2^{k+2}\sqrt{km}$ columns are spoiled in all k rows. For $k \leq c \log n$, we can choose $m = n^{4c}$, say, and prevent P^* from spoiling all of the columns.

In fact, the conclusion that $\overline{S}$ is in AM^{poly} follows from the weaker assumption that S has an adaptive rsr with $O(\log n)$ rounds of queries and polynomially many queries in each round. Unfortunately, this proof technique does not work unless the number of rounds is $O(\log n)$. □

4. Complete sets above the polynomial hierarchy. We first recall the following known positive result.

THEOREM 4.1. *If f is* #P*-complete, then f is poly-rsr.*

Proof. This follows easily from the results on low-degree polynomials discussed in §2. Let PERM be the #P-complete function that computes permanents of integer matrices. An instance x of PERM can be reduced to the computation of $\mathrm{PERM}(x) \bmod p_i$, for some small collection of primes p_i—to recover $\mathrm{PERM}(x)$ from $\{\mathrm{PERM}(x) \bmod p_i\}$, use the Chinese Remainder Theorem. For each p_i, $\mathrm{PERM}(x) \bmod p_i$ is just a low-degree polynomial over a finite field. Thus, by Fact 2.5, it can be reduced to the evaluation of a small collection of random instances $\{\mathrm{PERM}(y_{ij}) \bmod p_i\}$. These y_{ij}'s can be regarded as random instances of PERM—from the value of $\mathrm{PERM}(y_{ij})$ over the integers, $\mathrm{PERM}(y_{ij}) \bmod p_i$ can be found simply by reducing mod p_i. In summary, the mapping from x to $\{y_{ij}\}$ is an rsr for PERM.

Let f be #P-complete. On input x of length n, the rsr for f proceeds as follows. Reduce x to one or more instances of PERM. Pad these instances if necessary so that their size depends only on n: For any $l \geq k$, a $k \times k$ matrix M can be "padded" out to an $l \times l$ matrix M' with the same permanent by letting $M'(i, j) = M(i, j)$ for $1 \leq i, j \leq k$, $M'(i, i) = 1$ for $k < i \leq l$, and $M'(i, j) = 0$ for all other values of i and j. Perform the above rsr of PERM. The random PERM-instances thus produced can be mapped back to f-instances because f is #P-complete. These f-instances leak at most n because the random PERM-instances leak at most n. □

We now proceed to our new positive results. The first one is a straightforward extension of Theorem 4.1.

COROLLARY 4.2. *If S is complete for* PP, *then S is poly-rsr.*

Proof. It is well known that the language classes P^{PP} and $\mathrm{P}^{\#\mathrm{P}}$ are equal. Thus S and PERM are polynomial-time equivalent. The rest of the proof is identical to that of Theorem 4.1. □

THEOREM 4.3. *If a complexity class* C *is* #P*-robust, then complete sets for* C *are poly-rsr.*

Proof. By Fact 2.9, it suffices to show that Gap-P-robustness of C implies that complete sets for C are poly-rsr. Suppose that C is Gap-P-robust, and let S be a complete set for C. For each $n \geq 1$, let p_n be a prime greater than n, let $f_n : \{0, 1\}^n \to \{0, 1\}$ be the characteristic function of S on strings of length n, and let $g = \{g_n\}_{n \geq 1}$ be the arithmetization of $f = \{f_n\}_{n \geq 1}$ over $\{\mathrm{GF}(p_n)\}_{n \geq 1}$. By Fact 2.10, we can compute g (by using equations (3) and (4)) in $\text{Gap-P}^{\mathrm{C}}$ and thus in FP^{C}. By Fact 2.5, g is (nonadaptively) $(n+1)$-rsr. On input x, the random-self-reduction of S proceeds as follows: Generate p_n; interpret the input instance x as an element of

Dom(g); apply the low-degree polynomial trick to get random instances $y_1, \ldots, y_{n+1}$; reduce the computation of $g(y_i)$ to membership queries about S, which can be done because $g \in \mathrm{FP}^{\mathrm{C}}$ and S is complete for C. The entire reduction leaks at most n because each of its components leaks at most n. □

COROLLARY 4.4. *Complete sets for* PSPACE *and* EXPTIME *are poly-rsr.*

Proof. This follows from the fact that PSPACE and EXPTIME are #P-robust. For example, let FPSPACE denote the set of functions computable in polynomial space. Then

$$\mathrm{FP}^{\mathrm{PSPACE}} \subseteq \#\mathrm{P}^{\mathrm{PSPACE}} \subseteq \mathrm{FPSPACE}^{\mathrm{PSPACE}} = \mathrm{FPSPACE}$$

and thus $\mathrm{FP}^{\mathrm{PSPACE}} = \#\mathrm{P}^{\mathrm{PSPACE}}$. □

Note that is unknown whether #P is itself #P-robust.

THEOREM 4.5. *If S is complete for* $\mathrm{MOD}_m\mathrm{P}$, *then S is poly-rsr. In particular, complete sets for $\oplus P$ are poly-rsr.*

Proof. By Facts 2.12 and 2.13, we can assume without loss of generality that m is prime. The reduction to the case of square-free m is trivial, by Fact 2.13. If $m = m_1 \cdots m_t$, where the m_i's are distinct primes, then Fact 2.12 tells us that $S = S_1 \cup \cdots \cup S_t$, where $S_i \in \mathrm{MOD}_{m_i}\mathrm{P}$. Thus, a query about membership of x in S reduces to the disjunction of queries about membership of x in $S_1, \ldots, S_t$. In what follows, we will show that S_i is rsr. Suppose that y is a random query produced by the rsr for S_i on input x. We must show how to compute $\chi_{S_i}(y)$ by making queries to an S oracle. First note that there is a set T_i in $\mathrm{MOD}_m\mathrm{P}$ such that $\chi_{S_i}(z) = \chi_{T_i}(z)$ for all z: If the underlying NP machine for S_i is M_i, then the underlying NP machine for T_i is M_i', where each computation path (accepting or rejecting) in M_i is replaced by m/m_i distinct paths in M_i'. Finally, T_i can be reduced to S because S is complete for $\mathrm{MOD}_m\mathrm{P}$.

Let S be a complete set in $\mathrm{MOD}_m\mathrm{P}$, where m is prime, and let $x = (x_1, \ldots, x_n)$ be an element of $\{0, 1\}^n$ for which we would like to determine membership in S. By Fact 2.16, there is a PRAB $P_n = \{p_1, \ldots, p_s\}$ for which $\chi_s(x) = P_n(x) \bmod m$. Furthermore, P_n can be generated in polynomial time, and it depends only on S and n (i.e., it leaks nothing about x except its length). We would like to apply Fact 2.5 (the low-degree polynomial trick) to p_s and then map the random p_s-instances back to S-oracle queries. However, there are only m distinct points in $\mathcal{Z}_m$, and the degree of p_s is a (polynomially bounded) function of n; thus, there will not be enough interpolation points to recover $p_s(x)$ this way.

We deal with this problem as it is dealt with in the proof that $\mathrm{MOD}_m\mathrm{P}$-complete sets are checkable (cf. [4]). For every positive integer k there is a unique finite field $\mathrm{GF}(m^k)$, and it is a vector space over $\mathcal{Z}_m$. Fix a basis for this vector space. This entails finding a polynomial of degree k that is irreducible over $\mathcal{Z}_m$, which can be done in probabilistic polynomial time [10], [24]. (In fact, we could choose k so that all that is required is a polynomial of degree l, where $\Omega(k/\log m) = l \leq k$, that is irreducible over $\mathcal{Z}_m$. Such a polynomial could be generated in deterministic polynomial time [2], but this is not necessary for our purpose, which is to use the polynomial in a reduction that is inherently probabilistic.) We can represent each element a of $\mathrm{GF}(m^k)$ as the $k \times k$ matrix M_a denoting the linear transformation $x \mapsto ax$ of $\mathrm{GF}(m^k)$ to itself. Then M_0 is the zero matrix, M_1 is the identity matrix, $M_{a+b} = M_a + M_b$, and $M_{ab} = M_{ba} = M_a M_b$.

Choose k so that $m^k > d = degree(p_s)$, and represent the elements computed by P_n as matrices over $\mathcal{Z}_m$. There is another PRAB, say, $\{p_1(1, 1), \ldots, p_1(k, k), \ldots, p_s(1, 1), \ldots, p_s(k, k)\}$, where $p_i(r, c)$ computes the element in row r, column c of $p_i(x_1, \ldots, x_n)$. Because $m^k > d$, we can apply Fact 2.5 to p_s in the following way: Let $\alpha_1, \ldots, \alpha_{d+1}$ be distinct elements of $\mathrm{GF}(m^k)$. Choose $c_1, \ldots, c_n$ independently and uniformly at random from the set of all $k \times k$ matrices over $\mathcal{Z}_m$ that encode elements of $\mathrm{GF}(m^k)$. Then $P_n(c_1 Z + x_1, \ldots, c_n Z + x_n)$ is a degree-d, one-variable polynomial with constant term $P_n(x) = \chi_s(x)$. So evaluate P_n at the $d+1$ uniformly distributed inputs $(c_1\alpha_1+x_1, \ldots, c_n\alpha_1+x_n), \ldots, (c_1\alpha_{d+1}+x_1, \ldots, c_n\alpha_{d+1}+x_n)$, and interpolate.

It remains to show that the computation of $P_n(c_1\alpha_j + x_1, \ldots, c_n\alpha_j + x_n)$ can be reduced in polynomial time to a sequence of S-oracle queries. In the matrix representation of P_n, each $p_i(r, c)$ is an instruction in a PRAB over $\mathcal{Z}_m$. Thus it is polynomial-time reducible to S by Fact 2.16, because S is complete for MOD$_m$P.

As in the previous proofs in this section, each S-oracle call leaks at most n because each random input $(c_1\alpha_j + x_1, \ldots, c_n\alpha_j + x_n)$ leaks at most n. $\square$

5. Open problems. Open problems abound and include the following:

• Do NP-complete sets have adaptive k-rsr's for some $k \gg \log n$?

• Are NP-complete sets checkable in the sense of [11]? Note that all known checkers for sets that are complete for natural complexity classes use rsr's.

• What other sets do or do not have rsr's? How about incomplete sets? Or sets and functions complete for classes C that satisfy PH $\subseteq$ BPP$^{\mathrm{C}}$ and P$^{\mathrm{C}}$ $\subseteq$ PSPACE? The classes MOD$_m$P, PP, and #P all fall between PH and PSPACE in this sense (cf. [27]).

Acknowledgments. We would like to thank Manuel Blum, Russell Impagliazzo, Steven Rudich, Gábor Tardos, and the referee for their comments on earlier versions of this paper.

REFERENCES

[1] M. Abadi, J. Feigenbaum, and J. Kilian, *On hiding information from an oracle*, J. Comput. System Sci., 39 (1989), pp. 21–50.

[2] L. Adleman and H. Lenstra, *Finding irreducible polynomials over finite fields*, in Proceedings of the 16th Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1986, pp. 350–355.

[3] L. Babai, *Random oracles separate* PSPACE *from the polynomial-time hierarchy*, Informat. Process. Lett., 26 (1987), pp. 51–53.

[4] L. Babai and L. Fortnow, *Arithmetization: A new method in structural complexity theory*, Comput. Complexity, 1 (1991), pp. 41–66.

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

[6] L. Babai and S. Moran, *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[7] D. Beaver and J. Feigenbaum, *Hiding instances in multioracle queries*, in Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Vol. 415, Springer, Berlin, 1990, pp. 37–48.

[8] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway, *Security with low communication overhead*, in Advances in Cryptology—Crypto '90, Lecture Notes in Computer Science, Vol. 537, Springer, Berlin, 1991, pp. 62–76.

[9] R. Beigel and J. Gill, *Counting classes: Thresholds, parity, mods, and fewness*, Theoret. Comput. Sci., 103 (1992), pp. 3–23.

[10] E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[11] M. Blum and S. Kannan, *Designing programs that check their work*, in Proceedings of the 21st Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1989, pp. 86–97.

[12] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting, with applications to numerical problems*, in Proceedings of the 22nd Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1990, pp. 73–83.

[13] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[14] J. Cai, *With probability one, a random oracle separates* PSPACE *from the polynomial-time hierarchy*, J. Comput. System Sci., 38 (1989), pp. 68–85.

[15] J. Feigenbaum, S. Kannan, and N. Nisan, *Lower bounds on random-self-reducibility*, in Proceedings of the 5th Structure in Complexity Theory Conference, IEEE Computer Society, Los Alamitos, CA, 1990, pp. 100–109.

[16] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, in Proceedings of the 6th Structure in Complexity Theory Conference, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 30–42.

[17] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[18] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof-systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[19] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, in Randomness and Computation, S. Micali, ed., Advances in Computing Research, Vol. 5, JAI Press, Greenwich, CT, 1989, pp. 73–90.

[20] U. Hertrampf, *Relations among* MOD-*classes*, Theoret. Comput. Sci., 74 (1990), pp. 325–328.

[21] R. Lipton, *New directions in testing*, in Distributed Computing and Cryptography, J. Feigenbaum and M. Merritt, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 2, American Mathematical Society, Providence, RI, 1991, pp. 191–202.

[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 859–868.

[23] C. Papadimitriou and S. Zachos, *Two remarks on the power of counting*, in Proceedings of the 6th GI Conference on Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 145, Springer, Berlin, 1983, pp. 269–276.

[24] M. Rabin, *Probabilistic algorithms in finite fields*, SIAM J. Comput., 9 (1980), pp. 273–280.

[25] A. Shamir, IP = PSPACE, J. Assoc. Comput. Mach., 39 (1992), pp. 869–877.

[26] J. Spencer, *Ten Lectures on the Probabilistic Method*, Conference Board of the Mathematical Sciences, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[27] S. Toda, PP *is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–877.

[28] M. Tompa and H. Woll, *Random self-reducibility and zero-knowledge interactive proofs of possession of information*, in Proceedings of the 28th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 472–482.

[29] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.

[30] C. Yap, *Some consequences of nonuniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 1006–1034, October 1993

LEARNING BINARY RELATIONS AND TOTAL ORDERS*

SALLY A. GOLDMAN[†], RONALD L. RIVEST[‡], AND ROBERT E. SCHAPIRE[§]

Abstract. The problem of learning a binary relation between two sets of objects or between a set and itself is studied. This paper represents a binary relation between a set of size n and a set of size m as an $n \times m$ matrix of bits whose (i, j) entry is 1 if and only if the relation holds between the corresponding elements of the two sets. Polynomial prediction algorithms are presented for learning binary relations in an extended on-line learning model, where the examples are drawn by the learner, by a helpful teacher, by an adversary, or according to a uniform probability distribution on the instance space.

The first part of this paper presents results for the case in which the matrix of the relation has at most k row types. It presents upper and lower bounds on the number of prediction mistakes any prediction algorithm makes when learning such a matrix under the extended on-line learning model. Furthermore, it describes a technique that simplifies the proof of expected mistake bounds against a randomly chosen query sequence.

In the second part of this paper the problem of learning a binary relation that is a total order on a set is considered. A general technique using a fully polynomial randomized approximation scheme (fpras) to implement a randomized version of the halving algorithm is described. This technique is applied to the problem of learning a total order, through the use of an fpras for counting the number of extensions of a partial order, to obtain a polynomial prediction algorithm that with high probability makes at most $n \lg n + (\lg e) \lg n$ mistakes when an adversary selects the query sequence. The case in which a teacher or the learner selects the query sequence is also considered

Key words. machine learning, computational learning theory, on-line learning, mistake-bounded learning, binary relations, total orders, fully polynomial randomized approximation schemes

AMS subject classifications. 68Q25, 68T05

1. Introduction.

In many domains it is important to acquire information about a relation between two sets. For example, one may wish to learn a "has-part" relation between a set of animals and a set of attributes. We are motivated by the problem of designing a prediction algorithm to learn such a binary relation when the learner has limited prior information about the predicate forming the relation. Although one could model such problems as concept learning, they are fundamentally different problems. In concept learning there is a single set of objects and the learner's task is to classify these objects, whereas in learning a binary relation there are two sets of objects and the learner's task is to learn the predicate that relates the two sets. Observe that the problem of learning a binary relation can be viewed as a concept learning problem if one lets the instances be all ordered pairs of objects from the two sets. However, the ways in which the problem may be structured are quite different when the true task is to learn a binary relation as opposed to a classification rule. That is, instead of a rule that defines which objects belong to the target concept, the predicate defines a relationship between pairs of object.

*Received by the editors December 1, 1991; accepted for publication (in revised form) June 9, 1992. Most of this research was carried out while all three authors were at MIT Laboratory for Computer Science with support provided by National Science Foundation grant DCR-8607494, U.S. Army Research Office grant DAAL03-86-K-0171, Defense Advanced Research Projects Agency contract N00014-89-J-1988, and a grant from the Siemens Corporation. Preliminary versions of this paper appeared in the Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, October 1989, and as Massachusetts Institute of Technology's Laboratory for Computer Science Technical Memo, MIT/LCS/TM-413, May 1990.

[†]Department of Computer Science, Washington University, St. Louis, Missouri 63130 (sg@cs.wustl.edu). This author currently receives support from G.E. Foundation Junior Faculty Grant and from National Science Foundation grant CCR-9110108.

[‡]MIT Laboratory for Computer Science, Cambridge, Massachusetts 02139 (rivest@theory.lcs.mit.edu).

[§]AT&T Bell Laboratories, Murray Hill, New Jersey 07974 (schapire@research.att.com). This author received additional support from U.S. Air Force Office of Scientific Research grant 89-0506 while at Harvard University.

A binary relation is defined between two sets of objects. Throughout this paper we assume that one set has cardinality n and the other has cardinality m. We also assume that for all possible pairings of objects the predicate relating the two sets of variables is either true (1) or false (0). Before defining a prediction algorithm, we first discuss our representation of a binary relation. Throughout this paper we represent the relation as an $n \times m$ binary matrix, where an entry contains the value of the predicate for the corresponding elements. Since the predicate is binary valued, all entries in this matrix are either 0 (false) or 1 (true). The *two-dimensional* structure arises from the fact that we are learning a binary relation.

For the sake of comparison we now briefly mention other possible representations. One could represent the relation as a table with two columns, where each entry in the first column is an item from the first set and each entry in the second column is an item from the second set. The rows of the table consist of the subset of the potential nm pairings for which the predicate is true. One could also represent the relation as a bipartite graph with n vertices in one vertex set and m vertices in the other set. An edge is placed between two vertices exactly when the predicate is true for corresponding items.

Having introduced our method for representing the problem, we now informally discuss the basic learning scenario. The learner is repeatedly given a pair of elements, one from each set, and is asked to predict the corresponding matrix entry. After making its prediction, the learner is told the correct value of the matrix entry. The learner wishes to minimize the number of incorrect predictions. Since we assume that the learner must eventually make a prediction for each matrix entry, the number of incorrect predictions depends on the size of the matrix.

Unlike problems typically studied, in which the natural measure of the size of the learner's problem is the size of an instance (or example), for this problem the natural measure is the size of the matrix. Such concept classes with polynomial-sized instance spaces are uninteresting in Valiant's probably approximately correct (PAC) model of learning [27]. In this model instances are chosen randomly from an arbitrary unknown probability distribution on the instance space. A concept class is PAC-learnable if the learner, after seeing a number of instances that are polynomial in the problem size, can output a hypothesis that is correct on all but an arbitrarily small fraction of the instances with high probability. For concepts whose instance space has cardinality polynomial in the problem size, by asking to see enough instances the learner can see almost all of the probability weight of the instance space. Thus it is not hard to show that these concept classes are trivially PAC-learnable. One goal of our research is to build a framework for studying such problems.

To study learning algorithms for these concept classes we extend the basic mistake bound model [14], [15], [19] to the cases in which a helpful teacher or the learner selects the query sequence, and, in addition, to the cases in which instances are chosen by an adversary or according to a probability distribution on the instance space. Previously, helpful teachers have been used to provide counterexamples to conjectured concepts [1], [2] or to break up the concept into smaller subconcepts [23]. In our framework the teacher selects only the presentation order for the instances.

If the learner is to have any hope of doing better than random guessing, there must be some structure in the relation. Furthermore, since there are so many ways to structure a binary relation, we give the learner some prior knowledge about the nature of this structure. Not surprisingly, the learning task depends greatly on the prior knowledge provided. One way to impose structure is to restrict one set of objects to have relatively few types. For example, a circus may contain many animals but only a few different species. In the first part of this paper we study the case in which the learner has a priori knowledge that there are a limited number of object types. Namely, we restrict the matrix representing the relation to have at most k distinct row types. (Two rows are of the same type if they agree in all columns.) We define a *k-binary-relation* to be a binary relation for which the corresponding matrix has at most k row types. This restriction is satisfied whenever there are only k types of objects in the set of n objects being considered in the relation. The learner receives *no* other knowledge about the predicate forming the relation. With this restriction we prove that any prediction algorithm makes at least $(1-\beta)km + n\lfloor \lg(\beta k)\rfloor - (1-\beta)k\lfloor \lg(\beta k)\rfloor$ mistakes[1] in the worst case for any fixed $0 < \beta \leq 1$ against any query sequence. So for $\beta = \frac{1}{2}$, we get a lower bound of $\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1\rfloor$ on the number of mistakes made by any prediction algorithm. If computational efficiency is not a concern, the halving algorithm [4], [19] makes at most $km + (n-k)\lg k$ mistakes against any query sequence. (The halving algorithm predicts according to the majority of the feasible relations (or concepts), and thus each mistake halves the number of remaining relations.)

We present an efficient algorithm making at most $km + (n-k)\lfloor \lg k\rfloor$ mistakes in the case in which the learner chooses the query sequence. We prove a tight mistake bound[2] of $km + (n-k)(k-1)$ in the case in which the helpful teacher selects the query sequence. When the adversary selects the query sequence, we present an efficient algorithm for $k=2$ that makes at most $2m + n - 2$ mistakes, and for arbitrary k we present an efficient algorithm that makes at most $km + n\sqrt{(k-1)m}$ mistakes. We prove that any algorithm makes at least $km + (n-k)\lfloor \lg k\rfloor$ mistakes in the case in which an adversary selects the query sequence, and we use the existence of projective geometries to improve this lower bound to $\Omega(km + (n-k)\lfloor \lg k\rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ for a large class of algorithms. Finally, we describe a technique for simplifying the proof of expected mistake bounds when the query sequence is chosen at random, and we use it to prove an $O(km + nk\sqrt{H})$ expected mistake bound for a simple algorithm. (Here H is the maximum Hamming distance between any two rows.)

Another possibility for known structure is the problem of learning a binary relation on a set where the predicate induces a total order on the set. (For example, the predicate may be "<".) In the second half of this paper we study the case in which the learner has *a priori* knowledge that the relation forms a total order. Once again, we see that the halving algorithm [4], [19] yields a good mistake bound against any query sequence. This motivates a second goal of this research: to develop efficient implementations of the halving algorithm. We uncover an interesting application of randomized approximation schemes to computational learning theory. Namely, we describe a technique that uses a fully polynomial randomized approximation scheme (fpras) to implement a randomized version of the halving algorithm. We apply this technique, using a fpras due to Dyer, Frieze, and Kannan [10] and to Matthews [22] for counting the number of linear extensions of a partial order, to obtain a polynomial prediction algorithm that makes at most $n\lg n + (\lg e)\lg n$ mistakes with very high probability against an adversary-selected query sequence. The small probability of making too many mistakes is determined by the coin flips of the learning algorithm and not by the query sequence selected by the adversary. We contrast this result with an $n-1$ mistake bound when the learner selects the query sequence [28] and with an $n-1$ mistake bound when a teacher selects the query sequence.

The remainder of this paper is organized as follows. In the next section we formally introduce the basic problem, the learning scenario, and the extended mistake bound model. In §3 we present our results for learning k-binary-relations. We first give a motivating example and present some general mistake bounds. In the following subsections we consider query sequences selected by the learner, by a helpful teacher, by an adversary, or at random. In §4 we turn our attention to the problem of learning total orders. We begin by discussing

[1]Throughout this paper we use lg to denote $\log_2$.

[2]The tight mistake bound is a worst-case mistake bound taken over all consistent learners; see §2 for formal definitions.

the relationship between the halving algorithm and approximate counting schemes in §4.1. In particular, we describe how an fpras can be used to implement an approximate halving algorithm. Then in §4.2 we present our results on learning a total order. Finally, in §5 we conclude with a summary and discussion of related open problems.

2. Learning scenario and mistake bound model. In this section we give formal definitions and discuss the learning scenario used in this paper. To be consistent with the literature we discuss these models in terms of concept learning. As we have mentioned, the problem of learning a binary relation can be viewed in this framework by letting the instance space be all pairs of objects, one from each of the two sets.

A *concept* c is a Boolean function on some domain of instances. A *concept class* C is a family of concepts. The learner's goal is to infer some unknown target concept chosen from some *known* concept class. Often C is decomposed into subclasses C_n according to some natural dimension measure n. That is, for each $n \geq 1$ let X_n denote a finite *learning domain*. Let $X = \bigcup_{n \geq 1} X_n$, and let $x \in X$ denote an *instance*. To illustrate these definitions we consider the concept class of monomials. (A monomial is a conjunction of literals, where each literal is either some Boolean variable or its negation.) For this concept class n is just the number of variables. Thus $|X_n| = 2^n$, where each $x \in X_n$ is chosen from $\{0, 1\}^n$ and represents the assignment for each variable. For each $n \geq 1$ let C_n be a *family of concepts* on X_n. Let $C = \bigcup_{n \geq 1} C_n$ denote a *concept class* over X. For example, if C_n contains monomials over n variables, then C is the class of all monomials. Given any concept $c \in C_n$, we say that x is a *positive instance* of c if $c(x) = 1$, and we say that x is a *negative instance* of c if $c(x) = 0$. In our example the target concept for the class of monomials over five variables might be $x_1\overline{x_4}x_5$. Then the instance 10001 is a positive instance and the instance 00001 is a negative instance. Finally, the *hypothesis space* of algorithm A is simply the set of all hypotheses (or rules) h that A may output. (A hypothesis for C_n must make a prediction for each $x \in X_n$.)

A *prediction algorithm* for C is an algorithm that runs under the following scenario. A *learning session* consists of a set of *trials*. In each trial the learner is given an unlabeled instance $x \in X_n$. The learner uses its current hypothesis to predict whether x is a positive or negative instance of the target concept $c \in C_n$, and then the learner is told the correct classification of x. If the prediction is incorrect, the learner has made a *mistake*. Note that in this model there is no training phase. Instead, the learner receives *unlabeled instances* throughout the entire learning session. However, after each prediction the learner discovers the correct classification. This feedback can then be used by the learner to improve the learner's hypothesis. A learner is *consistent* if on every trial there is some concept in C_n that agrees both with the learner's prediction and with all the labeled instances observed on preceding trials.

The number of mistakes made by the learner depends on the sequence of instances presented. We extend the mistake bound model to include several methods for the selection of instances. A *query sequence* is a permutation $\pi = \langle x_1, x_2, \ldots, x_{|X_n|} \rangle$ of X_n, where x_t is the instance presented to the learner at the tth trial. We call the agent selecting the query sequence the *director*. We consider the following directors:

Learner. The learner chooses π. To select x_t the learner may use time polynomial in n and all information obtained in the first $t - 1$ trials. In this case we say that the learner is *self-directed*.

Helpful teacher. A teacher who knows the target concept and wants to minimize the learner's mistakes chooses π. To select x_t the teacher uses knowledge of the target concept, $x_1, \ldots, x_{t-1}$, and the learner's predictions on $x_1, \ldots, x_{t-1}$. To avoid allowing the learner and teacher to have a coordinated strategy, in this scenario we consider the worst-case mistake bound over all consistent learners. In this case we say the learner is *teacher directed*.

TABLE 1
Summary of testing reactions for allergy testing example.

Degree of patient allergy	Epicutaneous (scratch)	Intradermal (under the skin)
Not allergic	Negative	Negative
Mildly allergic	Negative	Weak positive
Highly allergic	Weak positive	Strong positive

Adversary. The adversary who selected the target concept chooses π. This adversary, who tries to maximize the learner's mistakes, knows the learner's algorithm and has unlimited computing power. In this case we say the learner is *adversary directed.*

Random. In this model, π is selected randomly according to a uniform probability distribution on the permutations of X_n. Here the number of mistakes made by the learner for some target concept c in C_n is defined to be the expected number of mistakes over all possible query sequences. In this case we say the learner is *randomly directed.*

We consider how a prediction algorithm's performance depends on the director. Namely, we let $\mathrm{MB}_Z(A, C_n)$ denote the worst-case number of mistakes made by A for any target concept in C_n when the query sequence is provided by Z. (When Z = adversary, $\mathrm{MB}_Z(A, C_n) = M_A(C_n)$ in the notation of Littlestone [19].) We say that A is a *polynomial prediction algorithm* if A makes each prediction in time polynomial in n.

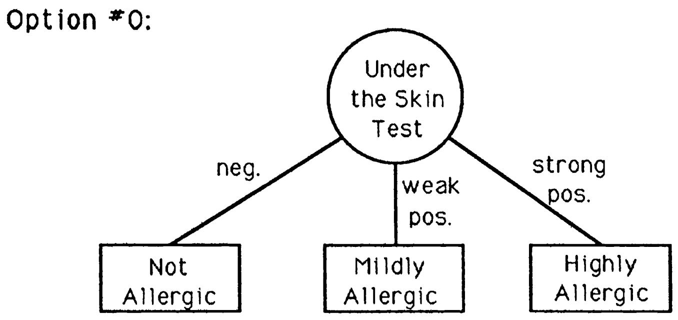
3. Learning binary relations. In this section we apply the learning scenario of the extended mistake bound model to the concept class C of k-binary-relations. For this concept class the dimension measure is denoted by n and m and by $X_{n,m} = \{1, \ldots, n\} \times \{1, \ldots, m\}$. An instance (i, j) is in the target concept $c \in C_{n,m}$ if and only if the matrix entry in row i and column j is a 1. So in each trial the learner is repeatedly given an instance x from $X_{n,m}$ and is asked to predict the corresponding matrix entry. After making a prediction the learner is told the correct value of the matrix entry. The learner wishes to minimize the number of incorrect predictions during a learning session in which the learner must eventually make a prediction for each matrix entry.

We begin this section with a motivating example from the domain of allergy testing. We use this example to motivate both the restriction that the matrix has k row types and the use of the extended mistake bound model. We then present general upper and lower bounds on the number of mistakes made by the learner, regardless of the director. Finally, we study the complexity of learning a k-binary-relation under each director.

3.1. Motivation: Allergist example. In this subsection we use the following example taken from the domain of allergy testing to motivate the problem of learning a k-binary-relation.

Consider an allergist with a set of patients to be tested for a given set of allergens. Each patient is either highly allergic, mildly allergic, or not allergic to any given allergen. The allergist may use either an epicutaneous (scratch) test, in which the patient is given a fairly low dose of the allergen, or an intradermal (under the skin) test, in which the patient is given a larger dose of the allergen. The patient's reaction to the test is classified as strong positive, weak positive, or negative. Table 1 describes the reaction that occurs for each combination of allergy level and dosage level. Finally, we assume that a strong positive reaction is extremely uncomfortable to the patient but not dangerous.

What options does the allergist have in testing a patient for a given allergen? One option (option 0) is just to perform the intradermal test. Another option (option 1) is to perform an epicutaneous test and, if it is not conclusive, then perform an intradermal test. (See Fig. 1 for decision trees describing these two testing options.) Which testing option is best? If the



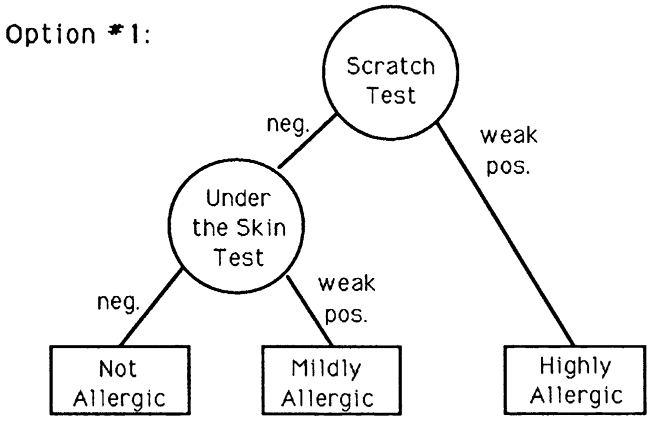


FIG. 1. *Testing options available to the allergist.*

patient has either no allergy or a mild allergy to the given allergen, then testing option 0 is best, since the patient need not return for the second test. However, if the patient is highly allergic to the given allergen, then testing option 1 is best, since the patient does not experience a bad reaction. We assume that the inconvenience of going to the allergist twice is approximately the same as having a bad reaction. That is, the allergist has no preference for an error in a particular direction. Although the allergist's final goal is to determine each patient's allergies, we consider the problem of learning the optimal testing option for each combination of patient and allergen.

The allergist interacts with the environment as follows. In each trial the allergist is asked to predict the best testing option for a given patient–allergen pair. The allergist is then told the testing results, thus learning whether the patient is not allergic, mildly allergic, or highly allergic to the given allergen. In other words, the allergist receives feedback as to the correct testing option. Note that we make no restrictions on how the hypothesis is represented, as long as it can be evaluated in polynomial time. In other words, all we require is that given any patient–allergen pair, the allergist decides which test to perform in a reasonable amount of time.

How can the allergist possibly predict a patient's allergies? If the allergies of the patients are completely random, then there is not much hope. What prior knowledge does the allergist have? He or she knows that people often have exactly the same allergies, and so there is a set of allergy types that occur frequently. (We do not assume that the allergist has *a priori* knowledge of the actual allergy types.) This knowledge can help guide the allergist's predictions.

Having specified the problem, we discuss our choice of using the extended mistake bound model to evaluate learning algorithms for this problem. First of all, observe that we want an on-line model. There is no training phase here; the allergist wants to predict the correct testing option for each patient–allergen pair. Also, we expect that the allergist has time to test each patient for each allergen; that is, the instance space is polynomial sized. Thus, as discussed in §1, the distribution-free model is not appropriate.

How should we judge the performance of the learning algorithm? For each wrong prediction made, a patient is inconvenienced by making a second trip or having a bad reaction. Since the learner wants to give all patients the best possible service, he or she strives to minimize the number of incorrect predictions made. Thus we want to use the absolute mistake bound success criterion. Namely, we judge the performance of the learning algorithm by the number of incorrect predictions made during a learning session in which the allergist must eventually test each patient for each allergen.

Up to now the standard on-line model (which uses absolute mistake bounds to judge the learners) appears to be the appropriate model. We now discuss the selection of the instances. Since the allergist has no control over the target relation (i.e., the allergies of the patients), it makes sense to view the feedback as coming from an adversary. However, do we really want an adversary to select the presentation order for the instances? It could be that the allergist is working for a cosmetic company and, because of the restrictions of the Food and Drug Administration and the cosmetic company, the allergist is essentially told when to test each person for each allergen. In this case it is appropriate to have an adversary select the presentation order. However, in the typical situation the allergist can decide in what order to perform the testing so that he or she can make the best predictions possible. In this case we want to allow the learner to select the presentation order. One could also imagine a situation in which an intern is being guided by an experienced allergist; in this case a teacher helps to select the presentation order. Finally, random selection of the presentation order may provide us with a better feeling for the behavior of an algorithm.

3.2. Learning k-binary-relations. In this section we begin our study of learning k-binary-relations by presenting general lower and upper bounds on the mistakes made by the learner, regardless of the director.

Throughout this section we use the following notation: We say an entry (i, j) of the matrix (M_{ij}) is *known* if the learner was previously presented that entry. We assume without loss of generality that the learner is never asked to predict the value of a known entry. We say that rows i and i' are *consistent* (given the current state of knowledge) if $M_{ij} = M_{i'j}$ for all columns j in which both entries (i, j) and (i', j) are known.

We now look at general lower and upper bounds on the number of mistakes that apply for all directors. First of all, note that $k \leq 2^m$ since there are only 2^m possible row types for a matrix with m columns. Clearly, any learning algorithm makes at least km mistakes for some matrix, regardless of the query sequence. The adversary can divide the rows into k groups and reply that the prediction was incorrect for the first column queried for each entry of each group. We generalize this approach to force mistakes for more than one row of each type.

THEOREM 3.1. *For any* $0 < \beta \leq 1$ *any prediction algorithm makes at least* $(1-\beta)km + n\lfloor \lg(\beta k)\rfloor - (1-\beta)k\lfloor \lg(\beta k)\rfloor$ *mistakes, regardless of the query sequence.*

Proof. The adversary selects its feedback for the learner's predictions as follows. For each entry in the first $\lfloor \lg(\beta k) \rfloor$ columns the adversary replies that the learner's response is incorrect. At most βk new row types are created by this action. Likewise, for each entry in the first $(1-\beta)k$ rows the adversary replies that the learner's response is incorrect. This creates at most $(1-\beta)k$ new row types. The adversary makes all remaining entries in the matrix zero (see Fig. 2). The number of mistakes is at least the area of the unmarked region. Thus the adversary has forced at least $(1-\beta)km + n\lfloor \lg(\beta k) \rfloor - (1-\beta)k\lfloor \lg(\beta k) \rfloor$ mistakes while creating at most $\beta k + (1-\beta)k = k$ row types. $\square$

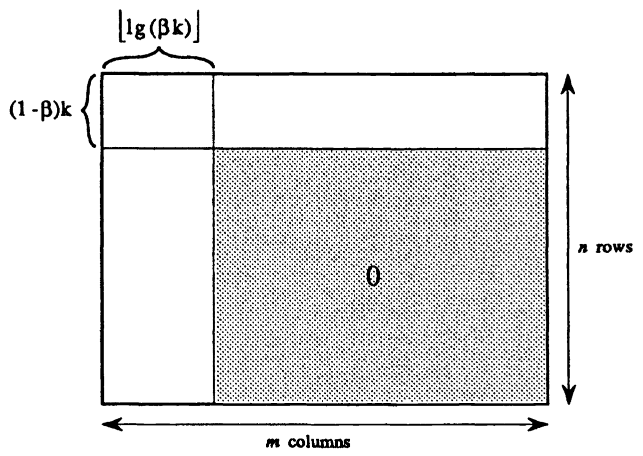


FIG. 2. *Final matrix created by the adversary in the proof of Theorem* 3.1. *All entries in the unmarked area will contain the bit not predicted by the learner; that is, a mistake is forced on each entry in the unmarked area. All entries in the marked area will be zero.*

By letting $\beta = \frac{1}{2}$ we obtain the following corollary.

COROLLARY 3.2. *Any algorithm makes at least* $\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$ *mistakes in the worst case, regardless of the query sequence.*

If computational efficiency is not a concern, for all query sequences the halving algorithm [4], [19] provides a good mistake bound.

Observation. The halving algorithm achieves a $km + (n-k)\lg k$ mistake bound.

Proof. We use a simple counting argument on the size of the concept class $C_{n,m}$. There are 2^{km} ways to select the k row types, and there are $k^{(n-k)}$ ways to assign one of the k row types to each of the remaining $n-k$ rows. Thus $|C_{n,m}| \leq 2^{km}k^{(n-k)}$. Littlestone [19] proves that the halving algorithm makes at most $\lg|C_{n,m}|$ mistakes. Thus the number of mistakes made by the halving algorithm for this concept class is at most $\lg(2^{km}k^{(n-k)}) \leq km + (n-k)\lg k$. $\square$

In the remainder of this section we study efficient prediction algorithms designed to perform well against each of the directors. In some cases we are also able to prove lower bounds that are better than that of Theorem 3.1. In §3.3 we consider the case in which the query sequence is selected by the learner. We study the helpful-teacher director in §3.4. In §3.5 we consider the case of an adversary director. Finally, in §3.6 we consider instances drawn uniformly at random from the instance space.

3.3. Self-directed learning. In this section we present an efficient algorithm for the case of self-directed learning.

THEOREM 3.3. *There exists a polynomial prediction algorithm that achieves a $km + (n - k)\lfloor \lg k \rfloor$ mistake bound with a learner-selected query sequence.*

Proof. The query sequence selected simply specifies the entries of the matrix in row-major order. The learner begins by assuming that there is only one row type. Let $\hat{k}$ denote the learner's current estimate for k. Initially $\hat{k} = 1$. For the first row the learner guesses each entry. (This row becomes the template for the first row type.) Next the learner assumes that the second row is the same as the first row. If a mistake is made, then the learner revises the estimate for $\hat{k}$ to be 2, guesses for the rest of the row, and uses that row as the template for the second row type. In general, to predict M_{ij} the learner predicts according to a majority vote of the recorded row templates that are consistent with row i (breaking ties arbitrarily). Thus if a mistake is made, then at least half of the row types can be eliminated as the potential type of row i. If more than $\left\lfloor \lg \hat{k} \right\rfloor$ mistakes are made in a row, then a new row type has been found. In this case, $\hat{k}$ is incremented, the learner guesses for the rest of the row, and the learner makes this row the template for row type $\hat{k} + 1$.

How many mistakes are made by this algorithm? Clearly, at most m mistakes are made for the first row found of each of the k types. For the remaining $n - k$ rows, since $\hat{k} \le k$, at most $\lfloor \lg k \rfloor$ mistakes are made. □

Observe that this upper bound is within a constant factor of the lower bound of Corollary 3.2. Furthermore, we note that this algorithm need not know k a priori. In fact, it obtains the same mistake bound even if an adversary tells the learner which row to examine and in what order to predict the columns, provided that the learner sees all of a row before going on to the next. As we will later see, this problem becomes harder if the adversary can select the query sequence without restriction.

3.4. Teacher-directed learning. In this section we present upper and lower bounds on the number of mistakes made under the helpful-teacher director. Recall that in this model we consider the worst-case mistake bound over all consistent learners. Thus the question asked here is: What is the minimum number of matrix entries a teacher must reveal so that there is a unique completion of the matrix? That is, until there is a unique completion of the partial matrix, a mistake could be made on the next prediction.

We now prove an upper bound on the number of entries needed to uniquely define the target matrix.

THEOREM 3.4. *The number of mistakes made with a helpful teacher as the director is at most $km + (n - k)(k - 1)$.*

Proof. First the teacher presents the learner with one row of each type. For each of the remaining $n - k$ rows the teacher presents an entry to distinguish the given row from each of the $k - 1$ incorrect row types. After these $km + (n - k)(k - 1)$ entries have been presented we claim that there is a unique matrix with at most k row types that is consistent with the partial matrix. Since all k distinct row types have been revealed in the first stage, all remaining rows must be the same as one of the first k rows presented. However, each of the remaining rows have been shown to be inconsistent with all but one of these k row templates. □

Is Theorem 3.4 the best such result possible? Clearly, the teacher must present a row of each type. But, in general, is it really necessary to present $k - 1$ entries of the remaining rows to uniquely define the matrix? We now answer this question in the affirmative by presenting a matching lower bound.

THEOREM 3.5. *The number of mistakes made with a helpful teacher as the director is at least* $\min\{nm, km + (n - k)(k - 1)\}$.

Proof. The adversary selects the following matrix. The first row type consists of all zeros. For $2 \le z \le \min\{m + 1, k\}$, row type z contains $z - 2$ zeros, followed by a one, followed by

$m - z + 1$ zeros. The first k rows are each assigned to be a different one of the k row types. Each remaining row is assigned to be the first row type (see Fig. 3). Until there is a unique completion of the partial matrix, by definition there exists a consistent learner that could make a mistake. Clearly, if the learner has not seen each column of each row type, then the final matrix is not uniquely defined. This part of the argument accounts for km mistakes. When $m + 1 \geq k$, for the remaining rows, unless all of the first $k - 1$ columns are known, there is some row type besides the first row type that must be consistent with the given row. This argument accounts for $(n - k)(k - 1)$ mistakes. Likewise, when $m + 1 < k$, if any of the first m columns are not known then there is some row type besides the first row type that must be consistent with the given row. This accounts for $(n - k)m$ mistakes. Thus the total number of mistakes is at least $\min\{nm, km + (n - k)(k - 1)\}$. □

$$\begin{array}{c|ccccccccc|}
\hline
 & 0&0&0&0&0&0&0&0&0\\
 & 1&0&0&0&0&0&0&0&0\\
\text{5 row types}\ \{ & 0&1&0&0&0&0&0&0&0\\
 & 0&0&1&0&0&0&0&0&0\\
 & 0&0&0&1&0&0&0&0&0\\
 & 0&0&0&0&0&0&0&0&0\\
 & &&&&\vdots&&&&\\
 & 0&0&0&0&0&0&0&0&0\\
\hline
\end{array}$$

FIG. 3. *Matrix created by the adversary against the helpful teacher director. In this example five row types appear in the first five rows of the matrix.*

Because of the requirement that mistake bounds in the teacher-directed case apply to all consistent learners, we note that it is possible to get mistake bounds that are not as good as those obtained when the learner is self-directed. Recall that in §3.2 we proved a $km + (n - k)\lfloor \lg k \rfloor$ mistake bound for the learner director. This bound is better than that obtained with a teacher because the learner uses a majority vote among the known row types for making predictions. However, a consistent learner may use a *minority vote* and could thus make $km + (n - k)(k - 1)$ mistakes.

3.5. Adversary-directed learning. In this section we derive upper and lower bounds on the number of mistakes made when the adversary is the director. We first present a stronger information-theoretic lower bound on the number of mistakes an adversary can force the learner to make. Next we present an efficient prediction algorithm that achieves an optimal mistake bound if $k \leq 2$. We then consider the related problem of computing the minimum number of row types needed to complete a partially known matrix. Finally, we consider learning algorithms that work against an adversary for arbitrary k.

We now present an information-theoretic lower bound on the number of mistakes made by any prediction algorithm when the adversary selects the query sequence. We obtain this result by modifying the technique used in Theorem 3.1.

THEOREM 3.6. *Any prediction algorithm makes at least* $\min\{nm, km + (n - k)\lfloor \lg k \rfloor\}$ *mistakes against an adversary-selected query sequence.*

Proof. The adversary starts by presenting all entries in the first $\lfloor \lg k \rfloor$ columns (or m columns if $m < \lfloor \lg k \rfloor$) and by replying that each prediction is incorrect. If $m \geq \lfloor \lg k \rfloor$, this step causes the learner to make $n\lfloor \lg k \rfloor$ mistakes. Otherwise, this step causes the learner to make nm mistakes. Each row can now be classified as one of k row types. Next the adversary presents the remaining columns for one row of each type, again replying that each prediction is incorrect. For $m \geq \lfloor \lg k \rfloor$ this step causes the learner to make $k(m - \lfloor \lg k \rfloor)$ additional mistakes. For the remaining matrix entries the adversary replies as dictated by the completed row of the same row type as the given row. So the number of mistakes made by the learner is at least $\min\{nm, n\lfloor \lg k \rfloor + km - k\lfloor \lg k \rfloor\} = \min\{nm, km + (n-k)\lfloor \lg k \rfloor\}$. □

Special case: $k = 2$. We now consider efficient prediction algorithms for learning the matrix under an adversary-selected query sequence. (Recall that if efficiency is not a concern, the halving algorithm makes at most $km + (n-k)\lg k$ mistakes.) In this section we consider the case in which $k \leq 2$ and present an efficient prediction algorithm that performs optimally.

THEOREM 3.7. *There exists a polynomial prediction algorithm that makes at most* $2m + n - 2$ *mistakes against an adversary-selected query sequence for* $k = 2$.

Proof. The algorithm uses a graph G whose vertices correspond to the rows of the matrix and that initially has no edges. To predict M_{ij} the algorithm 2-colors the graph G and then proceeds as follows:

1. If no entry of column j is known, it guesses randomly.
2. Else if every known entry of column j is zero (respectively, one), it guesses zero (one).
3. Else it finds a row i' assigned the same color as i and known in column j, and it guesses $M_{i'j}$.

Finally, after the prediction is made and the feedback received, the graph G is updated by adding an edge $\overline{ii'}$ to G for each row i' known in column j for which $M_{ij} \neq M_{i'j}$. Note that one of the above cases always applies. Also, since $k = 2$, it will always be possible to find a 2-coloring.

How many mistakes can this algorithm make? It is not hard to see that cases 1 and 2 each occur only once for every column, and so there are at most m mistakes made in each of these cases. Furthermore, the first case-2 mistake adds at least one edge to G. We now argue that each case-3 mistake reduces the number of connected components of G by at least 1. We use a proof by contradiction. That is, assume that a case-3 mistake does *not* reduce the number of connected components. Then it follows that the edge $e = \overline{v_1 v_2}$ added to G must form a cycle (see Fig. 4). We now separately consider the cases in which this cycle contains an odd number of edges or an even number of edges.

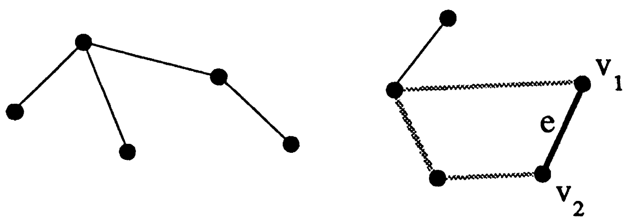


FIG. 4. *Situation that occurs if a case-3 mistake does not reduce the number of connected components of G. The thick gray edges and the thick black edge show the cycle created in G. Let e (shown as a thick black edge) be the edge added to form the cycle.*

Case 1: *Odd-length cycle.* Since G is known to be 2-colorable, this case cannot occur.

Case 2: *Even-length cycle.* Before e is added, since v_1 and v_2 were connected by an odd number of edges, in any legal 2-coloring they must have been different colors. Since step 3 of the algorithm picks nodes of the same color, an edge could have never been placed between v_1 and v_2. Thus we again have a contradiction.

In both cases we reach a contradiction, and thus we have shown that every case-3 mistake reduces the number of connected components of G. Thus after at most $n-2$ case-3 mistakes, G must be fully connected and thus there must be a unique 2-coloring[3] of G and no more mistakes can occur. Thus the worst-case number of mistakes made by this algorithm is $2m+n-2$. □

Note that for $k=2$ this upper bound matches the information-theoretic lower bound of Theorem 3.6. Also note that if there is only one row type, then the algorithm given in Theorem 3.7 makes at most m mistakes, matching the information-theoretic lower bound.

An interesting theoretical problem is to find a linear mistake bound for constant $k \geq 3$ when provided with a k-colorability oracle. However, such an approach would have to be greatly modified to yield a polynomial prediction algorithm since a polynomial-time k-colorability oracle exists only if $\mathcal{P} = \mathcal{NP}$. Furthermore, even good polynomial-time approximations to a k-colorability oracle are not known [5], [18].

The remainder of this section focuses on designing polynomial prediction algorithms for the case in which the matrix has at least three row types. One approach that may seem promising is to make predictions as follows: Compute a matrix that is consistent with all known entries and that has the fewest possible row types; then use this matrix to make the next prediction. We now show that even computing the minimum number of row types needed to complete a partially known matrix is $\mathcal{NP}$-complete. Formally, we define the *matrix k-complexity* problem as follows: Given an $n \times m$ binary matrix M that is partially known, decide if there is some matrix with at most k row types that is consistent with M. The matrix k-complexity problem can be shown to be $\mathcal{NP}$-complete by a reduction from graph k-colorability for any fixed $k \geq 3$.

THEOREM 3.8. *For fixed $k \geq 3$ the matrix k-complexity problem is $\mathcal{NP}$-complete.*

Proof. Clearly, this problem is in $\mathcal{NP}$ since we can easily verify that a guessed matrix has k row types and is consistent with the given partial matrix.

To show that the problem is $\mathcal{NP}$-complete, we use a reduction from graph k-colorability. Given an instance $G=(V,E)$ of graph k-colorability we transform it into an instance of the matrix k-complexity problem. Let $m=n=|V|$. For each edge $\{v_i, v_j\} \in E$ we add entries to the matrix so that row i and row j cannot be the same row type. Specifically, for each vertex v_i we set $M_{ii}=0$, and $M_{ji}=1$ for each neighbor v_j of v_i. An example demonstrating this reduction is given in Fig. 5.

We now show that there is some matrix of at most k row types that is consistent with this partial matrix if and only if G is k-colorable. We first argue that if there is a matrix M' consistent with M that has at most k row types, then G is k-colorable. By construction, if two rows are of the same type, there cannot be an edge between the corresponding nodes. So just let the node color for each node be the type of the corresponding row in M'.

Conversely, if G is k-colorable, then there exists a matrix M' consistent with M that has at most k row types. By the construction of M, if a set of vertices are the same color in G, then the corresponding rows are consistent with each other. Thus there exists a matrix with at most k row types that is consistent with M. □

Row-filter algorithms. In this section we study the performance of a whole class of algorithms designed to learn a matrix with arbitrary complexity k when an adversary selects the query sequence. We say that an algorithm A is a *row-filter algorithm* if A makes its prediction for M_{ij} strictly as a function of j and all entries in the set I of rows consistent with row i and defined in column j. That is, A's prediction is $f(I, j)$, where f is some (possibly

[3]Two 2-colorings under renaming of the colors are considered to be the same.

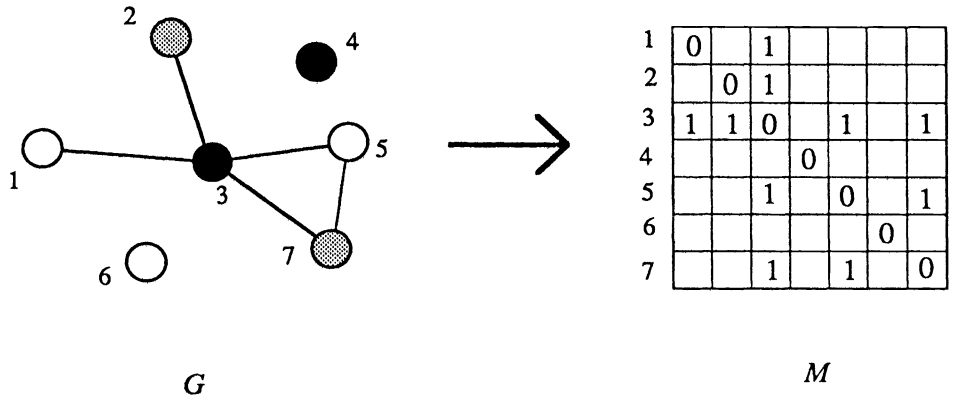


FIG. 5. *Example of the reduction used in Theorem* 3.8. *Graph G is the instance for the graph coloring problem, and partial matrix M is the instance for the matrix complexity problem. Note that there exists a matrix that is a completion of M that uses only three row types. The corresponding* 3*-coloring of G is demonstrated by the node colorings used in G.*

probabilistic) function. So to make a prediction for M_{ij} a row-filter algorithm considers all rows that could be the same type as row i and whose value for column j is known and uses these rows in any way one could imagine to make a prediction. For example, it could take a majority vote on the entries in column j of all rows that are consistent with row i. Or, of the rows defined in column j, it could select the row that has the most known values in common with row i and predict according to its entry in column j. We have found that many of the prediction algorithms we considered are row-filter algorithms.

Consider the simple row-filter algorithm *ConsMajorityPredict*, in which $f(I, j)$ computes the majority vote of the entries in column j of the rows in I. (Guess randomly in the case of a tie.) Note that *ConsMajorityPredict* takes only time linear in the number of known entries of the matrix to make a prediction. We now give an upper bound on the number of mistakes made by *ConsMajorityPredict*.

THEOREM 3.9. *The algorithm ConsMajorityPredict makes at most $km + n\sqrt{(k-1)m}$ mistakes against an adversary-selected query sequence.*

Proof. For all i let $d(i)$ be the number of rows consistent with row i. We define the *potential* of a partially known matrix to be $\Phi = \sum_{i=1}^{n} d(i)$. We first consider how much the potential function can change over the entire learning session. □

LEMMA 3.10. *The potential function Φ decreases by at most $\frac{k-1}{k}n^2$ during the learning session.*

Proof. Initially, for all i, $d(i) = n$. So $\Phi_{\text{init}} = n^2$. Let $C(z)$ be the number of rows of type z for $1 \le z \le k$. By definition, $\Phi_{\text{final}} = \sum_{z=1}^{k} C(z)^2$. Thus our goal is to minimize $\sum_{z=1}^{k} C(z)^2$ under the constraint that $\sum_{z=1}^{k} C(z) = n$. Using the method of Lagrange multipliers, we obtain that Φ_{final} is minimized when for all z, $C(z) = n/k$. Thus $\Phi_{\text{final}} \ge (\frac{n}{k})^2 k = \frac{n^2}{k}$. So $\Delta\Phi = \Phi_{\text{init}} - \Phi_{\text{final}} \le n^2 - \frac{n^2}{k} = \frac{k-1}{k}n^2$. □

Now that the total decrease in Φ over the learning session is bounded, we need to determine how many mistakes can be made without Φ decreasing by more than $\frac{k-1}{k}n^2$. We begin by noting that Φ is strictly nonincreasing. Once two rows are found to be inconsistent, they remain inconsistent. So to bound the number of mistakes made by *ConsMajorityPredict* we must compute a lower bound on the amount Φ is decreased by each mistake. Intuitively, one expects Φ to decrease by larger amounts as more of the matrix is seen. We formalize this intuition in the next two lemmas. For a given row type z let $B(j, z)$ denote the set of matrix entries that are in column j of a row of type z.

LEMMA 3.11. *The* rth *mistake made when predicting an entry in* $B(j, z)$ *causes* Φ *to decrease by at least* $2(r-1)$.

Proof. Suppose that this mistake occurs in predicting entry (i, j) where row i is of type z. Consider all the rows of type z. Since $r-1$ mistakes have occurred in column j, at least $r-1$ entries of $B(j, z)$ are known. Since *ConsMajorityPredict* is a row-filter algorithm, these rows must be in I. Furthermore, *ConsMajorityPredict* uses a majority voting scheme, and thus if a mistake occurs there must be at least $r-1$ entries in I (and thus consistent with row i) that differ in column j with row i. Thus if a mistake is made, row i is found to be inconsistent with at least $r-1$ rows it was thought to be consistent with. When two previously consistent rows are found to be inconsistent, Φ decreases by 2. Thus the total decrease in Φ caused by the rth mistake made when an entry is predicted in $B(j, z)$ is at least $2(r-1)$. □

From Lemma 3.11 we see that the more entries known in $B(j, z)$, the greater the decrease in Φ for future mistakes on such entries. So intuitively it appears that the adversary can maximize the number of mistakes made by the learner by balancing the number of entries seen in $B(j, z)$ for all j and z. We prove that this intuition is correct and apply it to obtain a lower bound on the amount Φ must have decreased after the learner has made μ mistakes.

LEMMA 3.12. *After* μ *mistakes are made, the total decrease in* Φ *is at least* $km\left(\frac{\mu}{km}-1\right)^2$.

Proof. From Lemma 3.11, after the rth mistake in the prediction of an entry from $B(j, z)$, the total decrease in Φ from its initial value is at least $\sum_{x=1}^{r} 2(x-1) \geq (r-1)^2$. Let $W(j, z)$ be the number of mistakes made in column j of rows of type z. The total decrease in Φ is at least

$$D = \sum_{j=1}^{m} \sum_{z=1}^{k} (W(j,z)-1)^2,$$

subject to the constraint $\sum_{j=1}^{m} \sum_{z=1}^{k} W(j, z) = \mu$.

Using the method of Lagrange multipliers, we obtain that D is minimized when $W(j, z) = \frac{\mu}{km}$ for all j and z. (Since any algorithm clearly must make km mistakes, $\mu \geq km$ and thus $\mu/km \geq 1$.) So the total decrease in Φ is at least

$$\sum_{j=1}^{m} \sum_{z=1}^{k} \left(\frac{\mu}{km}-1\right)^2 = km\left(\frac{\mu}{km}-1\right)^2. \quad \square$$

We now complete the proof of the theorem. Combining Lemma 3.10 and Lemma 3.12, along with the observation that Φ is strictly nonincreasing, we have shown that

$$km\left(\frac{\mu}{km}-1\right)^2 \leq \frac{k-1}{k}n^2.$$

This implies that $\mu \leq km + n\sqrt{(k-1)m}$. □

We note that by using the simpler argument that each mistake, except for the first mistake in each column of each row type, decreases Φ by at least 2, we obtain a $km + \frac{k-1}{2k}n^2$ mistake bound for *any* row-filter algorithm. Also, Goldman and Warmuth [12] give an algorithm, based on the weighted majority algorithm of Littlestone and Warmuth [20], that achieves an $O(km + n\sqrt{m}\lg k)$ mistake bound. Their algorithm builds a complete graph of n vertices in which row i corresponds to vertex v_i and all edges have initial weights of 1. To predict a value for (i, j) the learner takes a weighted majority of all active neighbors of v_i (v_k is *active* if M_{kj} is known). After receiving feedback the learner sets the weight on the edge from v_i to v_k to be 0 if $M_{kj} \neq M_{ij}$. Finally, if a mistake occurs, the learner doubles the weight of (v_i, v_k) if $M_{kj} = M_{ij}$ (i.e., the edges to neighbors that predicted correctly). We note that this algorithm is not a row-filter algorithm.

Does *ConsMajorityPredict* give the best performance possible by a row-filter algorithm? We now present an information-theoretic lower bound on the number of mistakes an adversary can force against any row-filter algorithm.

THEOREM 3.13. *Any row-filter algorithm for learning an $n \times m$ matrix with $m \geq \frac{n}{2}$ and $k \geq 2$ makes $\Omega(n\sqrt{m})$ mistakes when the adversary selects the query sequence.*

Proof. We assume that the adversary knows the learner's algorithm and has access to any random bits the learner uses. (One can prove a similar lower bound on the expected mistake bound when the adversary cannot access the random bits.)

X	X		X			
	X	X		X		
		X	X		X	
			X	X		X
X				X	X	
	X				X	X
X		X				X

FIG. 6. *Projective geometry for $p = 2$, $m' = 7$.*

Let $m' = (p^2 + p + 1)$ be the largest integer of the given form such that p is prime and $m' \leq m$. Without loss of generality we assume in the remainder of this proof that the matrix has m' columns, and we prove an $\Omega(n\sqrt{m'})$ mistake bound. From Bertrand's conjecture[4] it follows from this result that the adversary has forced $\Omega(n\sqrt{m})$ mistakes in the original matrix.

Our proof depends on the existence of a *projective geometry* Γ on m' points and lines [6]. That is, there exists a set of m' points and a set of m' lines such that each line contains exactly $p + 1$ points and each point is at the intersection of exactly $p + 1$ lines. Furthermore, any pair of lines intersects at exactly one point, and any two points define exactly one line. (The choice of $m' = p^2 + p + 1$ for p prime comes from the fact that projective geometries are known to exist only for such values.) Figure 6 shows a matrix representation of such a geometry; an "X" in entry (i, j) indicates that point j is on line i. Let Γ' denote the first $\lfloor \frac{n}{2} \rfloor$ lines of Γ. Note that since $m' \geq \frac{n}{2}$, all entries of Γ' are contained within M.

The matrix M consists of two row types: The odd rows are filled with ones and the even rows with zeros. Two consecutive rows of M are assigned to each line of Γ' (see Fig. 7). We now prove that the adversary can force a mistake for each entry of Γ'. The adversary's query sequence maintains the condition that an entry (i, j) is not revealed unless line $\lceil \frac{i}{2} \rceil$ of Γ' contains point j. In particular, the adversary will begin by presenting one entry of the matrix for each entry of Γ'. We prove that for each entry of Γ' the learner must predict the same value for the two corresponding entries of the matrix. Thus the adversary forces a mistake for the $\lfloor \frac{n}{2} \rfloor (p + 1) = \Omega(n\sqrt{m'})$ entries of Γ'. The remaining entries of the matrix are then presented in any order.

Let I be the set of rows that may be used by the row-filter algorithm when it predicts entry $(2i, j)$. Let I' be the set of rows that may be used by the row-filter algorithm when it predicts

[4]Bertrand's conjecture states that for any integer $n \geq 2$ there exists a prime p such that $n < p < 2n$. Although this is known as Bertrand's *conjecture*, it was proved by Chebyshev in 1831.

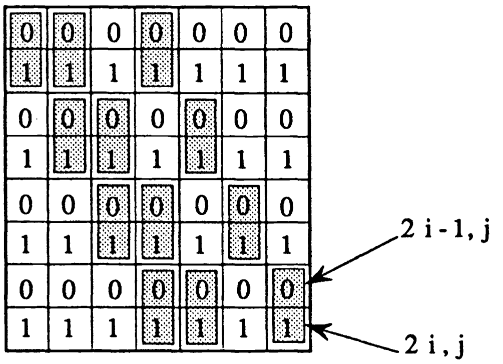


FIG. 7. *Matrix created by the adversary in the proof of Theorem* 3.13. *The shaded regions correspond to the entries in* Γ'. *The learner is forced to make a mistake on one of the entries in each shaded rectangle.*

entry $(2i-1, j)$. We prove by contradiction that $I = I'$. If $I \neq I'$, then it must be the case that there is some row r that is defined in column j and consistent with row $2i$, yet inconsistent with row $2i-1$ (or vice versa). By the definition of the adversary's query sequence it must be the case that lines $\lceil \frac{r}{2} \rceil$ and $\left\lceil \frac{(2i-1)}{2} \right\rceil = i$ of Γ' contain point j. Furthermore, since $(2i-1, j)$ is being queried, that entry is not known. Thus rows r and $2i-1$ must both be known in some other column j' since they are known to be inconsistent. Thus since only entries in Γ' are shown, it follows that lines $\lceil \frac{r}{2} \rceil$ and i of Γ' also contain point j' for $j' \neq j$. So this implies that lines $\lceil \frac{r}{2} \rceil$ and i of Γ' must intersect at two points, giving a contradiction. Thus $I = I'$, and so $f(I, j) = f(I', j)$ for entry $(2i, j)$ and entry $(2i-1, j)$. Since rows $2i$ and $2i-1$ differ in each column and the adversary has access to the random bits of the learner, the adversary can compute $f(I, j)$ just before making the query and then ask the learner to predict the entry for which the mistake will be made. This procedure is repeated for the pair of entries corresponding to each element of Γ'. □

We use a similar argument to get an $\Omega(m\sqrt{n})$ bound for $m < \frac{n}{2}$. This bound, combined with the lower bound of Theorem 3.6 and Theorem 3.13, permits us to obtain a $\Omega(km + (n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ lower bound on the number of mistakes made by a row-filter algorithm.

COROLLARY 3.14. *Any row-filter algorithm makes* $\Omega(km+(n-k)\lfloor \lg k \rfloor+\min\{n\sqrt{m}, m\sqrt{n}\})$ *mistakes against an adversary-selected query sequence.*

Comparing this lower bound to the upper bound proven for *ConsMajorityPredict*, we see that for fixed k the mistake bound of *ConsMajorityPredict* is within a constant factor of optimal.

Given this lower bound, one may question the $2m + n - 2$ upper bound for $k = 2$ given in Theorem 3.7. However, the algorithm described is not a row-filter algorithm. Also, compared to our results for the learner-selected query sequence, it appears that allowing the learner to select the query sequence is quite helpful.

3.6. Randomly directed learning. In this section we consider the case in which the learner is presented at each step with one of the remaining entries of the matrix selected uniformly and independently at random. We present a prediction algorithm that makes $O(km + nk\sqrt{H})$ mistakes on average, where H is the maximum Hamming distance between any two rows of the matrix. We note that when $H = \Omega(\frac{m}{k})$ the result of Theorem 3.9 supersedes this result. A key result of this section is a proof relating two different probabilistic models for analyzing the mistake bounds under a random presentation. We first consider a simple probabilistic model in which the requirement that t matrix entries are known is simulated by assuming that each entry of the matrix is seen independently with probability $\frac{t}{nm}$. We then prove that any upper bound obtained on the number of mistakes under this simple probabilistic model holds under the true model (to within a constant factor) in which there are exactly t entries known. This result is extremely useful since in the true model the dependencies among the probabilities that matrix entries are known makes the analysis significantly more difficult.

We define the algorithm *RandomConsistentPredict* to be the row-filter algorithm in which the learner makes a prediction for M_{ij} by choosing one row i' of I uniformly at random and predicting the value $M_{i'j}$. (If I is empty, then *RandomConsistentPredict* makes a random guess.)

THEOREM 3.15. *Let H be the maximum Hamming distance between any two rows of M. Then the expected number of mistakes made by RandomConsistentPredict is $O(k(n\sqrt{H}+m))$.*

Proof. Let U_t be the probability that the prediction rule makes a mistake on the $(t+1)$st step. That is, U_t is the chance that a prediction error occurs on the next randomly selected entry, given that exactly t other randomly chosen entries are already known. Clearly, the expected number of mistakes is $\sum_{t=0}^{S-1} U_t$, where $S = nm$. Our goal is to find an upper bound for this sum.

The condition that exactly t entries are known makes the computation of U_t rather messy since the probability of having seen some entry of the matrix is *not* independent of knowing the others. Instead, we compute the probability V_t of a mistake under the simpler assumption that each entry of the matrix has been seen with probability $\frac{t}{S}$, independent of the rest of the matrix. We first compute an upper bound for the sum $\sum_{t=0}^{S-1} V_t$, and we then show that this sum is within a constant factor of $\sum_{t=0}^{S-1} U_t$.

LEMMA 3.16. $\sum_{t=0}^{S-1} V_t = O(km + nk\sqrt{H})$.

Proof. Fix t, and let $p = \frac{t}{S}$. Also, let $d(i)$ be the number of rows of the same type as row i. We bound V_0 by 1 trivially, and we assume henceforth that $p > 0$.

By definition, V_t is the probability of a mistake occurring when a randomly selected unknown entry is presented, given that all other entries are known with probability p. Since each entry (i, j) is presented next with probability $\frac{1}{S}$, it follows that

$$V_t = \frac{1}{S} \sum_{i,j} R_{ij},$$

where R_{ij} is the probability of a mistake occurring, given that entry (i, j) is unknown and is presented next.

Let I_{ij} be the random variable describing the set of rows consistent with row i and known in column j, and let J_{ij} be the random variable describing the set of rows i' in I_{ij} for which $M_{ij} \neq M_{i'j}$. If I_{ij} is nonempty, then the probability of choosing a row i' for which $M_{ij} \neq M_{i'j}$ is clearly $|J_{ij}| \,/\, |I_{ij}|$. Thus the probability of a mistake is just the expected value of this fraction, if it is assumed that $I_{ij} \neq \emptyset$.

Unfortunately, expectations of fractions are often hard to deal with. To handle this situation we therefore place a probabilistic lower bound on the denominator of this ratio, i.e., on $|I_{ij}|$. Note that if i and i' are of the same type, then the probability that $i' \in I_{ij}$ is just the chance p that (i', j) is known. Since there are $d(i)$ rows of type i (including i itself), we see that $\Pr[|I_{ij}| < y]$ is at most the chance that fewer than y of the other $d(i) - 1$ rows of the same type as i are in I_{ij}. In other words, this probability is bounded by the chance of fewer than y successes in a sequence of $d(i) - 1$ Bernoulli trials, each succeeding with probability p.

We use the following form of Chernoff bounds, due to Angluin and Valiant [3], to bound this probability:

LEMMA 3.17. *Consider a sequence of m independent Bernoulli trials, each succeeding with probability p. Let S be the random variable describing the total number of successes. Then for $0 \leq \gamma \leq 1$ the following hold:*

$$\Pr[S < (1-\gamma)mp] \leq e^{-\gamma^2 mp/2},$$

and

$$\Pr[S > (1+\gamma)mp] \leq e^{-\gamma^2 mp/3}.$$

Thus by letting $y = p(d(i)-1)/2$ and applying this lemma, it follows that

$$\Pr[|I_{ij}| < p(d(i)-1)/2] \leq e^{-p(d(i)-1)/8}.$$

Note that this bound applies even if $d(i) = 1$.

Thus we have

$$\begin{aligned} R_{ij} &\leq \Pr[|I_{ij}| < y] + \mathrm{E}\left[\frac{|J_{ij}|}{|I_{ij}|} \mid |I_{ij}| \geq y\right] \cdot \Pr[|I_{ij}| > y] \\ &\leq \Pr[|I_{ij}| < y] + \frac{\mathrm{E}[|J_{ij}| \mid |I_{ij}| \geq y]}{y} \cdot \Pr[|I_{ij}| \geq y] \\ &\leq \Pr[|I_{ij}| < y] + \frac{\mathrm{E}[|J_{ij}|]}{y}. \end{aligned}$$

So to bound R_{ij} it will be useful to bound $\mathrm{E}[|J_{ij}|]$.

We have

$$\mathrm{E}[|J_{ij}|] = \sum_{i' \neq i, M_{i'j} \neq M_{ij}} \Pr[i' \in I_{ij}].$$

If $M_{ij} \neq M_{i'j}$, then $\Pr[i' \in I_{ij}]$ is the chance that (i', j) is known and that i and i' are consistent. Entry (i', j) is known with probability p, and i and i' are consistent if either (i, j') or (i', j') is unknown for each column $j' \neq j$ in which i and i' differ. If $h(i, i')$ is the Hamming distance between rows i and i', then this probability is $(1-p^2)^{h(i,i')-1}$.

Combining these facts, we have

$$\begin{aligned} V_t &\leq \frac{1}{S}\sum_i \sum_j e^{-p(d(i)-1)/8} + \frac{1}{S}\sum_{d(i)>1}\sum_j \frac{\sum_{i' \neq i, M_{i'j} \neq M_{ij}} p(1-p^2)^{h(i,i')-1}}{p(d(i)-1)/2} \\ &= \frac{1}{n}\sum_i e^{-p(d(i)-1)/8} + \frac{1}{S}\sum_{d(i)>1}\sum_{i' \neq i} \frac{2h(i,i')(1-p^2)^{h(i,i')-1}}{d(i)-1}. \end{aligned}$$

Recall that our goal is to upper bound the sum $\sum_{t=0}^{S-1} V_t$. Applying the above upper bound for V_t, we get

$$\begin{aligned} \sum_{t=0}^{S-1} V_t \leq &\sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_i e^{-(t/S)(d(i)-1)/8}\right) \\ &+ \sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{d(i)>1}\sum_{i' \neq i}\frac{2h(i,i')}{d(i)-1}\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}\right). \end{aligned} \tag{1}$$

We now bound the first part of the above expression. We begin by noting that

$$\sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_{i=1}^{n} e^{-(t/S)(d(i)-1)/8}\right) \leq \frac{1}{n}\sum_{i=1}^{n}\left(1+\int_0^S e^{-(t/S)(d(i)-1)/8}dt\right)$$
$$\leq \frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{16S}{d(i)}\right),$$

where this last bound follows by evaluating the integral in the two cases that $d(i) = 1$ and $d(i) > 1$. This last expression equals

$$1 + 16m\sum_{i=1}^{n}\frac{1}{d(i)} = 16km + 1,$$

where the last step is obtained by rewriting the summation to go over all the row types: There are $d(i)$ terms for rows of the same type as row i; thus each row type contributes 1 to the summation.

We next bound the second part of expression (1). To complete the proof of the lemma it suffices to show that

$$\sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}\right) = O(nk\sqrt{H}).$$

We begin by noting that this expression is bounded above by

$$\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt\right).$$

If $h(i,i') = 1$, then this integral is trivially evaluated to be S. Otherwise, by applying the inequality $e^x \geq 1 + x$ we get

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \int_0^S \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt. \tag{2}$$

A standard integral table [13] gives

$$\int_0^\infty \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt = \frac{S\sqrt{\pi}}{2\sqrt{h(i,i')-1}}. \tag{3}$$

Combining these bounds, we have

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \frac{S\sqrt{\pi}}{\sqrt{2h(i,i')}} \tag{4}$$

for $h(i,i') \geq 1$. Thus we arrive at an upper bound of

$$\frac{1}{S}\sum_{d(i)>1}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt\right)$$

$$\begin{aligned}
&\le \frac{1}{S}\sum_i \sum_{i'\neq i} \frac{2h(i,i')}{d(i)}\left(1+\frac{S\sqrt{\pi}}{\sqrt{2h(i,i')}}\right)\\
&\le \frac{2m}{S}\sum_i \sum_{i'\neq i}\frac{1}{d(i)} + \sqrt{2\pi}\sum_i\sum_{i'\neq i}\frac{\sqrt{h(i,i')}}{d(i)}\\
&= O(nk\sqrt{H}).
\end{aligned}$$

This implies the desired bound. □

To complete the theorem we prove the main result of this section, namely, that the upper bound obtained under this simple probabilistic model holds (to within a constant factor) for the true model. In other words, to compute an upper bound on the number of mistakes made by a prediction algorithm when the instances are selected according to a uniform distribution on the instance space, one can replace the requirement that exactly t matrix entries are known by the requirement that each matrix entry is known with probability $\frac{t}{nm}$.

LEMMA 3.18. $\sum_{t=0}^{S-1} U_t = O\left(\sum_{t=0}^{S-1} V_t\right)$.

Proof. We first note that

$$V_t = \sum_{r=0}^{S-1}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r} U_r.$$

To see this, observe that for each r, where r is the number of known entries, we need only multiply U_r by the probability that exactly r entries are known if it is assumed that each entry is known with probability of $\frac{t}{S}$. Therefore,

$$(5)\qquad \begin{aligned}
\sum_{t=0}^{S-1} V_t &= \sum_{t=0}^{S-1}\sum_{r=0}^{S-1} U_r\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r}\\
&= \sum_{r=0}^{S-1} U_r\left[\sum_{t=0}^{S-1}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r}\right].
\end{aligned}$$

Thus to prove the lemma it suffices to show that the inner summation is bounded below by a positive constant. By symmetry assume that $r \le \frac{S}{2}$ and let $y = S-r$. Stirling's approximation implies that

$$\binom{S}{r} = \Theta\left(\frac{S^S}{r^r y^y}\sqrt{\frac{S}{ry}}\right).$$

Applying this formula to the desired summation, we obtain that

$$(6)\qquad \begin{aligned}
\sum_{t=0}^{S-1}\binom{S}{r}\left(\frac{t}{S}\right)^r\left(1-\frac{t}{S}\right)^{S-r} &= \Theta\left(\sqrt{\frac{S}{ry}}\sum_{t=0}^{S}\left(\frac{t}{r}\right)^r\left(\frac{S-t}{y}\right)^y\right)\\
&= \Omega\left(\sqrt{\frac{S}{ry}}\sum_{x=1}^{\sqrt{ry/S}}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y\right).
\end{aligned}$$

The last step above follows by letting $x = t-r$ and reducing the limits of the summation. To complete the proof that equation (6) is bounded below by a positive constant we need prove only that

$$\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y=\Omega(1)$$

for all $1\le x\le\sqrt{ry/S}$.

By using the inequality $1+x\le e^x$ it can be shown that for $1+y>0$, $1+y\ge e^{y/(1+y)}$. We apply this observation to get that

$$\begin{aligned}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y&=\left(1+\frac{x}{r}\right)^r\left(1-\frac{x}{y}\right)^y\\&\ge\exp\left\{\frac{x}{1+\frac{x}{r}}-\frac{x}{1-\frac{x}{y}}\right\}\\&=\exp\left\{\frac{rx}{r+x}-\frac{yx}{y-x}\right\}\\&=\exp\left\{\frac{-x^2(r+y)}{(r+x)(y-x)}\right\}\\&=\exp\left\{\frac{-Sx^2}{(r+x)(y-x)}\right\}.\end{aligned}$$

Since $x\le\sqrt{ry/S}$, it follows that $Sx^2\le ry$. By applying this observation to the above inequality it follows that

$$\begin{aligned}\left(\frac{r+x}{r}\right)^r\left(\frac{y-x}{y}\right)^y&\ge\exp\left\{\frac{-ry}{(r+x)(y-x)}\right\}\\&=\exp\left\{\frac{-ry}{ry+(y-r)x-x^2}\right\}\\&\ge\exp\left\{\frac{-ry}{ry-\frac{ry}{S}}\right\}\\&=\exp\left\{\frac{-1}{1-\frac{1}{S}}\right\}.\end{aligned}$$

Finally, we note that for $S\ge 2$, $e^{-1/(1-1/S)}\ge e^{-2}$. This completes the proof of the lemma. □

Clearly, Lemma 3.16 and Lemma 3.18 together imply that $\sum_{t=0}^{S-1}U_t=O(km+nk\sqrt{H})$, giving the desired result. □

This completes our discussion of learning k-binary-relations.

4. Learning a total order. In this section we present our results for learning a binary relation on a set where it is known a priori that the relation forms a total order. One can view this problem as that of learning a total order on a set of n objects where an instance corresponds to comparing which of two objects is greater in the target total order. Thus this problem is like comparison-based sorting, except for two key differences: We vary the agent selecting the order in which comparisons are made (in sorting the learner does the selection), and we charge the learner only for *incorrectly predicted* comparisons.

Before describing our results, we motivate this section with the following example. There are n basketball teams that are competing in a round-robin tournament. That is, each team will play all other teams exactly once. Furthermore, we make the (admittedly simplistic) assumption that there is a ranking of the teams such that a team wins its match if and only if its opponent is ranked below it. A gambler wants to place a \$10 bet on each game: if he bets on the winning team he wins \$10 and if he bets on the losing team he loses \$10. Of course, his goal is to win as many bets as possible.

We formalize the problem of learning a total order as follows. The instance space $X_n = \{1, \ldots, n\} \times \{1, \ldots, n\}$. An instance (i, j) in X_n is in the target concept if and only if object i precedes object j in the corresponding total order.

If computation time is not a concern, then the halving algorithm makes at most $n \lg n$ mistakes. However, we are interested in efficient algorithms, and thus our goal is to design an efficient version of the halving algorithm. In the next section we discuss the relation between the halving algorithm and approximate counting. Then we show how to use an approximate counting scheme to implement a randomized version of the approximate halving algorithm, and we apply this result to the problem of learning a total order on a set of n elements. Finally, we discuss how a majority algorithm can be used to implement a counting algorithm.

4.1. The halving algorithm and approximate counting. In this section we review the halving algorithm and approximate counting schemes. We first cover the halving algorithm [4], [19]. Let $\mathcal{V}$ denote the set of concepts in C_n that are consistent with the feedback from all previous queries. Given an instance x in X_n, for each concept in $\mathcal{V}$ the halving algorithm computes the prediction of that concept for x and predicts according to the majority. Finally, all concepts in $\mathcal{V}$ that are inconsistent with the correct classification are deleted. Littlestone [19] shows that this algorithm makes at most $\lg|C_n|$ mistakes. Now suppose the prediction algorithm predicts according to the majority of concepts in set $\mathcal{V}'$, the set of all concepts in C_n consistent with all *incorrectly* predicted instances. Littlestone [19] also proves that this *space-efficient halving algorithm* makes at most $\lg|C_n|$ mistakes.

We define an *approximate halving algorithm* to be the following generalization of the halving algorithm. Given instance x in X_n, an approximate halving algorithm predicts in agreement with at least $\varphi|\mathcal{V}|$ of the concepts in $\mathcal{V}$ for some constant $0 < \varphi \leq \frac{1}{2}$.

THEOREM 4.1. *An approximate halving algorithm makes at most* $\log_{(1-\varphi)^{-1}} |C_n|$ *mistakes for learning* C_n.

Proof. Each time a mistake is made, the number of concepts that remain in $\mathcal{V}$ are reduced by a factor of at least $1 - \varphi$. Thus after at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes there is only one consistent concept left in C_n. □

We note that the above result holds also for the space-efficient version of the approximate halving algorithm.

When we are given an instance $x \in X_n$, one way to predict as dictated by the halving algorithm is to count the number of concepts in $\mathcal{V}$ for which $c(x) = 0$ and for which $c(x) = 1$ and then to predict with the majority. As we shall see, by using these ideas we can use an approximate counting scheme to implement the approximate halving algorithm.

We now introduce the notion of an approximate counting scheme for counting the number of elements in a finite set $\mathcal{S}$. Let x be a description of a set $\mathcal{S}_x$ in some natural encoding. An *exact counting scheme* on input x outputs $|\mathcal{S}_x|$ with probability 1. Such a scheme is polynomial if it runs in time polynomial in $|x|$. Sometimes exact counting can be accomplished in polynomial time; however, many counting problems are $\#\mathcal{P}$-complete and thus are assumed to be intractable. (For a discussion of the class $\#\mathcal{P}$ see Valiant [26].) For many $\#\mathcal{P}$-complete problems good approximations are possible [16], [24], [25]. A *randomized approximation scheme* R for a counting problem satisfies the following condition for all $\epsilon, \delta > 0$:

$$\Pr\left[\frac{|S_x|}{(1+\epsilon)} \le R(x,\epsilon,\delta) \le |S_x|(1+\epsilon)\right] \ge 1-\delta,$$

where $R(x,\epsilon,\delta)$ is R's estimate on input x, ϵ, and δ. In other words, with high probability, R estimates $|S_x|$ within a factor of $1+\epsilon$. Such a scheme is *fully polynomial* if it runs in time polynomial in $|x|$, $\frac{1}{\epsilon}$, and $\lg \frac{1}{\delta}$. For further discussion see Sinclair [24].

We now review work on counting the number of linear extensions of a total order. That is, given a partial order on a set of n elements, the goal is to compute the number of total orders that are linear extensions of the given partial order. We discuss the relationship between this problem and that of computing the volume of a convex polyhedron. (For more details on this subject, see Lovász [21, §2.4].) Given a convex set S and an element a of $\Re^n$, a *weak separation oracle* either (i) asserts that $a \in S$ or (ii) asserts that $a \notin S$ and supplies a reason why. In particular, for closed convex sets in $\Re^n$, if $a \notin S$, then there exists a hyperplane separating a from S. So if $a \notin S$, the oracle responds with such a separating hyperplane as the reason why $a \notin S$.

We now discuss how to reduce the problem of counting the number of extensions of a partial order on n elements to that of computing the volume of a convex n-dimensional polyhedron given by a separation oracle. The polyhedron built in the reduction will be a subset of $[0, 1]^n$ (i.e., the unit hypercube in $\Re^n$), where each dimension corresponds to one of the n elements. Observe that any inequality $x_i > x_j$ defines a half-space in $[0, 1]^n$. Let $\Delta(t)$ denote the polyhedron obtained by taking the *intersection* of the half-spaces given by the inequalities of the partial order t. (See Fig. 8 for an example with $n = 3$.) For any pair

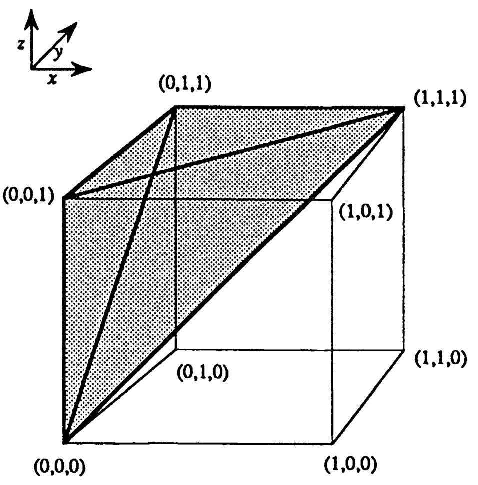


FIG. 8. *Polyhedron formed by the total order $z > y > x$.*

of total orders t_1 and t_2 the polyhedra $\Delta(t_1)$ and $\Delta(t_2)$ are simplices that intersect only in a face (zero volume): A pair of elements, say, x_i and x_j, that are ordered differently in t_1 and t_2 (such a pair must exist) define a hyperplane $x_i = x_j$ that separates $\Delta(t_1)$ and $\Delta(t_2)$. Let T_n be the set of all $n!$ total orders on n elements. Then

$$[0, 1]^n = \bigcup_{t \in T_n} \Delta(t). \tag{7}$$

In other words, the union of the polyhedra associated with all total orders yields the unit hypercube. We have already seen that polyhedra associated with the $t \in T_n$ are disjoint. To see that they cover all of $[0, 1]^n$ observe that any point $y \in [0, 1]^n$ defines some total order t, and clearly $y \in \Delta(t)$. Let P be a partial order on a set of n elements. From equation (7) and the observation that the volumes of the polyhedra formed by each total order are equal, it follows that the volume of the polyhedron defined by any total order is $\frac{1}{n!}$. Thus it follows that for any partial order P

$$\frac{\text{number of extensions of } P}{n!} = \text{volume of } \Delta(P). \tag{8}$$

Rewriting equation (8), we obtain that

$$\text{number of extensions of } P = n! \cdot (\text{volume of } \Delta(P)). \tag{9}$$

Finally, we note that the weak separation oracle is easy to implement for any partial order. Given inputs a and S, it just checks each inequality of the partial order to see whether a is in the convex polyhedron S. If a does not satisfy some inequality, then it replies that $a \notin S$ and returns that inequality as the separating hyperplane. Otherwise, if a satisfies all inequalities, it replies that $a \in S$.

Dyer, Frieze, and Kannan [10] give a fully polynomial randomized approximation scheme (fpras) for approximating the volume of a polyhedron, given a weak separation oracle. From equation (9) we see that this fpras for estimating the volume of a polyhedron can be easily applied to estimating the number of extensions of a partial order. Furthermore, Dyer and Frieze [9] prove that it is $\#\mathcal{P}$-hard to exactly compute the volume of a polyhedron given either by a list of its facets or its vertices.

Independently, Matthews [22] has described an algorithm to generate a random linear extension of a partial order. Consider the convex polyhedron K defined by the partial order. Matthew's main result is a technique to sample nearly uniformly from K. Given such a procedure to sample uniformly from K, one can sample uniformly from the set of extensions of a partial order by choosing a random point in K and then selecting the total order corresponding to the ordering of the coordinates of the selected point. A procedure to generate a random linear extension of a partial order can then be used repeatedly to approximate the number of linear extensions of a partial order [22].

4.2. Application to learning. We begin this section by studying the problem of learning a total order under teacher-directed and self-directed learning. Then we show how to use an fpras to implement a randomized version of the approximate halving algorithm, and we apply this result to the problem of learning a total order on a set of n elements.

Under the teacher-selected query sequence we obtain an $n-1$ mistake bound. The teacher can uniquely specify the target total order by giving the $n-1$ instances that correspond to consecutive elements in the target total order. Since $n-1$ instances are needed to uniquely specify a total order, we get a matching lower bound. Winkler [28] has shown that under the learner-selected query sequence, one can also obtain an $n-1$ mistake bound. To achieve this bound the learner uses an insertion sort, as described, for instance, by Cormen, Leiserson, and Rivest [8], where for each new element the learner guesses it is smaller than each of the ordered elements (starting with the largest) until a mistake is made. When a mistake occurs, this new element is properly positioned in the chain. Thus at most $n-1$ mistakes will be made by the learner. In fact, the learner can be forced to make at least $n-1$ mistakes. The adversary gives feedback by using the following simple strategy: The first time an object is involved in a comparison, reply that the learner's prediction is wrong. In doing so, one creates a set of *chains*, where a chain is a total order on a subset of the elements. If c chains are created by this process, then the learner has made $n - c$ mistakes. Since all these chains must be combined to get a total order, the adversary can force $c - 1$ additional mistakes by always replying that a mistake occurs the first time that elements from two different chains are compared. (It is not hard to see that the above steps can be interleaved.) Thus the adversary can force $n - 1$ mistakes.

Next we consider the case in which an adversary selects the query sequence. We first prove an $\Omega(n \lg n)$ lower bound on the number of mistakes made by any prediction algorithm. We use the following result of Kahn and Saks [17]: Given any partial order P that is not a total order there exists an incomparable pair of elements x_i, x_j such that

$$\frac{3}{11} \le \frac{\text{number of extensions of } P \text{ with } x_i \le x_j}{\text{number of extensions of } P} \le \frac{8}{11}.$$

So the adversary can always pick a pair of elements, so that regardless of the learner's prediction, the adversary can report that a mistake was made while only eliminating a constant fraction of the remaining total orders.

Finally, we present a polynomial prediction algorithm making $n \lg n + (\lg e) \lg n$ mistakes with very high probability. We first show how to use an exact counting algorithm R, for counting the number of concepts in C_n consistent with a given set of examples, to implement the halving algorithm.

LEMMA 4.2. *Given a polynomial algorithm R to exactly count the number of concepts in C_n consistent with a given set E of examples, one can construct an efficient implementation of the halving algorithm for C_n.*

Proof. We show how to use R to efficiently make the predictions required by the halving algorithm. To make a prediction for an instance x in X_n the following procedure is used: Construct E^- from E by appending x as a negative example to E. Use the counting algorithm R to count the number of concepts $C^- \in \mathcal{V}$ that are consistent with E^-. Next, construct E^+ from E by appending x as a positive example to E. As before, use R to count the number of concepts $C^+ \in \mathcal{V}$ that are consistent with E^+. Finally, if $|C^-| \ge |C^+|$, then predict that x is a negative example; otherwise, predict that x is a positive example.

Clearly, a prediction is made in polynomial time, since it only requires calling R twice. It is also clear that each prediction is made according to the majority of concepts in $\mathcal{V}$. □

We modify this basic technique to use an fpras instead of the exact counting algorithm to obtain an efficient implementation of a randomized version of the approximate halving algorithm. In doing so, we obtain the following general theorem describing when the existence of an fpras leads to a good prediction algorithm. We then apply this theorem to the problem of learning a total order.

THEOREM 4.3. *Let R be an fpras for counting the number of concepts in C_n consistent with a given set E of examples. If $|X_n|$ is polynomial in n, one can produce a prediction algorithm that for any $\delta > 0$ runs in time polynomial in n and $\lg \frac{1}{\delta}$ and makes at most $\lg |C_n|(1 + \frac{\lg e}{n})$ mistakes with probability at least $1 - \delta$.*

Proof. The prediction algorithm implements the procedure described in Lemma 4.2 with the exact counting algorithm replaced by the fpras $R(n, \frac{1}{n}, \frac{\delta}{2|X_n|})$. Consider the prediction for an instance $x \in X_n$. Let $\mathcal{V}$ be the set of concepts that are consistent with all previous instances. Let r^+ (respectively, r^-) be the number of concepts in $\mathcal{V}$ for which x is a positive (negative) instance. Let $\hat{r}^+$ (respectively, $\hat{r}^-$) be the estimate output by R for r^+ (r^-). Since R is an fpras, with probability at least $1 - \frac{\delta}{|X_n|}$

$$\frac{r^-}{1+\epsilon} \le \hat{r}^- \le (1+\epsilon) r^- \quad \text{and} \quad \frac{r^+}{1+\epsilon} \le \hat{r}^+ \le (1+\epsilon) r^+,$$

where $\epsilon = \frac{1}{n}$. Without loss of generality assume that the algorithm predicts that x is a negative instance and thus $\hat{r}^- \geq \hat{r}^+$. Combining the above inequalities and the observation that $r^- + r^+ = |V|$, we obtain that $r^- \geq \frac{|V|}{1+(1+\epsilon)^2}$.

We define an *appropriate* prediction to be a prediction that agrees with *at least* $\frac{|V|}{1+(1+\epsilon)^2}$ of the concepts in V. To analyze the mistake bound for this algorithm we suppose that each prediction is appropriate. For a single prediction to be appropriate, both calls to the fpras R must output a count that is within a factor of $1+\epsilon$ of the true count. So any given prediction is appropriate with probability at least $1 - \frac{\delta}{|X_n|}$, and thus the probability that all predictions are appropriate is at least

$$1 - |X_n|\left(\frac{\delta}{|X_n|}\right) = 1 - \delta.$$

Clearly, if all predictions are appropriate, then the above procedure is in fact an implementation of the approximate halving algorithm with $\varphi = \frac{1}{1+(1+\epsilon)^2}$, and thus by Theorem 4.1 at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes are made. Substituting ϵ with its value of $\frac{1}{n}$ and simplifying the expression, we obtain that with probability at least $1-\delta$

$$\text{(10)} \qquad \text{number of mistakes} \leq \frac{\lg |C_n|}{\lg \frac{1}{1-\varphi}} = \frac{\lg |C_n|}{\lg\left(1 + \frac{n^2}{n^2+2n+1}\right)}.$$

Since $\frac{n^2}{n^2+2n+1} \geq 1 - \frac{2}{n}$,

$$\begin{aligned} \frac{1}{\lg\left(1+\frac{n^2}{n^2+2n+1}\right)} &\leq \frac{1}{\lg\left(1+1-\frac{2}{n}\right)} \\ &= \frac{1}{1+\lg\left(1-\frac{1}{n}\right)} \\ &= 1 - \frac{\lg\left(1-\frac{1}{n}\right)}{1+\lg\left(1-\frac{1}{n}\right)}. \end{aligned}$$

By applying the inequalities $\lg\left(1-\frac{1}{n}\right) \geq \frac{-\lg e}{n-1}$ and $1+\lg\left(1-\frac{1}{n}\right) \leq 1 - \frac{\lg e}{n}$ it follows that

$$\begin{aligned} \frac{\lg\left(1-\frac{1}{n}\right)}{1+\lg\left(1-\frac{1}{n}\right)} &\geq \frac{\frac{-\lg e}{n-1}}{1-\frac{\lg e}{n}} \\ &= \frac{-\lg e}{n-1-\frac{n-1}{n}\lg e} \\ &\geq \frac{-\lg e}{n}. \end{aligned}$$

Finally, applying these inequalities to equation (10) yields that

$$\text{number of mistakes} \leq \frac{\lg |C_n|}{\lg\left(1+\frac{n^2}{n^2+2n+1}\right)} \leq \lg |C_n| \left(1 + \frac{\lg e}{n}\right). \qquad \square$$

Note that we could modify the above proof by not requiring that all predictions be appropriate. In particular, if we allow γ predictions not to be appropriate, then we get a mistake bound of $\lg |C_n| \left(1 + \frac{\lg e}{n}\right) + \gamma$.

We now apply this result to obtain the main result of this section. Namely, we describe a randomized polynomial prediction algorithm for learning a total order in the case in which the adversary selects the query sequence.

THEOREM 4.4. *There exists a prediction algorithm A for learning total orders such that on input δ (for all $\delta > 0$), and for any query sequence provided by the adversary, A runs in time polynomial in n and $\lg \frac{1}{\delta}$ and makes at most $n \lg n + (\lg e) \lg n$ mistakes with probability at least $1 - \delta$.*

Proof. We apply the results of Theorem 4.3 by using the fpras for counting the number of extensions of a partial order given independently by Dyer, Frieze, and Kannan [10] and by Matthews [22]. We know that with probability at least $1 - \delta$ the number of mistakes is at most $\lg |C_n|(1 + \frac{\lg e}{n})$. Since $|C_n| = n!$, the desired result is obtained. □

We note that the probability that A makes more than $n \lg n + (\lg e) \lg n$ mistakes does not depend on the query sequence selected by the adversary. The probability is taken over the coin flips of the randomized approximation scheme.

Thus, as in learning a k-binary-relation by using a row-filter algorithm, we see that a learner can do asymptotically better with self-directed learning than with adversary-directed learning. Furthermore, whereas the self-directed learning algorithm is deterministic, here the adversary-directed algorithm is randomized.

As a final note, observe that we have just seen how a counting algorithm can be used to implement the halving algorithm. In her thesis Goldman [11] has described conditions under which the halving algorithm can be used to implement a counting algorithm.

5. Conclusions and open problems.

We have formalized and studied the problem of learning a binary relation between two sets of objects and between a set and itself under an extension of the on-line learning model. We have presented general techniques to help develop efficient versions of the halving algorithm. In particular, we have shown how a fully polynomial randomized approximation scheme can be used to efficiently implement a randomized version of the approximate halving algorithm. We have also extended the mistake bound model by adding the notion of an instance selector. The specific results are summarized in Table 2. In this table all lower bounds are information-theoretic bounds and all upper bounds are for polynomial-time learning algorithms. Also, unless otherwise stated, the results listed are for deterministic learning algorithms.

From Table 2 one can see that several of the above bounds are tight and several others are asymptotically tight. However, for the problem of learning a k-binary-relation there is a gap in the bound for the random and adversary (except $k \leq 2$) directors. Note that the bounds for row-filter algorithms are asymptotically tight for k constant. Clearly, if we want asymptotically tight bounds that include a dependence on k, we cannot use only two row types in the matrix used for the projective geometry lower bound.[5]

For the problem of learning a total order all the above bounds are tight or asymptotically tight. Although the fully polynomial randomized approximation scheme for approximating the number of extensions of a partial order is a polynomial-time algorithm, the exponent on n is somewhat large and the algorithm is quite complicated. Thus an interesting problem is to find a practical prediction algorithm for the problem of learning a total order. Another interesting direction of research is to explore other ways of modeling the structure in a binary relation. Finally, we hope to find other applications of fully polynomial randomized approximation schemes to learning theory.

[5]Chen [7] has recently extended the projective geometry argument to obtain a lower bound of $\Omega(n\sqrt{m \lg k})$ for $m \geq n \lg k$.

TABLE 2
Summary of results.

Concept class	Director	Lower bound	Upper bound	Notes
Binary relation (k row types)	Learner	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$km + (n - k)\lfloor \lg k \rfloor$	
	Teacher	$km + (n - k)(k - 1)$	$km + (n - k)(k - 1)$	
	Adversary	$km + (n - k)\lfloor \lg k \rfloor$	$O(km + n\sqrt{m \lg k})$	Due to M. Warmuth[a]
	Adversary	$2m + n - 2$	$2m + n - 2$	$k = 2$
	Adversary	$\Omega(km + (n-k)\lg k + \min\{n\sqrt{m}, m\sqrt{n}\})$	$km + n\sqrt{(k-1)m}$	Row-filter algorithm
	Uniform dist.	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$O(km + nk\sqrt{H})$	Avg. case, row-filter alg.
Total order	Teacher	$n - 1$	$n - 1$	
	Learner	$n - 1$	$n - 1$	Due fo P. Winkler
	Adversary	$\Omega(n \lg n)$	$n \lg n + (\lg e) \lg n$	Randomized algorithm

[a]Note that if computation time is not a concern, we have shown that the halving algorithm makes at most $km + (n - k) \lg k$ mistakes.

Acknowledgments. The results in §4 were inspired by an open problems session led by Manfred Warmuth at our weekly Machine Learning Reading Group meeting, where he proposed the basic idea of using an approximate halving algorithm based on approximate and probabilistic counting and also suggested the problem of learning a total order on n elements. We thank Tom Leighton for helping to improve the lower bound of Theorem 3.13. We also thank Nick Littlestone, Bob Sloan, and Ken Goldman for their comments.

REFERENCES

[1] D. ANGLUIN, *Learning regular sets from queries and counterexamples*, Inform. and Comput., 75 (1987), pp. 87–106.
[2] ———, *Queries and concept learning*, Mach. Learning, 2 (1988), pp. 319–342.
[3] D. ANGLUIN AND L. G. VALIANT, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.
[4] J. BARZDIN AND R. FREIVALD, *On the prediction of general recursive functions*, Soviet Math. Dok., 13 (1972), pp. 1224-1228.
[5] A. BLUM, *An $\tilde{O}(n^{0.4})$-approximation algorithm for* 3*-coloring*, in Procceedings of the Twenth First Annual ACM Symposium on Theory of Computing, 1989, pp. 535–542.
[6] R. CARMICHAEL, *Introduction to the Theory of Groups of Finite Order*, Dover, New York, 1937.
[7] W. CHEN, personal communication, 1991.
[8] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press/McGraw-Hill, Cambridge, MA, 1990.
[9] M. DYER AND A. FRIEZE, *On the complexity of computing the volume of a polyhedron*, SIAM J. Comput., 17 (1988), pp. 967–974.
[10] M. DYER, A. FRIEZE, AND R. KANNAN, *A random polynomial-time algorithm for approximating the volume of convex bodies*, J. Assoc. Comput. Mach., 38 (1991), pp. 1–17.
[11] S. A. GOLDMAN, *Learning Binary Relations, Total Orders, and Read-once Formulas*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1990.
[12] S. A. GOLDMAN AND M. K. WARMUTH, *Learning binary relations with weighted majority voting*, Proc. of the Sixth Annual Workshop on Computational Learning Theory, July 1993, to appear.
[13] T. GRADSHTEYN AND I. RYZHIK, *Tables of Integral, Series, and Products*, Academic Press, New York, 1980; corrected and enlarged edition by A. Jeffrey.

[14] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth, *Equivalence of models for polynomial learnability*, Information and Computation, 95 (1991), pp. 129–161.

[15] D. Haussler, N. Littlestone, and M. Warmuth, *Expected mistake bounds for on-line learning algorithms*, unpublished manuscript, 1988.

[16] M. Jerrum and A. Sinclair, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[17] J. Kahn and M. Saks, *Balancing poset extensions*, Order, 1 (1984), pp. 113–126.

[18] N. Linial and U. Vazirani, *Graph products and chromatic numbers*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1989, pp. 124–128.

[19] N. Littlestone, *Learning when irrelevant attributes abound: A new linear-threshold algorithm*, Mach. Learning, 2 (1988), pp. 285–318.

[20] N. Littlestone and M. K. Warmuth, *The weighted majority Algorithm*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1989, pp. 256–261. To appear in Information and Computation.

[21] L. Lovász, *An algorithmic theory of numbers, graphs and convexity*, in CBMS–NSF Regional Conference Series on Applied Mathematics, Philadelphia, PA., 1986.

[22] P. Matthews, *Generating a random linear extension of a partial order*, Ann. Prob., 19 (1991), pp. 1367–1392.

[23] R. L. Rivest and P. Sloan, *Learning complicated concepts reliability and usefully*, in Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann, San Mateo, CA, 1988, pp. 69–79.

[24] A. Sinclair, *Randomised Algorithms for Counting and Generating Combinatorial Structures*, Ph.D. thesis, Department of Computer Science, University of Edinburgh, Edinburgh, Scotland, U.K., 1988.

[25] L. Stockmeyer, *An approximation algorithm for # P*, SIAM J. Comput., 14 (1985), pp. 849–861.

[26] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 198–201.

[27] ———, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[28] P. Winkler, personal communication, 1989.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 1035–1052, October 1993

DRAWING GRAPHS IN THE PLANE WITH HIGH RESOLUTION*

M. FORMANN†, T. HAGERUP‡, J. HARALAMBIDES§, M. KAUFMANN¶, F. T. LEIGHTON1, A. SYMVONIS2, E. WELZL†, AND G. WOEGINGER3

Abstract. This paper presents the problem of drawing a graph in the plane so that edges appear as straight lines and the minimum angle formed by any pair of incident edges is maximized. The *resolution of a layout* is defined to be the size of the minimum angle formed by incident edges of the graph, and the *resolution of a graph* to be the maximum resolution of any layout of the graph. The resolution R of a graph is characterized in terms of the maximum node degree d of the graph by proving that $\Omega(\frac{1}{d^2}) \le R \le \frac{2\pi}{d}$ for any graph. Moreover, it is proved that $R = \Theta(\frac{1}{d})$ for many graphs including planar graphs, complete graphs, hypercubes, multidimensional meshes and tori, and other special networks. It is also shown that the problem of deciding if $R = \frac{2\pi}{d}$ for a graph is NP-hard for $d = 4$, and by using a counting argument that $R = O(\frac{\log d}{d^2})$ for many graphs.

Key words. coloring, graph layout, resolution, square graph

AMS subject classifications. 05C15, 05C85, 68Q25, 68R10, 05C99

1. Introduction.

Graph layout problems have been extensively studied in a wide variety of contexts. Examples include both linear [12], [19] and planar [1], [2], [4], [7], [9], [10], [13], [15]–[18] layout problems. Typically, nodes are represented by distinct points to be embedded in a line or plane, and they are sometimes restricted to be grid points. (Alternatively, nodes are sometimes represented by line segments [13]). Edges are often constrained to be drawn as straight lines [4], [6], [9], [10], [13], [15] or as a contiguous set of line segments [1], [7], [16], [18] (e.g., when bends are allowed). The objective is to find a layout for a graph that minimizes some cost function, such as area [1], [7], [15], number of edge crossings [1], maximum edge length [1], [2], number of bends [1], [7], [13], [16], visual complexity [17], density [9], [10], and so on. Note that the above references on graph layout algorithms form a representative set, but are far from complete. In [3], a survey on graph layout problems is given. For a list of references on graph drawing algorithms, see the annotated bibliography by Eades and Tamassia [5]. In [14], the same cost function as in this paper was investigated heuristically for planar graphs.

In this paper, we consider straight line layouts of graphs in the plane. Specifically, we introduce a new cost function for such layouts, called *resolution*. We define the *resolution of a layout* of the graph to be the size of the minimum angle formed by any two edges incident to a common node (the angles formed by crossing edges are not taken into account). We define the *resolution of a graph* to be the maximum resolution of any (straight line) layout for the

*Received by the editors February 19, 1991; accepted for publication (in revised form) June 3, 1992.

†Institut für Informatik, Fachbereich Mathematik, Freie Universität Berlin, D-1000 Berlin 33, Germany. This author's research was partially supported by the ESPRIT II Basic Research Actions Program of the EC contract 3075 (project ALCOM).

‡Fachbereich Informatik, Universität des Saarlandes, D-6600 Saarbrücken, Germany. This author's research was partially supported by the ESPIRIT II Basic Research Actions Program of the EC contract 3075 (project ALCOM).

§Department of Computer Science, Barry University, Miami Shores, Florida 33161.

¶Fachbereich Informatik, Universität des Saarlandes, D-6600 Saarbrücken, Germany. This author's research was supported by the DFG, Sonderforschungsbereich 124, Teilprojekt B2 (VLSI-Entwurfsmethoden und Parallelität).

1Mathematics Department and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. This author's research was supported by Air Force Office of Scientific Research contract AFOSR-89-0721, Defense Advanced Research Projects Agency contract N00014-87-K-825, Army Research Office contract DAAL-03-86-K-0171, and the Clear Center at UTD.

2Basser Department of Computer Science, University of Sydney, Sydney N.S.W. 2006, Australia.

3Institut für Mathematík, Technical University of Graz, Kopernikusgasse 24, A-8010 Graz, Austria. This author's research was supported by the Fonds zur Förderung der Wissenschaftlichen Forschung (FWF), Project S32/01.

graph in the plane. For example, the resolution of K_3 (the 3-node complete graph) is $\frac{\pi}{3}$. Our objective in this paper is to find layouts for graphs with the highest possible resolution.

An obvious upper bound on the resolution of a graph with maximum degree d is $\frac{2\pi}{d}$. Of course, this bound is not tight for many graphs. (For example, $d = 2$ for K_3 but the resolution is $\frac{\pi}{3}$.) Unfortunately, we will show that the problem of determining whether or not a graph with maximum degree d has resolution $\frac{2\pi}{d}$ is NP-hard, at least in the case $d = 4$. Whether or not the resolution problem is in NP is still unknown. Determining the precise complexity of the problem is complicated by the fact that there are simple 11-node graphs (such as that shown in Fig. 1) for which there is a layout with resolution $\frac{\pi}{3} - \epsilon$ for any $\epsilon > 0$, but for which there is no layout with resolution $\frac{\pi}{3}$. These difficulties can be overcome by restricting the problem so that the nodes of the graph are required to be placed at distinct grid points in a grid of a fixed size (e.g., $N \times N$), in which case the resolution of any N-node graph becomes a well defined NP-complete problem. We will consider both grid-based and unrestricted layouts in this paper.

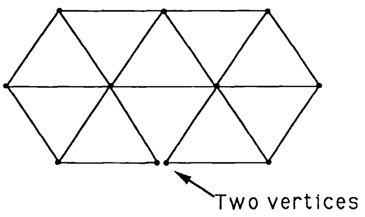


FIG. 1. *An 11-node graph for which there is a layout with resolution $\frac{\pi}{3} - \epsilon$ for any $\epsilon > 0$, but for which there is no layout with resolution $\frac{\pi}{3}$.*

On the positive side, we can prove nearly tight bounds on the resolution for many natural classes of graphs. For example, we prove that any planar graph with maximum degree d has resolution $\Theta(\frac{1}{d})$. We also prove similar bounds for special networks like the hypercube, torus, complete graph, and others. We construct a layout for an arbitrary graph with maximum degree d that has resolution $\Omega(\frac{1}{d^2})$. Hence, the resolution of any bounded-degree graph is constant, independent of the number of nodes in the graph. Whether or not there exists a family of graphs with maximum degree d and resolution $\Theta(\frac{1}{d^2})$ is still unknown. Natural candidates for graphs with low resolution such as the $(d^2 - d + 1)$-point projective plane and the $(\frac{d}{2} + 1) \times (\frac{d}{2} + 1)$ mesh of cliques (both of which are d-regular) have resolution $\Theta(\frac{1}{d})$. We do not even know of any simply constructible family of graphs with maximum degree d that has resolution $o(\frac{1}{d})$, although, using a counting argument, we can prove the existence of many graphs with maximum degree d and resolution $O(\frac{\log d}{d^2})$. Hence, the $\Omega(\frac{1}{d^2})$ worst-case lower bound is not to far from reality for many graphs.

Several of our constructions are based on the close relationship between the chromatic number of the square of a graph and its resolution. In particular, we will show that the resolution of any graph G is at least $\frac{\pi}{\chi(G^2)} - \epsilon$ for any $\epsilon > 0$, where $\chi(G^2)$ denotes the chromatic number of G^2. (The graph G^2 is formed from G by connecting pairs of nodes that are within distance 2 of each other in G.) Moreover, we will show that the resolution of any N-node graph G is $\Omega(\frac{1}{\chi(G^2)})$, even if we are restricted to position the nodes of G at distinct grid points of a square grid with area $O(\chi(G^2)^3 N)$. Hence, we can produce linear area layouts (in the sense of [1], [7], [18]) that have constant resolution for any bounded-degree graph.

Our $\Theta(\frac{1}{d})$ bound on the resolution of any planar graph with maximum degree d, in particular, stems from the fact that the square of any planar graph with maximum degree d has chromatic number $O(d)$. In this paper, we show an upper bound of $\frac{13}{7}d + O(d^{2/3})$, which is close to the best known existential lower bound of $\frac{3}{2}d$. (The lower bound is provided by the graph shown in Fig. 2.) The exact worst-case bound remains an unsolved problem.

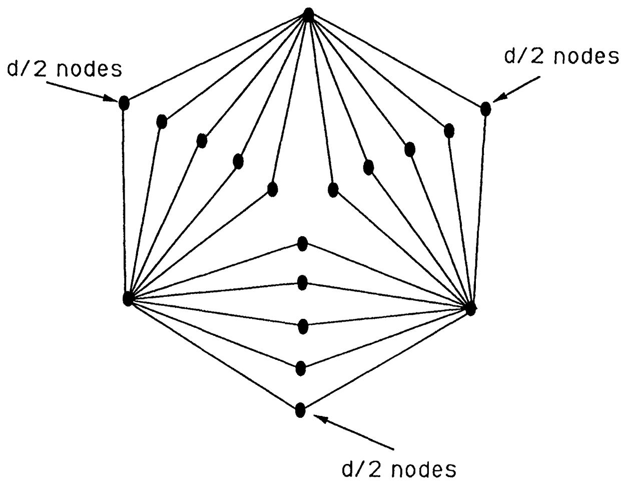


FIG. 2. *Example of a planar graph with maximum degree d for which $\chi(G^2) \geq \frac{3d}{2}$.*

Our interest in the problem of maximizing resolution stems largely from the fact that resolution seems to be a natural property of graphs that was previously (to our knowledge) unexplored. In addition, the resolution problem is related to problems that arise in network communication via optic beams [8], [11]. For example, consider a network in which each node represents a processor that can communicate via optical beams with its neighbors in the graph. By maximizing the resolution of the layout, we simplify the task of designing the processor and the task of recognizing one's neighbors. (It is hard to send or receive at very tight angles for a unit size processor.) Similar applications might arise in radio networks that make use of directional antennas.

The remainder of the paper is divided into sections as follows: In §2, we examine graphs of maximum degree 4 and prove that it is NP-hard to decide if we can draw them with resolution $\frac{\pi}{2}$. In §3, we first present an algorithm for general graph layouts. We then consider the case of planar graphs. Finally, because of their importance, we consider layouts of special networks. In §4, we present an upper bound on the resolution of random graphs. Section 5 contains some remarks and additional topics for research.

2. NP-hardness. Given a graph with maximum degree d, we know that its optimum embedding on the plane can have resolution at most $\frac{2\pi}{d}$. In what follows, we show that the problem of deciding whether a graph of maximum degree d has an embedding on the plane with resolution $\frac{2\pi}{d}$ is NP-hard in the case $d = 4$.

THEOREM 1. *Given a graph G of maximum degree 4, the decision problem of whether or not G can be embedded in the plane with resolution $\frac{\pi}{2}$ is NP-hard.*

Proof. The proof is done by a reduction from 3-SAT. Let S be a formula in 3-CNF, let $U = \{x_1, x_2, x_3, \ldots, x_n\}$ be the variables occurring in S, and let $C = \{c_1, c_2, c_3, \ldots, c_k\}$ be the clauses in S, such that every clause $c \in C$ consists of exactly three literals. We construct a graph G of maximum degree 4 that is embeddable with resolution $\frac{\pi}{2}$ if and only if S is satisfiable.

The skeleton of G is given in Figure 3(a). For each variable x_i, $1 \le i \le n$, there is a node in the skeleton. The same holds for each clause c_j, $1 \le j \le k$. Observe that up to reflections, rotations, and stretchings, the embedding of G with resolution $\frac{\pi}{2}$ (if there is any) essentially has to look like the one in Fig. 3(a). Now we append at each node x_i the tower of Fig. 3(b). For each such tower, there are two possible embeddings (in relation to the skeleton): The negated nodes to the left and the nonnegated nodes to the right, or vice versa. Finally, for each clause $c_j = \{x_i, x_k, x_l\}$, we connect the node c_j to the nodes $x_{i,j}$, $x_{k,j}$, and $x_{l,j}$ (or to the corresponding negated nodes, if the literals are negated) by a path consisting of three edges (Fig. 3(c)).

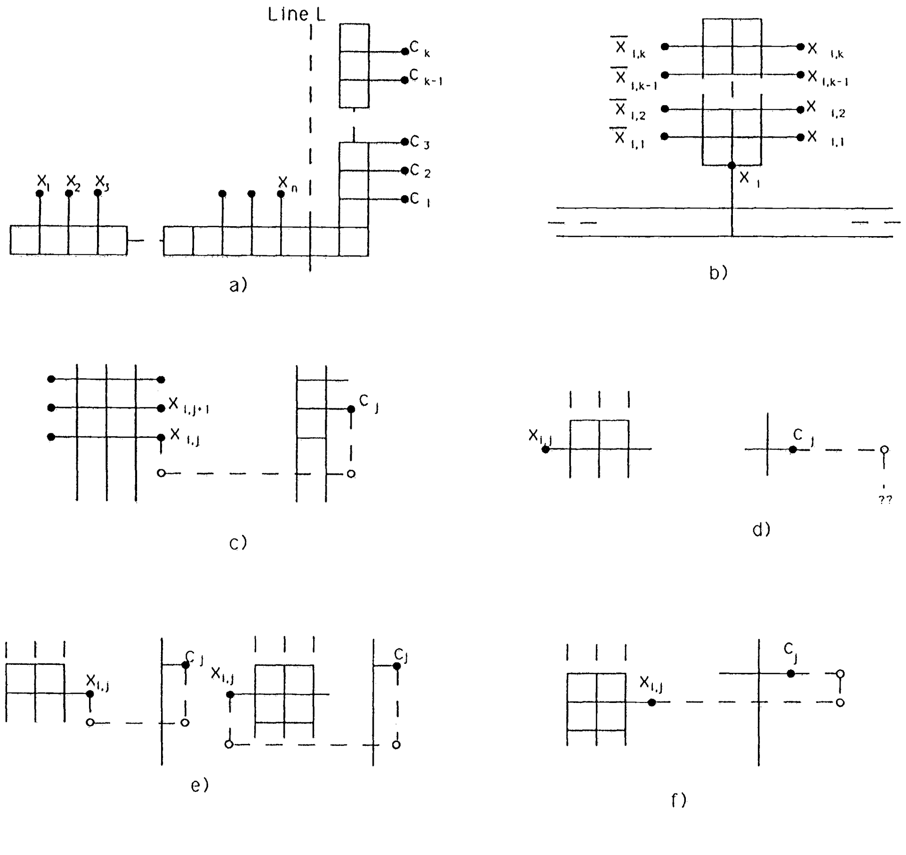


FIG. 3. *The components used in the* NP*-hardness proof.*

We claim that if there is an embedding of G with resolution $\frac{\pi}{2}$, then we can find a truth assignment for S. We make the following observations:

Observation 1. All nodes c_j, $1 \le j \le k$, have to be embedded to the right of line L (see Fig. 3(a)).

Observation 2. All nodes on the left side of a tower have to be embedded to the left of line L.

Observation 3. A path of length 3 that leaves c_j in the eastern direction can never reach any node on the left side of any tower (Fig. 3(d)).

Observation 4. A path of length 3 that leaves c_j in the southern or the northern direction can reach any node, on the left or on the right, or any tower (Fig. 3(e)).

Observation 5. A path of length 3 that leaves c_j in the eastern direction can reach any node on the right side of any tower (Fig. 3(f)).

With the above observations in mind, the rest of the proof is obvious: If the negated nodes $\overline{x}_{i,j}$, $1 \le j \le k$, are embedded on the left side of the tower at x_i, then x_i is set to TRUE. If the nonnegated nodes $x_{i,j}$, $1 \le j \le k$, are embedded on the left side, then x_i is set to FALSE. To see why the above assignment satisfies S, consider any embedded node c_j. There are three paths leaving c_j, one in the eastern, one in the northern, and one in the southern direction. The eastern path can never reach a false value. Hence, the clause will contain at least one true literal.

Conversely, if we are given a satisfying assignment for the 3-SAT problem, from the above discussion it is obvious how to embed the corresponding graph with resolution $\frac{\pi}{2}$. This completes the proof. □

3. Drawings with high resolution. In this section, we describe how to draw graphs in the plane with high resolution. Denote by $\chi(G)$ the chromatic number of graph G, i.e., the minimum number of colors necessary to color the vertices of G such that no two adjacent vertices receive the same color. We start by establishing the connection between resolution and $\chi(G^2)$ in §3.1. As a consequence, we show that any N-node graph with maximum degree d has resolution at least $\Omega(\frac{1}{d^2})$, even if we are restricted to embedding nodes in distinct grid points of a square $O(\min\{d^2, N\}^3 N)$-node grid. In §3.2, we show that $\chi(G^2) < \frac{13}{7}d + O(d^{2/3})$ for any planar graph G with maximum degree d, thereby obtaining a tight $\Theta(\frac{1}{d})$ bound on the resolution of any planar graph with maximum degree d. We conclude in §3.3 by constructing optimal-resolution layouts for a variety of special networks such as arrays, hypercubes, etc.

3.1. Drawings for general graphs based on $\chi(G^2)$. Given a graph $G = (V, E)$, the square of G (denoted by $G^2 = (V^2, E^2)$) is defined as follows: $V^2 = V$ and $E^2 = E \cup \{(i, j) | i, j \in V \text{ and } \exists\ k \in V \text{ such that } (i, k) \in E \text{ and } (k, j) \in E\}$. A simple argument reveals that if G has maximum degree d, then G^2 has maximum degree d^2, and thus that $\chi(G^2) \le d^2 + 1$. In what follows, we will show how to draw G in the plane with resolution $\frac{\pi}{\chi(G^2)} - \epsilon$ for any $\epsilon > 0$.

ALGORITHM: *DRAW*

```
Step 1 Given $G$, construct $G^2$ and color the nodes of $G^2$ with $u$ colors where $\chi(G^2) \le u \le$
       $d^2 + 1$. Adjacent nodes in $G^2$ should be assigned different colors.
Step 2 Draw a unit circle on the plane and $u$ equidistant points $P_1, \dots, P_u$ on the circle.
Step 3 Place the nodes of $G$ that are assigned color $i$ in $G^2$ into a ball of radius $\epsilon$ around
       $P_i (1 \le i \le u)$.
Step 4 Draw the edges of $G$ as straight line segments.
```

THEOREM 2. *Given a general graph G, a coloring of G^2 with u colors, and any $\epsilon > 0$, Algorithm DRAW constructs a straight-line drawing of G on the plane with resolution $\frac{\pi}{u} - O(\epsilon)$.*

Proof. Let v_1 be an arbitrary node of G and (v_1, v_2), (v_1, v_3) edges incident with v_1. In G^2, v_1, v_2, and v_3 are all adjacent, and they are colored differently (say, with colors c_1, c_2, c_3, respectively). Hence, they are placed within ϵ distance of three different points P_{c_1}, P_{c_2}, P_{c_3}, respectively, on the unit circle. The angles formed by edges connecting the u points P_i on the unit circle are all at least $\frac{\pi}{u}$. (In fact, they are all multiples of $\frac{\pi}{u}$.) Since $\text{dist}(v_i, P_{c_i}) \le \epsilon$ and

$\text{dist}(P_{c_i}, P_{c_j}) = \Omega(\frac{1}{u})$, for $1 \leq i, j \leq 3, i \neq j$, this means that the angle formed by v_1, v_2, v_3 must be at least $\frac{\pi}{u} - O(\epsilon)$, where the constant implicit in the $O(\epsilon)$ is independent of u and of the number of nodes in the graph. □

COROLLARY 1. *For any graph G with maximum degree d, and any given $\delta > 0$, we can draw G with resolution $\frac{\pi}{d^2+1} - \delta$.*

In fact, the layout given by algorithm *DRAW* can be modified so that the nodes of the graph G are placed at distinct grid points of an $O(\sqrt{u^3 N})$-side grid. This can be done by first refining the coloring of G^2 so that each color group contains at most $\frac{N}{u}$ nodes. This step introduces at most $u - 1$ new colors for a total of $2u - 1$ overall. We next place the nodes of each color group arbitrarily in a $\sqrt{N/u} \times \sqrt{N/u}$ square, and then arrange the $2u - 1$ squares at equidistant points around the perimeter of a circle of radius $u^{\frac{3}{2}}\sqrt{N}$. This results in a layout of area $O(u^3 N)$ and resolution $\Omega(\frac{1}{u})$. For graphs with bounded degree, u is constant and the layout has linear area, which is optimal. For graphs with larger u, however, it seems likely that the bound on area can be improved without dramatically affecting the bound on resolution.

3.2. Drawing planar and outerplanar graphs. We can substantially improve upon Corollary 1 in the case of planar and outerplanar graphs. In particular, we will prove that any planar graph of maximum degree d can be drawn with resolution $\Omega(\frac{1}{d})$, which matches the naive upper bound to within a constant factor. The proof is based on the fact that $\chi(G^2) = O(d)$ for any planar graph G with maximum degree d. Showing that $\chi(G^2) = O(d)$ is relatively straightforward. As the determination of the worst-case value of $\chi(G^2)$ is of independent interest, we have included the details of a $\frac{13}{7}d + O(d^{2/3})$ bound in what follows.

LEMMA 1. *Let U and W be disjoint node sets in a planar graph and suppose that each node in U has at least three neighbors in W. Then $|U| \leq 2|W| - 4$.*

Proof. Remove all nodes not in $U \cup W$ and all edges except those with one endpoint in each of U and W. The resulting graph G is planar and bipartite. Denote by m and f the number of edges and faces of G, respectively. It is easy to see that $4f \leq 2m$ and $m \geq 3|U|$. Hence, by Euler's formula,

$$|U| + |W| - 2 = m - f \geq m/2 \geq 3|U|/2$$

and $|U| \leq 2|W| - 4$. □

DEFINITION. For $k \geq 1$, denote by $\phi(k)$ the supremum, over all planar graphs G, of the proportion of nodes in G of degree $\geq k$.

LEMMA 2.

$$\phi(k) = \begin{cases} 1, & \text{for } k \leq 6; \\ \frac{3}{k-3}, & \text{for } 6 \leq k \leq 12; \\ \frac{2}{k-6} & \text{for } k \geq 12. \end{cases}$$

Proof. First, we will give proofs for the upper bounds. For the range $k \leq 6$, the upper bound is trivial.

For $6 \leq k \leq 12$, let $G = (V, E)$ be a planar graph and define W as the set of nodes in G of degree $\geq k$. At least $k|W| - 3|W|$ edges join a node in w with a node in $V\backslash W$. This implies that

$$|W|(k-3) \leq |E| \leq 3|V| = 3(|W| + |V\backslash W|)$$

and hence,

$$\begin{aligned} &|W|(k-6) \leq 3|V\backslash W| \\ \Rightarrow\ &\frac{|W|(k-6)}{3} \leq |V\backslash W|. \end{aligned}$$

Hence, the proportion of nodes of degree $\geq k$ is at most

$$\frac{|W|}{|W|+|V\backslash W|} \leq \frac{|W|}{|W|+|W|(\frac{k-6}{3})} = \frac{1}{1+(\frac{k-6}{3})} = \frac{3}{k-3}.$$

Next, we show the upper bound for $k \geq 12$.

Fix $k \geq 12$, let $G = (V, E)$ be a planar graph, and define W as the set of nodes in G of degree $\geq k$ and U as the set of nodes in $V\backslash W$ with at least three neighbors in W. By Lemma 1, $|U| < 2|W|$.

By planarity, the subgraph of G induced by $U \cup W$ has at most $3(|U| + |W|)$ edges, at most $3|W|$ of which have both endpoints in W. Hence, at least $k|W| - 3|U| - 6|W|$ edges in G join a node in W with a node in $V\backslash(U \cup W)$. It follows that

$$|V\backslash(U \cup W)| \geq \tfrac{1}{2}(k|W| - 3|U| - 6|W|,$$

and hence, that

$$\begin{aligned} |V| &= |W| + |U| + |V\backslash(U \cup W)| \\ &\geq |W| + |U| + \tfrac{1}{2}(k|W| - 3|U| - 6|W|) \\ &= \tfrac{1}{2}k|W| - 2|W| - \tfrac{1}{2}|U| \\ &\geq \tfrac{1}{2}k|W| - 2|W| - |W| = (\tfrac{k}{2} - 3)|W|. \end{aligned}$$

Finally, the proportion of nodes in G of degree $\geq k$ is

$$\frac{|W|}{|V|} \leq \frac{|W|}{(\frac{k}{2}-3)|W|} = \frac{2}{k-6}.$$

Now for each k we give a planar graph, which realizes the lower bound. The infinite triangular grid serves as a basis. The degree of each node is 6, such that the grid establishes the lower bound for $k \leq 6$. In the following, we consider the triangular grid embedded in the plane such that the faces are equal-sided triangles, and we can consider the faces as lying in horizontal rows.

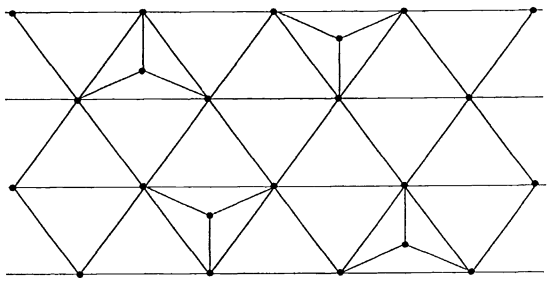
To construct the graph for a specific value of k between 7 and 12, we start with the graph for $k - 1$ and extend certain subset of the faces, which are left unchanged in previous steps. The extension of a single-face f consists of an insertion of a node v, which is connected to the three nodes on the boundary of f. In each step, we ensure that the degree of each original node is increased by exactly 1. Figure 4(a) shows the resulting graph for $k = 7$.

Then the number of new nodes is $\frac{k-6}{3}$ times greater than the number of the original nodes. And the proportional of the original nodes is

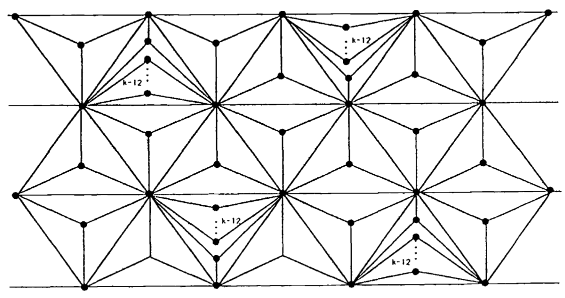
$$\frac{1}{1+\frac{k-6}{3}} = \frac{3}{3+k-6} = \frac{3}{k-3}.$$

For values $k \geq 12$, we extend the augmented graph for $k = 12$ by adding $k - 12$ paths of length 2 parallel to every other horizontal edge of the triangular grid (Fig. 4(b)). The degree of every original node is k, the nodes inserted in the previous extension have degree 3, and the new nodes have degree 2. Hence, there are as twice as many nodes inserted in the previous step as the number of original nodes. The number of new nodes is $\frac{k-12}{2}$ times larger than the number of the original nodes. Hence, the proportion of the original vertices is

$$\frac{1}{1+2+\frac{k-12}{2}} = \frac{2}{k-6}. \qquad \square$$



(a) *Graph for k* $= 7$.



(b) *Graph for k* ≥ 12.

FIG. 4. *Graphs that realize the lower bounds on the supremum over all planar graphs.*

THEOREM 3. *The square of any planar graph G with maximum degree d can be colored using at most* $\frac{13}{7}d + O(d^{2/3})$ *colors.*

Proof. Let k, l, and d be any integers for which $k \geq l \geq 12$ and $d \geq k + l$. We will prove by induction on the number of nodes that the square of any planar graph with maximum degree at most d can be colored using at most $\Delta = d + (l-3)(l-1) + \max\{k, d - \lfloor \frac{l-12}{l-6} \frac{k-6}{6} \rfloor + 1\}$ colors. By setting $k = \lceil \frac{6}{7}d \rceil$ and $l = \Theta(d^{1/3})$, this will produce the desired asymptotic bound as d becomes large. As can be seen easily, the constant factors associated with the low-order terms will not be large.

Define W as the set of nodes in G of degree $\geq k$ and U as the set of nodes in G of degree $< l$ and with at most two neighbors of degree $\geq l$. By Lemmas 1 and 2, $|W| \leq \frac{2}{k-6}|V|$ and

$$|U| > |V| - 3 \cdot \frac{2}{l-6}|V| = \frac{l-12}{l-6}|V| \geq 0.$$

Case 1. Some node v in U has at most one neighbor in W. Contract v into a neighbor w of v of minimal degree, i.e., add an edge between w and each node other than w that is a neighbor of v, but not of w, and subsequently remove v. Since $w \notin W$ unless v is of degree 1, the new degree of w is bounded by $\max\{d, k + l - 3\} = d$, and the inductive hypothesis implies that the square of the resulting graph can be colored with at most Δ colors. Furthermore, the colors assigned can be retained in a valid coloring of G^2, the only remaining problem being to color v.

Since the number of nodes in G at distance 1 or 2 from v is at most $d+(k-1)+(l-3)(l-1)$, the indicated number of colors suffices.

Case 2. Every node in U has exactly two neighbors in W. By planarity, at most $3|W|$ pairs of nodes in W can have a common neighbor in U. Hence, some pair of nodes in W has at least

$$\frac{|U|}{3|W|} > \frac{l-12}{l-6} \cdot \frac{k-6}{6}$$

common neighbors in U. Choose v as one of these common neighbors and contract v into a neighbor of minimal degree as above. Again, this does not increase the maximum degree, and the inductive assumption applies to the resulting graph. Finally, note that the number of nodes in G at distance 1 or 2 from v is less than

$$2d - \tfrac{l-12}{l-6} \cdot \tfrac{k-6}{6} + (l-3)(l-1),$$

and we can find an acceptable color for v. $\square$

From this fact, we can conclude that planar graphs with optimal resolution R for a fixed constant c can be embedded with resolution at least $\frac{7R}{26} - cR^{4/3}$.

THEOREM 4. *Any planar graph with maximum node degree d has resolution $\frac{7\pi}{13d-cd^{-1/3}} \leq R \leq \frac{2\pi}{d}$.*

Proof. The upper bound is trivial. The lower bound follows from Theorems 2 and 3. $\square$

For outerplanar graphs, the bounds on $\chi(G^2)$ are much tighter, as we show in what follows.

LEMMA 3. *Every biconnected outerplanar graph on at least three nodes contains a node of degree* 2 *with a neighbor of degree* 2 *or with adjacent neighbors, one of which is of degree at most* 4.

Proof. Let T be the dual of an outerplanar embedding $\mathcal{E}$ of the given graph, with (the node representing) the outer face removed. As is well known, T is a free tree, i.e., it is connected and acyclic. A face of $\mathcal{E}$ is a leaf, i.e., of degree 1, in T if and only if exactly one of its boundary edges does not bound the outer face. Let F_r and F_v be faces of $\mathcal{E}$ whose distance from each other in T is maximal. Root T at F_r, let v be any node of degree 2 on the boundary of F_v, and let v_α and v_β be the neighbors of v (Fig. 5). If v_α or v_β is of degree ≤ 2, we are done. Otherwise, define F, F_α, and F_β as shown in the figure. At least one of F_α and F_β, say F_α, is a child of F in T, and hence, by the choice of F_r and F_v, a leaf in T. But then v_α is of degree 4. $\square$

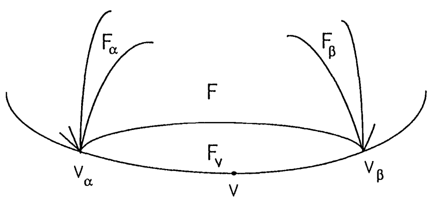


FIG. 5. *The faces of the planar graph used in the proof of Lemma* 3.

THEOREM 5. *The square of any outerplanar graph G of maximum degree d can be colored using at most $d+3$ colors.*

Proof. We can assume that G is biconnected and contains at least three nodes. Let H be the set of nodes in G of degree 2 with at least one neighbor of degree 2. If $H \neq \emptyset$, remove all nodes in H and, if any nodes are left, color the remaining graph inductively, using $d+3$ colors. Then obtain a coloring of the original graph by adding back the nodes in H and coloring them in an arbitrary order. Since at most $d+2$ nodes have distance 1 or 2 from each fixed node in H, this can be done using $d+3$ colors.

If $H \neq \emptyset$, remove a node v of degree 2 with adjacent neighbors, one of which is of degree ≤ 4, and color the remaining graph inductively. Since there are at most $d+2$ nodes at distance 1 or 2 from v, the proof again is completed easily. □

It is worth noting that the bound in Theorem 5 is nearly tight since the $(d+1)$-node star graph is an outerplanar graph G with maximum node degree d for which $\chi(G^2) = d+1$. There exist quite straightforward implementations, based on the proofs of Theorems 3 and 5, to find the coloring of the square of any outerplanar and planar graph. So, we state the following theorem 6.

THEOREM 6. (a) *The square of any outerplanar graph G with maximum degree d can be colored using at most $d+3$ colors in time linear with respect to the number of nodes.* (b) *The square of any planar graph G with maximum degree d can be colored using at most $\frac{13}{7}d + O(d^{2/3})$ colors in time quadratic to the number of nodes.*

3.3. Special networks. In this section, we examine the resolution of some special networks. We present optimal or nearly optimal layouts for the complete graph, the hypercube, multidimensional arrays and tori, the mesh of cliques, and the projective plane. The first four of these networks are important because of their uses as processor interconnection networks. The last two networks are interesting because they would seem to be good candidates for graphs with resolution $\Theta(\frac{1}{d^2})$ since the chromatic number of square of a d-regular mesh of cliques and projective plane is $\Theta(d^2)$. Somewhat surprisingly, however, we show that the resolution of all the special networks mentioned is $\Theta(\frac{1}{d})$.

3.3.1. The complete graph.

THEOREM 7. *The complete graph of N nodes has resolution $\frac{\pi}{N}$.*

Proof. For the lower bound, draw the N nodes of the graph at equidistant points on a circle. Then the angles formed by incident edges will have size at least $\frac{\pi}{N}$.

For the upper bound, consider the convex hull of the nodes in some layout. If the convex hull has l vertices, then the sum of the inner angles at the vertices is $(l-2)\pi$; so there is one vertex v, where the inner angle is at most $(1-\frac{2}{l})\pi \leq (1-\frac{2}{N})\pi$. All the other $N-1$ nodes are contained in that angle, so the edges to two of them form an angle of at most $(1-\frac{2}{N})\pi/(N-2) = \frac{\pi}{N}$. □

Actually, the proof gives us the following more general statement.

COROLLARY 2. *The resolution of any d-regular graph is at most $\frac{\pi}{d-1}$.*

3.3.2. Hypercubes and multidimensional meshes. In this section, we consider hypercubes and multidimensional meshes. The k-hypercube has 2^k nodes, each one represented by a k-tuple $(i_1, i_2, \ldots, i_k) \in \{0,1\}^k$. Edges occur between nodes that differ in precisely one bit. By Corollary 2, we know that any layout of a k-hypercube has resolution at most $\frac{\pi}{k-1}$. In what follows, we present an algorithm that draws the k-hypercube with resolution $\frac{\pi}{k}$. We then extend this algorithm to derive an optimal layout for the k-dimensional mesh.

```
ALGORITHM: HYPERCUBE(k)
Step 1 Design on the plane an angle $\hat{\phi}$ of size $\pi - \frac{\pi}{k}$. Divide $\hat{\phi}$ into k-1 equal angles
       which define a k-axes system.
Step 2 Initialization: Create a 1-hypercube on the 1st axis.
Step 3 for $j = 2$ to $k$ do
    3.1 Create a copy of the $(j-1)$-hypercube.
    3.2 Translate the copy parallel to the $j$th axis.
    3.3 Create connections between the corresponding nodes of the two $(j-1)$-hyper-
        cubes.
```

Since each line segment drawn by algorithm *HYPERCUBE*(k) is parallel to one of the k axes, we have the following Theorem 8.

THEOREM 8. *Algorithm HYPERCUBE draws the k-hypercube in the plane with resolution* $\frac{\pi}{k}$.

THEOREM 9. *For* 3*-hypercubes, the algorithm HYPERCUBE is optimal.*

Proof. Assume we have a layout of the 3-hypercube with resolution $> \frac{\pi}{3}$. Each vertex v lies on three 4-cycles. One of these 4-cycles has an interior angle $> \frac{2\pi}{3}$ at v. Hence, the six 4-cycles of the cube have eight angles $> \frac{2\pi}{3}$. We may conclude that one of the remaining angles of the 4-cycles in smaller than $\frac{\pi}{3}$, a contradiction. □

An algorithm to embed a k-dimensional mesh with resolution $\frac{\pi}{k}$ can be obtained by extending algorithm *HYPERCUBE*. Note that a k-hypercube is the basic unit component of a k-dimensional mesh. The maximum degree for any internal node of the mesh is $2k$. Since $\frac{2\pi}{d}$ is an obvious upper bound for the resolution of any graph of maximum degree d, the extended algorithm will produce an optimal embedding.

3.3.3. Tori. The m-dimensional torus network $T(m)$ is actually an m-dimensional mesh with wrap-around connections.

THEOREM 10. *The m-dimensional torus can be embedded on the plane with resolution* $\frac{\pi}{2m}$, *provided that all dimensions have size greater than* 3.

Proof. Consider the embedding of the 4×4 torus (Fig. 6). Observe that it replicates the embedding of the 4-hypercube. We use this embedding as a base layout of any two-dimensional torus. We can extend the embedding of the 4×4 torus to any $a \times b$ torus by inserting extra nodes in regular intervals of the edges of the respective dimensions. By a similar argument, the embedding of an m-dimensional $4 \times 4 \times \cdots \times 4$ torus can be used as a base layout of any m-dimensional torus, since it is isomorphic to the $2m$-hypercube. From Theorem 8, we know that the k-hypercube can be embedded with resolution $\frac{\pi}{k}$. Therefore, the proof follows. □

3.3.4. Mesh of cliques. The *m-regular mesh of cliques* is defined to be the regular graph with m^2 nodes arranged as an $m \times m$ mesh. All nodes on the same row of the mesh are connected in a clique. The same holds for nodes in the same column. Obviously, the m-regular mesh of cliques has degree $2m-2$. It also has the property that between any two of its nodes there exists a path of length 2. Thus, its square graph is a clique of size m^2. In the following, we present a layout of the m-regular mesh of cliques, which has resolution $O(\frac{1}{m})$.

THEOREM 11. *The m-regular mesh of cliques can be embedded on the plane with resolution* $\frac{\pi}{2m}$.

Proof (by construction). We group the cliques that correspond to rows into m regular m-gons. Then we place these m-gons on the plane such that their centers are located on the boundary of a circle and also form a regular m-gon. Moreover, the nodes of each column should form a regular m-gon. Observe that the angles formed by any two edges that belong to the same "row (column) clique" have resolution at least $\frac{\pi}{m}$ degrees. Thus, we only need to consider the angles formed by an edge that belongs to a "column clique" with an edge that belongs to a "row clique." Notice that for any regular m-gon the slopes of the lines in the m-gon all differ from each other by a multiple of $\frac{\pi}{m}$. Hence, if we rotate the m-gons that

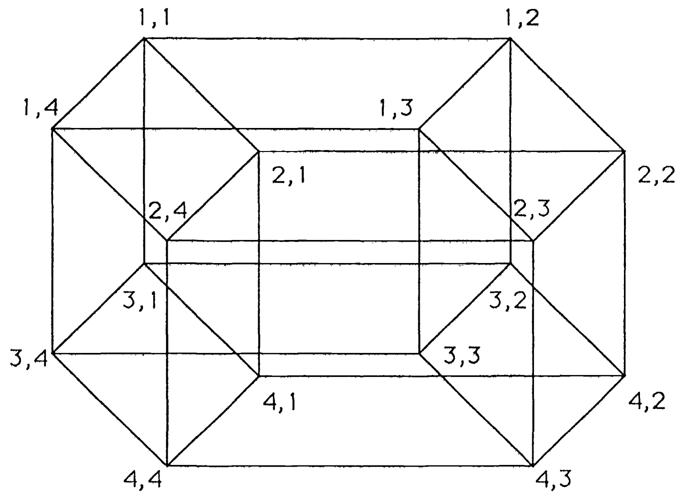


FIG. 6. *The embedding of the* 4 × 4 *torus.*

correspond to "row cliques" by $\frac{\pi}{2m}$ relative to m-gons that corresponds to "column cliques," then the embedding has resolution $\frac{\pi}{2m}$. □

3.3.5. Projective plane. The *projective plane* consists of a set of objects called *points*, a second set of objects called *lines*, and a notion of when a point lies on a line, so that the following three conditions are satisfied:

(C1) Two distinct points lie on one and only one common line.

(C2) Two distinct lines pass through one and only one point.

(C3) There are four distinct points, no three of which lie on the same line.

In a *projective plane of order* $d - 1$, every point lies on exactly d lines and every line passes through exactly d points. A projective plane of order $d - 1$ exists if $d - 1$ is a prime power. It has exactly $d^2 - d + 1$ points and $d^2 - d + 1$ lines. The projective plane of order $d - 1$ can be represented as a bipartite graph $G = (A, B, E)$ where the set of nodes A corresponds to points, the set of nodes B corresponds to lines, and an edge (u, v), $u \in A$ and $v \in B$, belongs to E if and only if point u lies on line v. Note that in the square graph of G the nodes of A and B form two cliques, and thus $\chi(G^2) = \Theta(d^2)$.

As in the case of the mesh of cliques, however, the projective plane can be drawn with resolution $\Theta(\frac{1}{d})$. The embedding is described roughly as follows.

A projective plane of order $d - 1$ can be drawn using the following method (non-straight-line drawing): We draw a square grid of $(d - 1)^2$ points. We say that a point belongs in the set P_i, if it belongs to the ith column, where $0 \leq i < d - 1$. Clearly, we have $d - 1$ points in each set. Grid points are connected by lines with slopes $0, 1, \ldots, d - 2$, and ∞. We say that a line belongs in the set L_i, $0 \leq i < d - 1$ or $i = \infty$, if it has slope i. In addition to the grid points, we have d more points called the *infinity points*. We call an infinity point the P_i^∞ point if it is contained in all lines with slope i, where $0 \leq i < d - 1$ or $i = \infty$ (parallel lines connected at infinity). We denote by L_∞^i the line that is of ∞ slope and contains the points of the ith column, where $0 \leq i < d-1$. Line L_∞^∞ contains all infinity points. Figure 7 illustrates the method for $d = 3$.

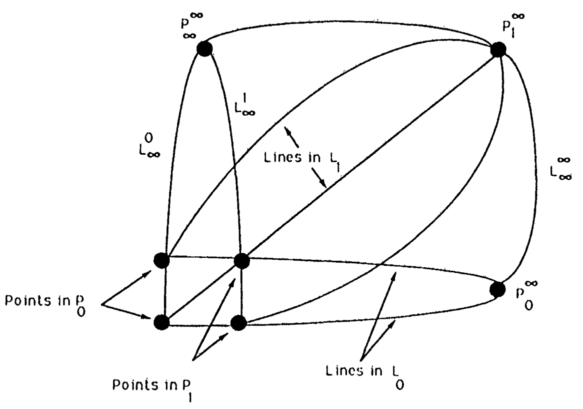


FIG. 7. *The projective plane of order* 2 ($d = 3$).

We can draw a projective plane of order $d - 1$ with straight lines using the following method: We arrange the nodes (points, lines) into two tree structures, which are identical if we interchange the notion of point and line. Therefore, we only describe one of the tree structures.

The nodes in the set P_i, $1 \leq i \leq d-1$, are leaves of the tree, and furthermore, they are drawn as colinear points. Nodes in set P_i are placed a unit apart from each other, while nodes in set P_{i+1} are placed at distance $O(d)$ units apart from the nodes in P_i, for $0 \leq i < d-2$. Node L_∞^i is the root of all the nodes in set P_i, for $0 \leq i < d-1$. The L_∞^i nodes are placed such that they form a line parallel to the one formed by the leaves of the tree. Node L_∞^i is placed $O(d)$ units apart from node L_∞^{i+1}, for $0 \leq i < d-1$. Node P_∞^∞ is the root of all L_∞^i nodes and is placed $O(d^2)$ units apart from them. This concludes the drawing of the tree structure.

We now place on tree structure as a mirror image of the other with the leaves facing each other and placed $O(d^2)$ units apart. We make all necessary connections between nodes P_i and L_j. Finally, we connect the roots of the trees and refine the structure so that the angles formed in the two roots are sufficiently large. Figure 8 illustrates the method. We observe that no angle is smaller than $\Omega(\frac{1}{d})$. Thus, we have the following Theorem 12.

THEOREM 12. *The projective plane of order $d-1$ can be embedded on the plane with resolution $O(\frac{1}{d})$.*

4. An upper bound on the resolution of a random graph. In this section, we prove that many graphs with maximum degree d have resolution at most $O(\frac{\log d}{d^2})$, for any d. We will prove this result with a counting argument. For simplicity we will consider directed graphs with outdegree d, and we will restrict our attention to the angles formed by the outgoing edges at each node. (Angles with incoming edges are ignored.) We will show that almost all such graphs with N nodes ($N \geq d^2$) have resolution $O(\frac{\log d}{d^2})$. Since almost all graphs with outdegree d have indegree $O(d + \log N)$, this means that many undirected graphs with degree $O(d)$ have resolution $O(\frac{\log d}{d^2})$. It is probably also true that almost all d-regular graphs have resolution $O(\frac{\log d}{d^2})$, but the proof appears to be more complicated.

The proof will make use of the following combinatorial facts.

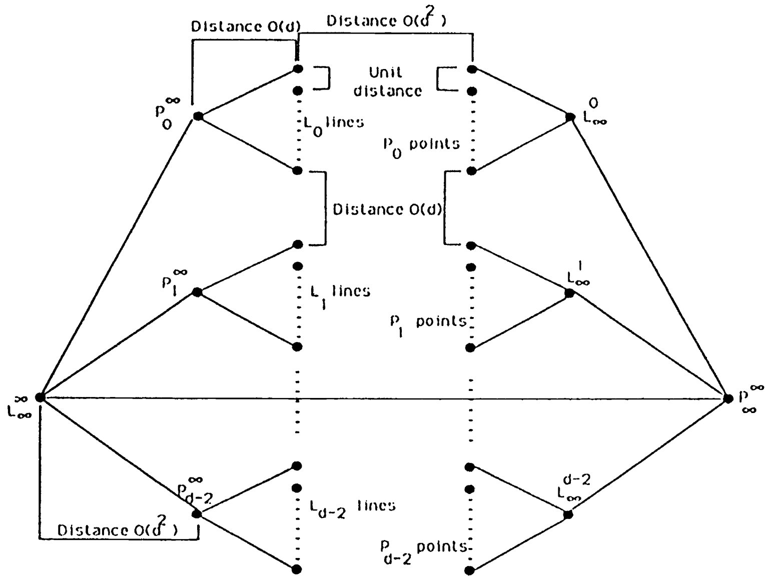


FIG. 8. *The general layout for the projective plane of order $d-1$.*

FACT 1. For all a, b, $\binom{a}{b} \leq (\frac{ae}{be^{b/2a}})^b$
Proof.

$$\binom{a}{b} \leq \frac{a^a}{b^b(a-b)^{a-b}} = \frac{a^b}{b^b(1-\frac{b}{a})^{a-b}}$$
$$= \frac{a^b}{b^b e^{(-\frac{b}{a}-\frac{b^2}{2a^2}-\frac{b^3}{3a^3}\ldots)(a-b)}} = \frac{a^b}{b^b e^{(-b+\frac{b^2}{2a}+\frac{b^3}{6a^2}\ldots)}} \leq \left(\frac{ae}{be^{b/2a}}\right)^b \qquad \square$$

FACT 2. For $a > 2b$, $\binom{a}{b} \geq \dfrac{a^b e^b}{\sqrt{2\pi b}\, b^b e^{\frac{b^2}{2a}+O(b^3/a^2)}}$
Proof.

$$\binom{a}{b} = \frac{a!}{b!(a-b)!} \sim \sqrt{\frac{a}{2\pi b(a-b)}}\,\frac{a^a}{b^b(a-b)^{a-b}}$$
$$\geq \frac{1}{\sqrt{2\pi b}}\,\frac{a^b}{b^b(1-\frac{b}{a})^{a-b}} \sim \frac{a^b e^b}{\sqrt{2\pi b}\, b^b e^{\frac{b^2}{2a}+O(b^3/a^2)}}$$

for $a > 2b$. $\square$

FACT 3. Given m boxes containing $n_1, n_2, \ldots, n_m$ labeled balls, respectively, the number of ways of choosing j balls from the boxes so that at most one ball is chosen from each box is at most

$$\binom{m}{j}\left(\frac{n}{m}\right)^j,$$

where $n = n_1 + n_2 + \cdots + n_m$.

Proof. The maximum is achieved when each box has the same number of balls; i.e., $n_i = \frac{n}{m}$ for all i. The bound then follows straightforwardly. $\square$

FACT 4. Given $m + 1$ boxes containing $n_0, n_1, \ldots, n_m$ labeled balls, respectively, the number of ways of choosing d balls from the boxes so that at most one ball is chosen from boxes $1, 2, \ldots, m$ (any number can be taken from box 0) is at most

$$(d+1)\left(\frac{ne}{de^{(1-x)^2d/4m}}\right)^d,$$

where $x = \frac{n_0}{n}$ and $n = n_0 + n_1 + \cdots + n_m$.

Proof. We can compute the number of possibilities using Facts 1 and 3 and by letting $j = \alpha d$ denote the number of balls coming from box 0. Then, by Fact 3, the number of ways of selecting the balls is at most

$$\sum_{j=0}^{d}\binom{n_0}{j}\binom{m}{d-j}\left(\frac{n-n_0}{m}\right)^{d-j}$$
$$\leq (d+1)\max_{0\leq\alpha\leq 1}\binom{n_0}{\alpha d}\binom{m}{(1-\alpha)d}\left(\frac{n-n_0}{m}\right)^{(1-\alpha)d}.$$

By using Fact 1, we get that the above is

$$\leq (d+1)\max_{0\leq\alpha\leq 1}\left(\frac{n_0e}{\alpha d}\right)^{\alpha d}\left(\frac{me}{(1-\alpha)de^{(1-\alpha)d/2m}}\right)^{(1-\alpha)d}\left(\frac{n-n_0}{m}\right)^{(1-\alpha)d}$$
$$\leq (d+1)\max_{0\leq\alpha\leq 1}\left(\frac{nex^{\alpha}(1-x)^{(1-\alpha)}}{d\alpha^{\alpha}(1-\alpha)^{(1-\alpha)}e^{(1-\alpha)^2d/2m}}\right)^d$$
$$\leq (d+1)\left(\frac{ne}{de^{(1-x)^2d/4m}}\right)^d$$

because the maximum of

$$\frac{x^{\alpha}(1-x)^{(1-\alpha)}}{\alpha^{\alpha}(1-\alpha)^{(1-\alpha)}e^{(1-\alpha)^2d/2m}}$$

occurs for a value of α very near x. $\square$

FACT 5. Given any placement of N points on the plane, it is possible to find concentric circles with radii r_1 and r_2 so that at least $\frac{N}{5}$ points are inside or on the boundary of the inner circle, and so that at least $\frac{N}{5}$ points are outside or on the boundary of the outer circle, where $r_2 \geq \sqrt{2}\, r_1$.

Proof. Find a smallest circle that contains at least $\frac{N}{5}$ points. This will be the inner circle; it has radius r_1. Let r_2 be the radius of the largest concentric outer circle that leaves $\frac{N}{5}$ points outside. Since the concentric circle with radius $\sqrt{2}\, r_1$ can be covered with four circles of radius r_1 (Fig. 9), there are at least $\frac{N}{5}$ points outside of the concentric circle with radius $\sqrt{2}\, r_1$. Hence, $r_2 \geq \sqrt{2}\, r_1$. $\square$

We can now bound the resolution of a random graph with outdegree d.

THEOREM 13. *Given a random N-node directed graph G in which every node has outdegree d, with high probability, every embedding of G has resolution $O(\frac{\log d}{d^2})$.*

Proof. We will count all graphs with N-labeled nodes and outdegree d that can be constructed so that there is an embedding in which every angle formed by outgoing edges is

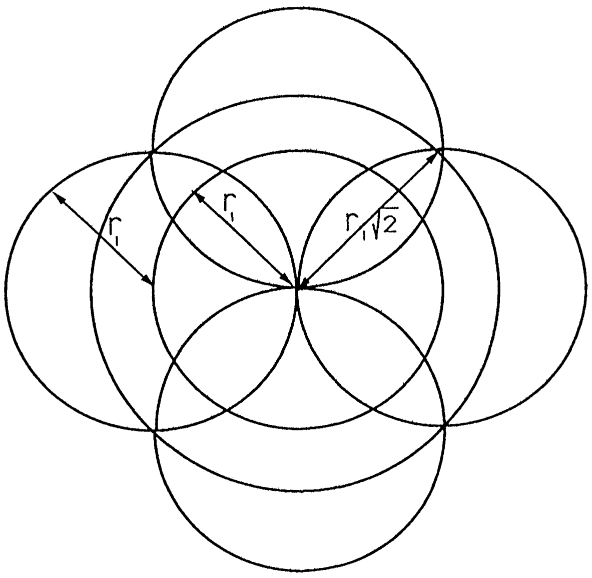


FIG. 9. *Covering a circle of radius $r_1\sqrt{2}$ with four circles of radius r_1.*

at least $\frac{\log d}{cd^2}$, where c is a sufficiently small constant. We will show that this number is far less than $\binom{N-1}{d}^N$, which is the number of N-node outdegree d graphs, thereby implying the theorem.

Given any embedding of any graph, we know by Fact 5 that there are concentric circles with radii r_1 and r_2 so that $\frac{N}{5}$ points are contained in the inner circle, $\frac{N}{5}$ points are outside the outer circle, and $r_2 \geq \sqrt{2}\, r_1$. Partition the region outside the outer circle into m equal slices, as shown in Fig. 10. Notice that if any node within the inner circle is connected by outgoing edges to two or more nodes in the same slice, then there must be an angle of size $O(\frac{1}{m})$.

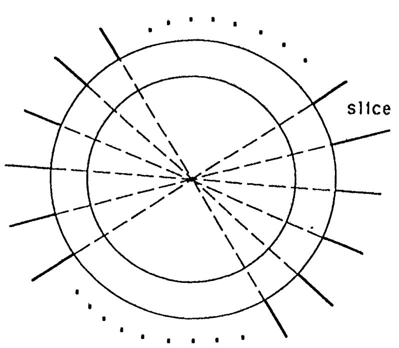


FIG. 10. *The partition of the plane used in the proof of Theorem* 13.

In what follows, we will show that, if $m = \frac{cd^2}{\log d}$, then there are far less than $\binom{N-1}{d}^N$ ways of constructing such a graph. The counting proceeds as follows.

(a) The number of ways to pick $\frac{N}{5}$ nodes to be inside the inner circle is

$$\binom{N}{N/5} \le (5e)^{\frac{N}{5}};$$

(b) The number of ways to pick $\frac{N}{5}$ nodes to be outside the outer circle is

$$\binom{4N/5}{N/5} \le (4e)^{\frac{N}{5}};$$

(c) The number of ways to assign the $\frac{N}{5}$ selected nodes outside to slices is

$$m^{\frac{N}{5}};$$

(d) The number of ways to connect $\frac{4N}{5}$ nodes outside the inner circle to other nodes is

$$\binom{N-1}{d}^{\frac{4N}{5}};$$

(e) The number of ways to connect $\frac{N}{5}$ nodes inside the inner circle to other nodes, such that no two outgoing edges of one of these nodes go to nodes in the same slice, is (Fact 4)

$$\le \left((d+1)\left(\frac{(N-1)e}{de^{(1-4/5)^2 d/4m}}\right)^d\right)^{\frac{N}{5}}.$$

Thus, the total number of graphs that do not have resolution $O(\frac{1}{m})$ divided by the total number of graphs overall is

$$\le \frac{(5e)^{\frac{N}{5}}(4e)^{\frac{N}{5}}m^{\frac{N}{5}}\binom{N-1}{d}^{\frac{4N}{5}}\left((d+1)\left(\frac{(N-1)e}{de^{(1-4/5)^2 d/4m}}\right)^d\right)^{\frac{N}{5}}}{\binom{N-1}{d}^N}$$
$$= \left(\frac{20e^2 m(d+1)\left(\frac{(N-1)e}{de^{d/100m}}\right)^d}{\binom{N-1}{d}}\right)^{\frac{N}{5}}.$$

By using Fact 2, we get that the above is

$$\le \left(\frac{20e^2 m(d+1)\frac{(N-1)^d e^d}{d^d e^{d^2/100m}}}{\frac{(N-1)^d e^d}{\sqrt{2\pi d} d^d e^{d^2/2(N-1)+O(d^3/N^2)}}}\right)^{\frac{N}{5}}$$
$$\le \left(\frac{20\sqrt{2\pi}e^2 m(d+1)^{3/2} e^{d^2/2(N-1)+O(d^3/N^2)}}{e^{d^2/100m}}\right)^{\frac{N}{5}}.$$

By choosing $N \ge d^2$ and $m \le \frac{cd^2}{\log d}$ for some small constant c, the value in the brackets can be made smaller than $\frac{1}{d^5}$. Hence, the probability of getting a graph with resolution $O(\frac{\log d}{d^2})$ is at least $1 - \frac{1}{d^N}$. $\square$

5. Remarks. There are several questions left open in this paper. We list some of them below.

1. Is the problem of determining the resolution of a graph in NP?
2. Are there interesting trade-offs between the resolution of a layout and its area for graphs with large maximum d? Can the area bound in §3 by improved?
3. What is the worst case value of $\chi(G^2)$ if G is a planar graph with maximum degree d?
4. What happens to the resolution of planar graphs if we restrict the layout to be planar? Does every degree-3 planar graph have a planar embedding with constant (independent of the number of nodes) resolution?
5. Is there a meaningful relationship between the resolution of a graph and its density (see [7]–[8] for definitions)? (In the case of planar graphs, the two quantities appear to be very similar.)

Acknowledgments. We would like to thank Alok Aggarwal, Leo Guibas, Mike Klugerman, Gary Miller, and Joel Spencer for helpful discussions.

REFERENCES

[1] S. N. Bhatt and F. T. Leighton, *A framework for solving* VLSI *graph layout problems*, J. Comput. System Sci., 28 (1984), pp. 300–343.

[2] S. N. Bhatt and C. E. Leiserson, *Minimizing the Longest Edge in a* VLSI *Layout*, unpublished memorandum, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

[3] F. J. Brandenburg, *Nice drawings of graphs are computationally hard*, Proceedings of the 7th Interdisciplinary workshop on Informatics and Psychology, May 1988, LNCS 439, pp. 1–15.

[4] H. De Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.

[5] P. Eades and R. Tamassia, *Algorithms for Automatic Graph Drawing: An Annotated Bibliography*, Tech. Report CS-89-09, Department of Computer Science, Brown University, Providence, RI, 1989.

[6] I. Fàry, *On straight lines representations of planar graphs*, Acta Sci. Math. (Szeged), 11 (1948), pp. 229–233.

[7] C. E. Leiserson, *Area-efficient graph layouts (for* VLSI), Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, October 1980, pp. 270–281.

[8] F. Lin, *Optical Holographic Interconnection Networks for Parallel and Distributed Processing*, Optical Comp. Technical Digest Series, 9 (1989), pp. 150–153.

[9] G. Miller and S. Vavasis, *Density graphs and separators*, Proceedings of the 2nd Annual Symposium on Discrete Algorithms, 1991.

[10] G. Miller, S.-H. Teng, and S. Vavásis, *A Unified Geometric Approach to Graph Separators*, unpublished manuscript, 1990.

[11] R. Arrathoon, ed., *Optical Computing: Digital and Symbolic*, Marcel Dekker, New York, 1989.

[12] C. H. Papadimitriou, *The NP-completeness of the bandwidth minimization problem*, Computing, 16 (1976), pp. 263–270.

[13] P. Rosenstiehl and R. E. Tarjan, *Rectilinear planar layouts and bipolar orientations of planar graphs*, Discrete Comput. Geom., 1 (1986), pp. 343–353.

[14] U. Schnieders, *Zeichnen von planaren Graphen mit Winkeloptimierung*, Diplomarbeit, Fakultät für Mathematik und Informatik, Universität Passau, 1990.

[15] W. Schnyder, *Embedding planar graphs on the grid*, Proceedings of the 1st Annual Symposium on Discrete Algorithms, 1990, pp. 138–148.

[16] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Comput., 16 (1987), pp. 421–444.

[17] R. Tamassia, C. Batini, and M. Talano, *An algorithm for automatic layout of entity-relationship diagrams*, J. Systems and Software, 4 (1984), pp. 147–162.

[18] L. G. Valiant, *Universality considerations in* VLSI *circuits*, IEEE Trans. Comput., C-30 (1981), pp. 135–140.

[19] M. Yannakakis, *A polynomial algorithm for the min cut linear arrangement of trees*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 274–281.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 1053–1074, October 1993

PROBABILISTIC ANALYSIS OF DISJOINT SET UNION ALGORITHMS*

BÉLA BOLLOBÁS† AND ISTVAN SIMON‡

Abstract. A number of open questions are settled about the expected costs of two disjoint set Union and Find algorithms raised by Knuth and Schönhage [*Theoret. Comput. Sci.*, 6 (1978), pp. 281–315]. This paper shows that the expected time of the Weighted Quick-Find (QFW) algorithm to perform $(n-1)$ randomly chosen unions is $cn + o(n/\log n)$, where $c = 2.0847\ldots$. Through an observation of Tarjan and Van Leeuwen in [*J. Assoc. Comput. Mach.*, 22 (1975), pp. 215–225] this implies linear time bounds to perform $O(n)$ unions and finds for a class of other union-find algorithms. It is also proved that the expected time of the Unweighted Quick-Find (QF) algorithm is $n^2/8 + O(n(\log n)^2)$. The expected costs of QFW and QF are analyzed when fewer than $(n-1)$ unions are performed. Among other results, for QFW it is shown that the expected cost of $m = o(n)$ randomly chosen unions is $m(1+o(1))$. If $m = \alpha n/2$, where $\alpha \le e^{-2}$, this cost is $m(1+\epsilon(\alpha)+o(1))$, where $\epsilon(\alpha) \to 0$ as $\alpha \to 0$ and $\epsilon(e^{-2}) \le .026$. For QF, the expected cost of $n/2 - n^{2/3}(\log n)^{2/3}$ randomly chosen unions is $O(n \log n)$.

Key words. probabilistic method, random graphs, analysis of algorithms, expected time bounds, Union-Find problem, equivalence algorithms, disjoint set union, Quick-Find algorithms

1. Introduction.

The *Union-Find* problem of Aho, Hopcroft, and Ullman [1] consists of maintaining a representation of a partition of an n-set S in such a way that two operations called *Find*$[x]$ and *Union*$[x, y]$ can be implemented efficiently. Suppose that *Name* is an injective mapping that associates with each equivalence class of the partition an integer called its *name*. Then the result of *Find*$[x]$ for $x \in S$ is the name of the equivalence class that contains x, and the result of *Union*$[x, y]$ for $x, y \in S$ is a new partition of S in which the equivalence classes C_x and C_y, which contain x and y, respectively, are merged into one equivalence class $C_x \cup C_y$. Thus, the partition before the execution of *Union*$[x, y]$ is a refinement of the partition that results after the Union operation is carried out.

The Union-Find problem arises in several applications, so that efficient implementations are of some practical interest. One of the simplest algorithms proposed for the problem uses an array *Name*$[x]$ to store the name of the equivalence class that contains x, and for each name s a linked list $L[s]$ of all the elements of the equivalence class s. With such a representation, the Find operation consists of a direct access to the array *Name* and, hence, can be implemented in constant time. To implement *Union*$[x, y]$, where *Name*$[x] \neq$ *Name*$[y]$, one must set *Name*$[u] \leftarrow$ *Name*$[y]$ for each u in $L[$*Name*$[x]]$ and append $L[$*Name*$[x]]$ to $L[$*Name*$[y]]$. Hence, the cost of a Union operation is proportional to $|C_x|$, the cardinality of C_x. This algorithm, which appears in Aho, Hopcroft, and Ullman [1], was baptized as the *Quick-Find* algorithm (QF) by Yao [27]. By maintaining a separate array $N[s]$ with the cardinality of the equivalence class s and changing the name of the elements of the smaller of the two classes, the cost of *Union*$[x, y]$ can be reduced to $\min(|C_x|, |C_y|)$. This version of the algorithm is known as the *Weighted Quick-Find* (QFW) algorithm, again following Yao [27]. It is usually assumed that both algorithms start with the finest partition of S in which each equivalence class is a singleton.

It is easily seen [1] that the worst-case cost of $(n-1)$ Unions that merge all the equivalence classes into one using QF is $n(n-1)/2 \cong n^2/2$. Using QFW reduces this cost to $n \log_2 n$. The

*Received by the editors December 16, 1987; accepted for publication (in revised form) July 6, 1992. A preliminary version of this work appeared in the Proceedings of the Seventeenth Annual Symposium on the Theory of Computing, 1985.

†University of Cambridge, Department of Pure Mathematics and Mathematical Statistics, 16 Mill Lane, Cambridge, England, CB2 1SB, and Department of Mathematics, Louisiana State University, Baton Rouge, Louisiana 70803.

‡Department of Mathematics and Computer Science, California State University, Hayward, California 94542-3092. This author's research was supported by CNPq grant 201.630-82-CC.

worst-case behavior of other Union-Find algorithms has been also investigated extensively by Hopcroft and Ullman [21], Tarjan [24], [25], and Tarjan and Van Leeuwen [26]. The best algorithms run in nonlinear time in the worst case. If the structure of the Unions is known in advance, Gabow and Tarjan [19] give a linear time algorithm.

The average case behavior is less well known and was examined by Doyle and Rivest [14], Yao [27], and Knuth and Schönhage [22], under three different input models. Of these the most natural approach seems to be the one based on Random Graphs defined in Yao [27] and adopted in Knuth and Schönhage [22].

Roughly speaking, Knuth and Schönhage [22] have shown that under the Random Graph model QFW runs in $O(n)$ expected time for $(n-1)$ Unions, while the corresponding cost of QF is $n^2/8 + o(n^2)$. Knuth and Schönhage posed several questions concerning the average behavior of QF and QFW. For example, they present empirical data that suggest that $2n$ might be an upper bound on the expected running time $T(n)$ of QFW, and prove that $T(n)/n$ is bounded, but give no explicit bound on $\lim_{n\to\infty} T(n)/n$. They also left open whether $T(n) = cn + o(n)$ for some constant c. In this paper, we prove that $T(n) = cn + o(n/\log n)$, where $c = 2.0847\ldots$, thus answering all the above questions.

Furthermore, among other results, we show that QFW is linear over the entire range. In fact, if $l = o(n)$ then the cost of l unions is $l(1 + o(1))$. This settles another open question of Knuth and Schönhage [22]. Concerning QF, we prove that the expected cost of $(n-1)$ Unions is $n^2/8 + O(n(\log n)^2)$, improving on the error term. We also show that the cost of $n/2 - n^{2/3}(\log n)^{2/3}$ Unions using QF is $O(n \log n)$. Hence, even QF is reasonably efficient on the average if we do not perform too many Unions. Of course, the worst-case cost for $n/2 - n^{2/3}(\log n)^{2/3}$ Unions for QF is still $\theta(n^2)$.

To derive the above results, we make use of techniques from Random Graph Theory. This theory was founded by Erdös and Rényi [15], [17], who were also the first to study the evolution of a random graph [16], [18]. In this paper, we rely on recent refinements and extensions of their theorems obtained by Bollobás [9], [10]. The application of these techniques to the analysis of algorithms is of independent interest and is particularly instructive in this case because of the precision of the results. There are a growing number of applications of the probabilistic method to the design and analysis of algorithms. See, for example, [2]–[5], [12], [13], [20], [22], [23], [27].

In §2, we define the basic concepts from Random Graph Theory and establish the connection of QF and QFW with random graphs. In §3, we analyze the expected cost of QFW for $(n-1)$ randomly chosen Unions. In §4, we study QFW for small t. The analysis of the unweighted algorithm is presented in §5.

2. Random graphs and quick-find. Let $N = \binom{n}{2}$ and $V = \{1, 2, \ldots, n\}$. As in [10], a *graph process* on V is a sequence $(G_t)_o^N$ such that (i) G_t is a graph on V with t edges and (ii) $G_0 \subset G_1 \subset \ldots \subset G_N$. The set of all $N!$ graph processes on V is denoted by $\tilde{\mathcal{G}}$. We turn $\tilde{\mathcal{G}}$ into a probability space by assigning the same probability $(N!)^{-1}$ to each graph process. A graph process $\tilde{G} \in \tilde{\mathcal{G}}$ is then called a *random graph process*. The sequence of graphs $\tilde{G} = (G_t)_o^N$ is often called the *evolution of a random graph*.

An execution of a sequence of Unions using QF or QFW can be associated in a natural way with a graph process: the vertices in V represent the elements of S, and the execution of the tth *Union*$[x, y]$ operation corresponds to adding the edge $\{x, y\}$ to G_{t-1} to obtain G_t. This way the partition after t such operations is given by the vertex sets of the connected components of G_t. The *cost* of the tth edge is the cost of the corresponding Union operation. Hence, for QF this cost is $(k + l)/2$, and for QFW it is $\min\{k, l\}$, provided that the edge joins two distinct components of G_{t-1} of sizes k and l, respectively. If the tth edge falls within an existing component of G_{t-1}, then no merge occurs, so we define the cost of such an edge to be zero. The *cost of the graph process* is the sum of the costs of its edges. In this way, the cost is a random variable (r.v.) on $\tilde{\mathcal{G}}$. We are interested in the expected value of this r.v.

We remark that $\tilde{\mathcal{G}}$ is one of the most widely used models in the theory of random graphs. It gives the most detailed information about a random graph. The model $\tilde{\mathcal{G}}$ is close to the Poisson process model used by Knuth and Schönhage in [22], but it is not quite the same. In fact, results about random graph processes can be readily translated to the Poisson process model, but the converse is not true.

In addition to $\tilde{\mathcal{G}}$, we shall need two other probability spaces over graphs:

(i) the space $\mathcal{G}_M$ that consists of all graphs on V with exactly M edges, where $0 \leq M \leq N$ and all its $\binom{N}{M}$ members are equiprobable;

(ii) the space $\mathcal{G}_p$ that consists of all 2^N graphs on V in which each of the N possible edges are chosen independently with probability p. Thus, a given graph with m edges occurs in $\mathcal{G}_p$ with probability $p^m(1-p)^{N-m}$.

For basic properties of these models the reader is referred to [10]. Clearly, the probability that a random graph $G_M \in \mathcal{G}_M$ has a property Q is the same as the probability that a random graph process $\tilde{G} = (G_t)_o^N \in \tilde{\mathcal{G}}$ is such that G_t has Q for $t = M$. Also, the expected number of edges of a random graph $G_p \in \mathcal{G}_p$ is pN, and loosely speaking, if M is "close" to pN, then G_M and G_p are "close."

We denote the expectations of a r.v. X by $E(X)$. Usually our random variables (r.v.'s) will be parameters of a random graph or graph process, and since we work with all the above models, to avoid confusion, we denote by $E_p(X)$ the expectation in $\mathcal{G}_p$, and by $E_M(X)$ the expectation in $\mathcal{G}_M$.

3. The weighted case. Let $\mathcal{G}^n$ be the set of all graphs on $V = \{1, 2, \ldots, n\}$. Let $\{z_G : G \in \mathcal{G}^n\}$ be a collection of real valued functions, where the domain of z_G is $V^{(2)} \backslash E(G)$, the set of nonedges of G. For a graph process $\tilde{G} = (G_t)_o^N$ set

$$Z(\tilde{G}) = \sum_{t=1}^{N} z_{G_{t-1}}(e_t),$$

where the set of edges of G_t is $E(G_t) = \{e_1, e_2, \ldots, e_t\}$. Furthermore, for $G \in \mathcal{G}^n$, denote by $z_a(G)$ the average of $z_G(e)$ as e runs over $V^{(2)} \backslash E(G)$ and set

$$Z_a(\tilde{G}) = \sum_{t=0}^{N-1} z_a(G_t).$$

If $\tilde{G}$ is a random graph process, then both $Z(\tilde{G})$ and $Z_a(\tilde{G})$ are r.v.'s. Furthermore, their expected value in $\tilde{\mathcal{G}}$ is the same, i.e.,

$$E(Z(\tilde{G})) = E(Z_a(\tilde{G})). \tag{1}$$

On the other hand, $z_a(G_t)$ is a r.v. on $\mathcal{G}_t$ and

$$E(Z_a(\tilde{G})) = \sum_{t=0}^{N-1} E_t(z_a(G_t)). \tag{2}$$

We shall consider two such collections of functions and their associated r.v.'s. For $G \in \mathcal{G}^n$ we define $w_G : V^{(2)} \backslash E(G) \to \mathbb{R}$ by

$$w_G(e) = \begin{cases} k & \text{if } e \text{ joins a component of } G \text{ of size } k \text{ to another component of size at least } k, \\ 0 & \text{otherwise.} \end{cases}$$

Hence, relative to QFW, $w_G(e)$ is the cost of adding the edge e to G, $w_a(G)$ is the average cost of adding an edge to G, and $W(\tilde{G})$ is the cost of the graph process $\tilde{G}$. Similarly, for $e \in V^{(2)} \setminus E(G)$ and $k_o = \lfloor 2(\ln n)^{13} \rfloor$ we let

$$w'_G(e) = \begin{cases} k & \text{if } e \text{ joins a component of } G \text{ of size } k \le k_o \text{ to another component of size at least } k, \\ 0 & \text{otherwise.} \end{cases}$$

By (1) and (2) the expected cost of a random graph process is

$$E(W(\tilde{G})) = \sum_{t=0}^{N-1} E_t(w_a(G_t)). \tag{3}$$

Our first aim is to show that, in fact,

$$E(W(\tilde{G})) = \sum_{t=0}^{N-1} E_t(w'_a(G_t)) + o(n/\log n). \tag{4}$$

This we do by relying on Theorem 1, an easy consequence of results in Bollobás [9], [10].

Let us call a component of a graph of order n *large* if it has order at least $n^{2/3}$. Otherwise, we call the component *small*. As it turns out, almost every graph process is such that, except for t close to $n/2$, small components are very small, and a little after $n/2$ there is only one large component.

THEOREM 1. *Le $s_o = \lfloor n(\ln n)^{-6} \rfloor$, $t_o = \lfloor n/2 + s_o \rfloor$, and $t = \lfloor n/2 + s \rfloor$. Denote by $\tilde{\mathcal{G}}_o$ the set of all graph processes satisfying the following four conditions:*

(i) *For $|s| \ge s_o$ all the small components of G_t have at most $k_o = \lfloor 2(\ln n)^{13} \rfloor$ vertices.*

(ii) *For $s < 0$ and $|s| \ge s_o$ all the components of G_t are small.*

(iii) *For $t \ge t_o$, G_t has exactly one large component.*

(iv) *The large component of G_{t_o} has at most $5n(\ln n)^{-6}$ vertices.*

Then the probability that a random graph process is in $\tilde{\mathcal{G}}_o$ is at least $1 - o((\ln n)^{-2})$.

Proof. It suffices to show that each of the four conditions occurs (independently of the others) with probability at least $1 - o((\ln n)^{-2})$. The result then follows because for any events A, B the probability of $A \cap B$ is at least $1 - P(\overline{A}) - P(\overline{B})$.

By [10, Thm. VI.1, pp. 124–126] a.e. graph process $\tilde{G}$ is such that G_t has no component of order k with $k_o \le k \le n^{2/3}$ for $0 \le t \le \lfloor n/2 - s_o \rfloor$ and $t_o \le t \le n$. Actually, as remarked in [10], the bound $t \le n$ is only for convenience and can be easily discarded. Furthermore, by analyzing the proof, one can readily see that the event occurs with probability at least $1 - o(n^{-8/15})$; hence, (i) follows.

To see that (ii) holds, we observe that G_t for $t = 0$ has only small components. Moreover, for any t such that $0 \le t < \lfloor n/2 - s_o \rfloor$ if G_t has only small components, then by (i) the order of the components of G_t is bounded by k_o with probability $1 - o((\ln n)^{-2})$. Hence, the order of the components of G_{t+1} is at most $2k_o < n^{2/3}$, i.e., G_{t+1} has only small components as well. This proves (ii).

Condition (iii) follows immediately from [10, Thm. VI.8, pp. 135–136] and condition (iv) from [10, Thm. VI.9, p. 136] for a.e. graph process $\tilde{G}$. By examining the proof we get that (iii) and (iv) hold with probability at least $1 - o((\ln n)^{-2})$. $\square$

THEOREM 2. *The expected cost of a random graph process $\tilde{G}$ relative to* QFW *is given by*

$$E(W(\tilde{G})) = \sum_{t=0}^{N-1} E_t(w'_a(G_t)) + o(n/\log n).$$

Proof. For every graph process $\tilde{G}$ we have $0 \le W'(\tilde{G}) \le W(\tilde{G}) \le n \log_2 n$. Hence,

$$0 \le E(W(\tilde{G}) - W'(\tilde{G}) | \tilde{G} \notin \tilde{\mathcal{G}}_o) \le n \log_2 n. \tag{5}$$

On the other hand, if $\tilde{G} \in \tilde{\mathcal{G}}_o$, then by Theorem 1,

$$W(\tilde{G}) - W'(\tilde{G}) = \sum_{|t-n/2|<s_o} \{w_{G_{t-1}}(e_t) | w'_{G_{t-1}}(e_t) \ne w_{G_{t-1}}(e_t)\}. \tag{6}$$

If $w'_{G_{t-1}}(e_t) \ne w_{G_{t-1}}(e_t)$, then e_t is an edge of the large component of G_{t_o}. Hence, (6) is bounded by $m \log_2 m$, where m is the order of the large component of G_{t_o}. Since $m \le 5n(\ln n)^{-6}$, it follows by Theorem 1 that when $\tilde{G} \in \tilde{\mathcal{G}}_0$,

$$E(W(\tilde{G}) - W'(\tilde{G}) | \tilde{G} \in \tilde{\mathcal{G}}_o) = O(n(\ln n)^{-5}). \tag{7}$$

From (5) and (7) by Theorem 1,

$$\begin{aligned} E(W(\tilde{G}) - W'(\tilde{G})) &= E(W(\tilde{G}) - W'(\tilde{G}) | \tilde{G} \notin \tilde{\mathcal{G}}_o) \cdot P(\tilde{G} \notin \tilde{\mathcal{G}}_o) \\ &+ E(W(\tilde{G}) - W'(\tilde{G}) | \tilde{G} \in \tilde{\mathcal{G}}_o) \cdot P(\tilde{G} \in \tilde{\mathcal{G}}_o) = o(n/\log n). \end{aligned} \tag{8}$$

The theorem follows from (8) by (1) and (2). □

It is usually more pleasant to work in $\mathcal{G}_p$ than in $\mathcal{G}_t$. In this case, we may do so [10, Thm. II.4, p. 36], which we state below without proof as Lemma 3.

LEMMA 3. *Let X be a graph parameter, (i.e., a real-valued function on a graph). Then,*

$$\sum_{t=0}^{N} E_t(X) = (N+1) \int_0^1 E_p(X) dp. \qquad \square$$

By Lemma 3, we have

$$\sum_{t=0}^{N-1} E_t(w'_a(G_t)) = (N+1) \int_0^1 E_p(w'_a(G_p)) dp. \tag{9}$$

Furthermore, for $p \ge 5 \ln n/n$, the probability that G_p is connected is at least $1 - O(n^{-4})$. Therefore,

$$(N+1) \int_{5\ln n/n}^1 E_p(w'_a(G_p)) dp \le n^2 \cdot n \cdot O(n^{-4}) = o(n^{-1/2}). \tag{10}$$

Hence, it suffices to compute $E_p(w'_a(G_p))$ for $p \le 5 \ln n/n$. In Theorem 4, we estimate the expected number of components of order k in G_p for $p \le 5 \ln n/n$ and $k \le k_o$.

Let $C(k, d)$ denote the number of labeled connected graphs on k vertices with $(k + d)$ edges. Following [9] and [10] let us call a component with k vertices and $(k + d)$ edges a (k, d)-*component*, and let $X(k, d)$ be the number of (k, d)-components of a graph. Let $X(k)$ be the number of components of order k and $T(k)$ be the number of tree components of order k.

THEOREM 4. *For $p \le 5 \ln n/n$, $k \le k_0 = \lfloor 2(\ln n)^{-13} \rfloor$, and $c = pn$,*

$$\begin{aligned} E_p(T(k)) &= \frac{k^{k-2}}{k!} \left(\frac{n}{c}\right) (ce^{-c})^k (1 + o(n^{-1/2})) \\ E_p(X(k)) &= E_p(T(k))(1 + o(n^{-1/2})). \end{aligned}$$

Proof. The probability that a fixed labeled connected graph with k vertices and $(k+d)$ edges is a component of G_p is

$$p^{k+d}(1-p)^{k(n-k)+\binom{k}{2}-k-d}.$$

It follows that

$$E_p(X(k,d)) = \binom{n}{k} C(k,d) p^{k+d}(1-p)^{k(n-k)+\binom{k}{2}-k-d}.$$

By elementary approximations we get,

$$E_p(X(k,d)) = \frac{C(k,d)}{k!}\left(\frac{c}{n}\right)^d (ce^{-c})^k(1+O((\ln n)^{27}/n)), \tag{11}$$

where the constant in the error term does not depend on k. For $d=-1$ we have $C(k,d)=k^{k-2}$, so

$$E_p(T(k)) = \frac{k^{k-2}}{k!}\left(\frac{n}{c}\right)(ce^{-c})^k(1+o(n^{-1/2})). \tag{12}$$

From [10, Thm. V.20, p. 125], [11] it follows that for some absolute constant c_1,

$$C(k,d) \le c_1 6^{-d} k^{k+(3d-1)/2}. \tag{13}$$

Since $X(k) = \Sigma\{X(k,d) | -1 \le d \le \binom{k}{2} - k\}$, the Theorem follows from (11), (12), and (13) by simple manipulations. □

To be able to compute $E_p(w_a'(G_p))$ we also need $E_p(X(k)X(l))$, and $E_p(X(k)^2)$ for $k \ne l$, such that $k, l \le k_o$ and $p \le 5\ln n/n$. These are estimated in Theorem 5.

THEOREM 5. *For $p \le 5\ln n/n$, $k \ne l$, and $k, l \le k_o = \lfloor 2(\ln n)^{13} \rfloor$ we have*

$$E_p(X(k)X(l)) = E_p(T(k))E_p(T(l))(1+o(n^{-1/2})),$$

and

$$E_p(X(k)(X(k)-1)) = (E_p(T(k)))^2(1+o(n^{-1/2})).$$

Proof. The proof is similar to that in [10, Thm. VI.2, pp. 127–128]. If $(k_1,d_1) \ne (k_2,d_2)$, then by [10, Eq. (5), p. 127],

$$E_p(X(k_1,d_1)X(k_2,d_2)) = E_p(X(k_1,d_1))E_p(X(k_2,d_2))\frac{(n)_{k_1+k_2}}{(n)_{k_1}(n)_{k_2}}(1-p)^{-k_1k_2}.$$

By elementary manipulations, letting $c = pn$,

$$\begin{aligned}\delta(p,k_1,k_2,n) &= \frac{(n)_{k_1+k_2}}{(n)_{k_1}(n)_{k_2}}(1-p)^{-k_1k_2}\\ &= \exp\left\{-\frac{(k_1+k_2)^2}{2n}+\frac{k_1^2}{2n}+\frac{k_2^2}{2n}+\frac{c\,k_1k_2}{n}\right\}\left(1+O\left(\frac{(k_1+k_2)^3}{n^2}\right)\right)\\ &\le 1+\delta'(n), \quad \text{where} \quad \delta'(n) = o(n^{-1/2}).\end{aligned}$$

Hence, by Theorem 4 for $k \neq l$,

$$E_p(X(k)X(l)) = E_p(T(k))E_p(T(l))(1 + o(n^{-1/2})).$$

Similarly,

$$E_p(X(k)(X(k) - 1)) = (E_p(T(k)))^2(1 + o(n^{-1/2})). \qquad \square$$

Theorems 4 and 5 enable us to estimate $E_p(w'_a(G_p))$ for $p \leq 5 \ln n/n$.

THEOREM 6. *For $p \leq 5 \ln n/n$ and $k_o = \lfloor 2(\ln n)^{13} \rfloor$ we have,*

$$\begin{aligned} E_p(w'_a(G_p)) = \frac{1}{N} \Bigg\{ & \sum_{k=1}^{k_o} nk^2 E_p(T(k)) \\ & - \sum_{1 \leq l < k \leq k_o} k^2 l E_p(T(k)) E_p(T(l)) \\ & - \frac{1}{2} \sum_{k=1}^{k_o} k^3 E_p(T(k))^2 \\ & - \sum_{k=1}^{k_o} k^3 E_p(T(k)) \Bigg\} (1 + o(n^{-1/2})) + o(n^{-2}). \end{aligned}$$

Proof. Let $k \leq k_o$. The number of edges $e \in V^{(2)} \backslash E(G_p)$ such that $w'_{G_p}(e) = k$ is

$$\begin{aligned} & kX(k)\left(n - \sum_{l=1}^{k} lX(l)\right) + \frac{1}{2}k^2 X(k)(X(k) - 1) \\ & \quad = nkX(k) - \sum_{1 \leq l < k} klX(k)X(l) - \frac{1}{2}k^2 X(k)^2 - \frac{1}{2}k^2 X(k). \end{aligned}$$

Hence, the average cost of adding an edge to G_p, measured by w'_{G_p} is

$$w'_a(G_p) = \frac{\displaystyle\sum_{1 \leq k \leq k_o} nk^2 X(k) - \sum_{1 \leq l < k \leq k_o} k^2 l X(k) X(l) - \frac{1}{2} \sum_{1 \leq k \leq k_o} k^3 X(k)^2 - \frac{1}{2} \sum_{1 \leq k \leq k_o} k^3 X(k)}{N - e(G_p)}, \tag{14}$$

where $e(G_p)$ is the number of edges of G_p. The distribution of $e(G_p)$ is binomial with parameters N and p. Hence, for $p \leq 5 \ln n/n$, the probability that $e(G_p) > 5n \ln n \geq 2(5 \ln n/n)N$ is exponentially small; in particular, it is $o(n^{-3})$. Furthermore,

$$w'_a(G_p) \leq k_o \tag{15}$$

for every G_p. Let us denote the event $e(G_p) \leq 5n \ln n$ by R. Then

$$\begin{aligned} E_p(w'_a(G_p)) &= E_p(w'_a(G_p)|R)P(R) + E_p(w'_a(G_p)|\overline{R})P(\overline{R}) \\ &= E_p(w'_a(G_p)|R) + o(k_o n^{-3}) \\ &= E_p(w'_a(G_p)|R) + o(n^{-2}). \end{aligned} \tag{16}$$

On the other hand, if R holds, then $N - e(G_p) = N(1 - o(n^{-1/2}))$, so from (14) we get

$$\begin{aligned} E_p(w'_a(G_p)|R) = \frac{1}{N} \Bigg\{ & \sum_{1 \leq k \leq k_o} nk^2 E_p(X(k)|R) - \sum_{1 \leq l < k \leq k_o} k^2 l E_p(X(k)X(l)|R) \\ & - \frac{1}{2} \sum_{1 \leq k \leq k_o} k^3 E_p(X(k)^2|R) - \frac{1}{2} \sum_{1 \leq k \leq k_o} k^3 E_p(X(k)|R) \Bigg\} (1 + o(n^{-1/2})). \end{aligned} \tag{17}$$

Since $X(k) \leq n$ for every G_p, arguing as in (16) we obtain

$$E_p(w'_a(G_p)|R) = \frac{1}{N}\left\{\sum_{1\leq k\leq k_o} nk^2E_p(X(k)) - \sum_{1\leq l<k\leq k_o} k^2lE_p(X(k)X(l))\right.$$

$$\left. -\frac{1}{2}\sum_{1\leq k\leq k_o} k^3E_p(X(k)^2) - \frac{1}{2}\sum_{1\leq k\leq k_o} k^3E_p(X(k))\right\}(1+o(n^{-1/2}))+o(n^{-2}). \tag{18}$$

Finally, from (18) we get the result by Theorems 4 and 5. □

We now have all the elements needed for the analysis of QFW. From Theorems 4 and 6 we get an explicit formula for $E_p(w'_a(G_p))$. Integrating this formula, we obtain by Theorem 2, (9) and (10) the expected cost of a random graph process.

THEOREM 7. *The expected cost of a random graph process $\tilde{G}$ relative to* QFW *is given by*

$$E(W(\tilde{G})) = c_on + o(n/\log n),$$

where

$$\begin{aligned} c_o &= \ln 2 - 1 + \sum_{k\geq 1}\left(\frac{1}{k} - \frac{k^k}{k!}\sum_{l=1}^{k-1}\frac{l^{l-1}}{l!}\,\frac{(k+l-2)!}{(k+l)^{k+l-1}}\right) \\ &= 2.0847\ldots . \end{aligned}$$

Proof. By Theorem 2, (9) and (10) we have

$$E(W(\tilde{G})) = (N+1)\int_0^{5\ln n/n} E_p(w'_a(G_p))dp + o(n/\log n). \tag{19}$$

By Theorems 4 and 6,

$$E_p(w'_a(G_p)) = \frac{1}{N}(u_1 - u_2 - u_3 - u_4)(1+o(n^{-1/2})) + o(n^{-2}), \tag{20}$$

where

$$u_1 = n^2\sum_{k=1}^{k_o}\frac{k^k}{k!}\,\frac{1}{c}(ce^{-c})^k, \tag{21}$$

$$u_2 = n^2\sum_{1\leq l<k\leq k_o}\frac{k^k}{k!}\,\frac{l^{l-1}}{l!}\,\frac{1}{c^2}(ce^{-c})^{k+l}, \tag{22}$$

$$u_3 = n^2\sum_{k=1}^{k_o}\frac{k^{2k-1}}{(k!)^2}\,\frac{1}{2c^2}(ce^{-c})^{2k}, \quad \text{and} \tag{23}$$

$$u_4 = n\sum_{k=1}^{k_o}\frac{k^{k+1}}{k!}\,\frac{1}{c}(ce^{-c})^k. \tag{24}$$

Since $c = pn$, integrating by parts we get that for a positive integer a and a real b,

$$\int x^ae^{-bx}dx = -e^{-bx}\left(\frac{x^a}{b} + \frac{ax^{a-1}}{b^2} + \frac{(a)_2x^{a-2}}{b^3} + \cdots + \frac{a!}{b^{a+1}}\right), \tag{25}$$

We have from (21),

$$\int_0^{5\ln n/n} u_1 dp = \frac{1}{n}\int_0^{5\ln n} u_1 dc = n\sum_{k=1}^{k_o}\frac{k^k}{k!}\int_0^{5\ln n} c^{k-1}e^{-ck}dc$$

$$= n\sum_{k=1}^{k_o}\frac{k^k}{k!}\,\frac{(k-1)!}{k^k} + o(1)$$

$$(26)\qquad = n\sum_{k=1}^{k_o}\frac{1}{k} + o(1).$$

Similarly, from (22) to (24),

$$(27)\qquad \int_0^{5\ln n/n} u_2 dp = n\sum_{1\le l<k\le k_o}\frac{k^k}{k!}\,\frac{l^{l-1}}{l!}\,\frac{(k+l-2)!}{(k+l)^{k+l-1}} + o(1),$$

$$\int_0^{5\ln n/n} u_3 dp = n\sum_{k=1}^{k_o}\frac{(2k-2)!}{(k!)^2}2^{-2k} + o(1)$$

$$= n\sum_{k\ge 1}\binom{2k}{k}2^{-2k}\frac{1}{2k(2k-1)} + o(n/\log n)$$

$$(28)\qquad = n(1-\ln 2) + o(n/\log n),$$

and

$$(29)\qquad \int_0^{5\ln n/n} u_4 dp = k_o = \lfloor 2(\ln n)^{13}\rfloor = o(n/\log n).$$

The Theorem follows from (19), (20), and (26) to (29). □

4. The weighted case for small t. In this section, we analyze the behavior of QFW when $0 \le t \le \alpha n/2$ for $0 < \alpha \le e^{-2}$. We shall show that for t in this range, the average cost of adding an edge to G_t is at most $1 + \epsilon(\alpha) + o(1)$ for almost every random graph process, where $\epsilon(\alpha) \to 0$ as $\alpha \to 0$, and $\epsilon(e^{-2}) \le .026$. To carry out the analysis we shall need a few preliminary results.

LEMMA 8. *Let* $0 < \alpha \le 1/7$, $p = \alpha/n$ *and* $1 \le k \le 100\ln n$. *Then*

(i) *For* $i = 0, 1$ *or* 2 $\frac{(k+1)^i E_p(T(k+1))}{k^i E_p(T(k))} \le \alpha e^{1-\alpha}$;

(ii) $\frac{(k+1)^2(E_p(T(k+1)))^2}{k^2(E_p(T(k)))^2} \le (\alpha e^{1-\alpha})^2$;

(iii) $E_p(X(k)) \le E_p(T(k))(1 + (\ln n)^2/n)$;

(iv) $E_p(T(k)) \le n^{-9}$ *if* $10\ln n \le k \le 100\ln n$.

Proof. As in Theorem 4,

$$(30)\qquad E_p(T(k)) = \binom{n}{k}k^{k-2}p^{k-1}(1-p)^{k(n-k)+\binom{k}{2}-k+1}.$$

Hence, for $i = 0, 1, 2$,

$$(31)\qquad \frac{(k+1)^i E_p(T(k+1))}{k^i E_p(T(k))} = (1-k/n)(1+1/k)^{k-2+i}pn(1-p)^{n-k-2} \le \alpha e^{1-\alpha}.$$

This shows (i).

Statement (ii) follows trivially from (i).

To see (iii), note that the expected number of components of order k with $k+d$ edges is

$$E_p(X(k,d)) = \binom{n}{k} C(k,d) p^{k+d} (1-p)^{k(n-k)+\binom{k}{2}-k-d}, \tag{32}$$

where, as earlier, $C(k,d)$ is the number of labeled connected graphs of order k with $k+d$ edges.

By (13), for some absolute constant c_1

$$E_p(X(k,d)) \le F(n,k,p,d) = c_1 \binom{n}{k} k^{k+(3d-1)/2} p^{k+d} (1-p)^{k(n-k)+\binom{k}{2}-k-d}. \tag{33}$$

Therefore,

$$\begin{aligned} E_p(X(k)) &= E_p(T(k)) + \sum_{d=0}^{\binom{k}{2}-k} E_p(X(k,d)) \\ &\le E_p(T(k)) + \sum_{d=0}^{\binom{k}{2}-k} F(n,k,p,d). \end{aligned} \tag{34}$$

Since, for $d \ge -1$,

$$\frac{F(n,k,p,d+1)}{F(n,k,p,d)} = k^{3/2} \frac{p}{1-p} \le \frac{(\ln n)^{7/4}}{n},$$

and from (30) and (33),

$$F(n,k,p,-1) = c_1 E_p(T(k)),$$

(iii) follows.

Finally, if $k \ge 10 \ln n$, then from (i)

$$E_p(T(k)) \le (\alpha e^{1-\alpha})^{k-1} E_p(T(1)) \le n^{-9},$$

for $0 < \alpha \le 1/7$. □

Denote by $L_1(G)$ the order of the largest component of G. Next we show that in our range $L_1(G_t)$ is at most $10 \ln n$ with very high probability.

THEOREM 9. *Let $0 < \alpha \le 1/7$ and denote by $\tilde{\mathcal{G}}_1$ the set of graph processes such that for $0 \le t \le \alpha n/2$ the order of the largest component of G_t is at most $10 \ln n$. Then the probability that a random graph process $\tilde{G}$ is in $\tilde{\mathcal{G}}_1$ is at least $1 - n^{-7}$.*

Proof. Let $0 \le t \le \alpha n/2$, $0 < \alpha \le 1/7$ and $p = p(t) = 2t/n^2$. By the analogue of [10, Thm. II.2.iii, p. 35],

$$E_t(X(k)) \le 3n^{1/2} E_p(X(k)). \tag{35}$$

Hence, by Lemma 8 (iii) and (iv), for $10 \ln n \le k \le 100 \ln n$,

$$E_t(X(k)) \le 3n^{-9}(1 + (\ln n)^2/n) n^{1/2} \le n^{-8\,1/4}. \tag{36}$$

Therefore,

$$\sum_{t=0}^{\alpha n/2} \sum_{k=10\ln n}^{20 \ln n} E_t(X(k)) \le 5\alpha \ln n\, n^{-7\,1/4} \le n^{-7}. \tag{37}$$

From (37) it follows that the probability that a random graph process is such that up to time $\alpha n/2$, $10 \ln n \leq L_1(G_t) \leq 20 \ln n$ is at most n^{-7}. Since $L_1(G_o) = 1$ and $L_1(G_{t+1}) \leq 2L_1(G_t)$, this implies that for $0 \leq t \leq \alpha n/2$

$$P(L_1(G_t) \geq 10 \ln n) \leq n^{-7}. \qquad \square$$

For a graph G, denote by $n_k = n_k(G)$ the number of vertices on components of order at least k. In our next result, we estimate $E_p(n_k)$.

THEOREM 10. *Let* $0 < \alpha \leq e^{-2}$ *and* $p = \alpha/n$. *Then for* $1 \leq k \leq 100 \ln n$,

(i) $E_p(n_1) = n$, *and*

(ii) $E_p(n_k) \leq (1 - \alpha e^{1-\alpha})^{-1} k E_p(T(k)) + 2(\ln n)^2$.

Proof. Clearly, $n_1 = n$, so $E_p(n_1) = n$, and for $k \geq 1$,

$$E_p(n_k) = \sum_{j \geq k} j E_p(X(j)). \tag{38}$$

Hence, by Lemma 8 (i) and (iii), for $1 \leq k \leq 100 \ln n$

$$\begin{aligned} E_p(n_k) &= \sum_{j=k}^{10 \ln n} j E_p(X(j)) + \sum_{j=10 \ln n}^{n} j E_p(X(j)) \\ &\leq \left\{ \sum_{j=k}^{10 \ln n} j E_p(T(j)) \right\} (1 + (\ln n)^2/n) + \sum_{j=10 \ln n}^{n} j E_p(X(j)) \\ &\leq (1 - \alpha e^{1-\alpha})^{-1} k E_p(T(k))(1 + (\ln n)^2/n) + \sum_{j=10 \ln n}^{n} j E_p(X(j)). \end{aligned} \tag{39}$$

Denote by $e(G_p)$ the number of edges of G_p. Then,

$$\begin{aligned} P(L_1(G_p) \geq 10 \ln n) \leq{} & P\left(L_1(G_p) \geq 10 \ln n | e(G_p) \leq 1.05\alpha \frac{n}{2}\right) \\ & + P\left(e(G_p) > 1.05\alpha \frac{n}{2}\right). \end{aligned} \tag{40}$$

By Theorem 9, since $1.05\alpha n/2 < (1/7)n/2$,

$$P(L_1(G_p) \geq 10 \ln n | e(G_p) \leq 1.05\alpha n/2) \leq n^{-7}, \tag{41}$$

and

$$P(e(G_p) > 1.05\alpha n/2) = P(e(G_p) > 1.05pN) \leq n^{-7}. \tag{42}$$

Hence, from (40) to (42)

$$P(L_1(G_p) \geq 10 \ln n) \leq 2n^{-7}. \tag{43}$$

Therefore,

$$\sum_{j=10 \ln n}^{n} j E_p(X(j)) \leq 2n^{-5}. \tag{44}$$

The result follows from (39) and (44). $\square$

Let $V(u) = \sum_{k=1}^{u} kT(k)$. Thus, $V(u)$ is the number of vertices on tree components of order at most u. In our next result, we show that $V(u)$ can be approximated by its expectation in $\mathcal{G}_p$.

THEOREM 11. *There is a constant n_o such that if $n \geq n_o$, $p = \alpha/n$, $0 < \alpha \leq e^{-2}$, and $1 \leq u \leq 100 \ln n$, then*

$$|V(u) - E_p(V(u))| < (\ln n)n^{1/2}$$

with probability at least $1 - 2/(\ln n)^2$.

Proof. The proof is similar to [10, Thm. V.9, pp. 100–101]. We estimate the variance of $V(u)$ and obtain the result by Chebyshev's inequality.

The expected number of ordered pairs of tree components of orders k and l, respectively, is clearly

$$\binom{n}{k}\binom{n-k}{l} k^{k-2} l^{l-2} p^{k+l-2} (1-p)^{(k+l)(n-k-l)+\binom{k+l}{2}-k-l+2}$$

$$= E_p(T(k))E_p(T(l)) \frac{(n-k)(n-k-1)\cdots(n-k-l+1)}{n(n-1)\cdots(n-l+1)}(1-p)^{-kl}, \tag{45}$$

because of (30). By elementary approximations we get for $1 \leq k \leq l \leq 100 \ln n$

$$\frac{(n-k)(n-k-1)\cdots(n-k-l+1)}{n(n-1)\cdots(n-l+1)}(1-p)^{-kl} < 1. \tag{46}$$

Hence, if $1 \leq k < l \leq 100 \ln n$, then

$$E_p(T(k) \cdot T(l)) < E_p(T(k))E_p(T(l)), \tag{47}$$

and if $1 \leq k = l \leq 100 \ln n$, then

$$E_p(T(k) \cdot (T(k) - 1)) < (E_p(T(k)))^2. \tag{48}$$

Therefore,

$$E_p((T(k))^2) < (E_p(T(k)))^2 + E_p(T(k)). \tag{49}$$

Since,

$$(V(u))^2 = \left(\sum_{k=1}^{u} kT(k)\right)^2 = \sum_{k=1}^{u} k^2 T(k)^2 + 2 \sum_{1 \leq k < j \leq u} k \cdot jT(k)T(j), \tag{50}$$

we get from (50) by (47) and (49),

$$\begin{aligned} E_p((V(u))^2) &< \sum_{k=1}^{u} k^2 (E_p(T(k)))^2 + 2 \sum_{1 \leq k < j \leq u} k \cdot jE_p(T(k))E_p(T(j)) \\ &\quad + \sum_{k=1}^{u} k^2 E_p(T(k)) \\ &= (E_p(V(u)))^2 + \sum_{k=1}^{u} k^2 E_p(T(k)). \end{aligned} \tag{51}$$

Therefore, by Lemma 8 (i),

$$\sigma_p^2(V(u)) \le \sum_{k=1}^{u} k^2 E_p(T(k)) \le 2n. \tag{52}$$

The result follows from (52) by Chebyshev's inequality. □

From Theorem 11 we obtain the following easy consequence.

COROLLARY 12. *Let* $0 < \alpha \le e^{-2}$, $1 \le k \le 10\ln n$, $0 \le t \le \alpha n/2$, *and* $p = 2t/n^2$. *Then with probability at least* $1 - 5/(\ln n)^2$, *a random* G_t *is such that*

$$n_k(G_t) \le E_p(n_k(G_p)) + 1.1(\ln n)n^{1/2}.$$

Proof. Let $U(k)$ be the number of vertices on components of order at most k, which are not trees, and let $p = 2t/n^2$. Then $n_k = n - V(k-1) - U(k-1) \le n - V(k-1)$, so by Lemma 8 (iii),

$$E_p(n_k) \ge n - E_p(V(k-1)) - (\ln n)^2. \tag{53}$$

Hence, by Theorem 11, with probability at least $1 - 2/(\ln n)^2$ a random G_p is such that

$$n_k(G_p) \le E_p(n_k(G_p)) + (\ln n)n^{1/2} + (\ln n)^2 \le E_p(n_k(G_p)) + 1.1(\ln n)n^{1/2}. \tag{54}$$

Since $n_k(G)$ is monotonically increasing, that is, if $H \subset G$, then $n_k(H) \le n_k(G)$, we have that (54) is a convex property. Hence, the result follows by [10, Thm. II.2 (ii), p. 35]. □

We can now show the main result of this section.

THEOREM 13. *Let* $0 < \alpha \le e^{-2}$. *Then with probability at least* $1 - 6/(\ln n)^2$ *a random graph process* $\tilde{G}$ *is such that for* $0 \le t \le \alpha n/2$ *the average cost of adding an edge to* G_t *by* QFW *is at most* $1 + \epsilon(\alpha) + c_2(\ln n)^2/n^{1/2}$, *where* c_2 *is an absolute constant,* $\epsilon(\alpha) \to 0$ *as* $\alpha \to 0$, *and* $\epsilon(e^{-2}) \le 0.026$.

Proof. We shall show that the result holds with

$$\epsilon(\alpha) = (1 - \alpha e^{1-\alpha})^{-2}(1 - (\alpha e^{1-\alpha})^2)^{-1}\alpha^2 e^{-4\alpha}.$$

The number of edges that join a component of order k to another component of order at least k is at most $\binom{n_k}{2}$. Hence the average cost of adding an edge to G_t by QFW is

$$w_a(G_t) \le \frac{\sum_{k\ge1}\binom{n_k}{2}}{N-t} \le \left[\sum_{k\ge1}\binom{n_k}{2}\bigg/\binom{n}{2}\right](1 + 1/n). \tag{55}$$

By Theorem 9 and Corollary 12 with probability at least $1 - 6/(\ln n)^2$ the graph process $\tilde{G}$ is such that for $0 \le t \le \alpha n/2$ and $L_1(G_t) \le 10\ln n$ and for $p = 2t/n^2$ and $1 \le k \le 10\ln n$ we have,

$$n_k(G_t) \le E_p(n_k) + 1.1(\ln n)n^{1/2}. \tag{56}$$

Hence, for such a $\tilde{G}$ we have $n_k = 0$ for $k > 10\ln n = u$. Therefore, from (55) we get

$$w_a(G_t) \le \left[\sum_{k=1}^{u}\binom{n_k}{2}\bigg/\binom{n}{2}\right](1 + 1/n). \tag{57}$$

Since $n_1 = n$,

$$w_a(G_t) \le \left[1 + \sum_{k=2}^{u} \binom{n_k}{2} \Big/ \binom{n}{2}\right](1 + 1/n)$$

$$\le \left[1 + \sum_{k=2}^{u} (n_k/n)^2\right](1 + 3/n). \tag{58}$$

Furthermore, $n_k \le n$; therefore, by (56),

$$w_a(G_t) \le 1 + \sum_{k=2}^{u} (E_p(n_k)/n)^2 + 23(\ln n)^2/n^{1/2}. \tag{59}$$

Now by Theorem 10, for $\alpha_t = pn = 2t/n$,

$$E_p(n_k) \le \left(1 - \alpha_t e^{1-\alpha_t}\right)^{-1} k E_p(T(k)) + 2(\ln n)^2. \tag{60}$$

Therefore, since $\alpha_t < \alpha \le e^{-2}$,

$$E_p(n_k) \le (1 - \alpha e^{1-\alpha})^{-1} k E_p(T(k)) + 2(\ln n)^2. \tag{61}$$

Let $f(\alpha) = (1 - \alpha e^{1-\alpha})^{-1}$. Then

$$(E_p(n_k))^2 \le (f(\alpha))^2 k^2 (E_p(T(k)))^2 + 4 f(\alpha) k E_p(T(k))(\ln n)^2 + 4(\ln n)^4. \tag{62}$$

Therefore, by Lemma 8, for $g(\alpha) = (1 - (\alpha e^{1-\alpha})^2)^{-1}$ we get,

$$\sum_{k=2}^{u} (E_p(n_k))^2 \le 4(f(\alpha))^2 g(\alpha)(E_p(T(2)))^2 + h_1(\alpha) n (\ln n)^2, \tag{63}$$

where $h_1(\alpha)$ is bounded.

From (30) by elementary approximations

$$E_p(T(2)) \le \frac{1}{2}\alpha e^{-2\alpha} n(1 + 1/n). \tag{64}$$

Consequently, for $\epsilon(\alpha) = (f(\alpha))^2 g(\alpha) \alpha^2 e^{-4\alpha}$ and $h_2(\alpha)$ bounded

$$\sum_{k=2}^{u} (E_p(n_k))^2 \le \epsilon(\alpha) n^2 + h_2(\alpha) n (\ln n)^2. \tag{65}$$

From (59) and (65)

$$w_a(G_t) \le 1 + \epsilon(\alpha) + c_2 (\ln n)^2/n^{1/2}, \tag{66}$$

where c_2 is an absolute constant at most 24. □

Armed with Theorem 13 we can easily derive $E_t(w_a)$ in the interval $0 \le t \le \alpha n/2$.

THEOREM 14. *Let* $0 < \alpha \le e^{-2}, 0 \le t \le \alpha n/2$. *Then the expected cost of adding an edge to* G_t *by* QFW *is at most* $1 + \epsilon(\alpha) + c_3/(\ln n)$, *where* c_3 *is an absolute constant,* $\epsilon(\alpha) \to 0$ *as* $\alpha \to 0$ *and* $\epsilon(e^{-2}) \le 0.026$.

Proof. Let $\tilde{\mathcal{G}}_1$ be as in Theorem 9 and denote $\tilde{\mathcal{G}}_2$ the set of graph processes such that for $0 \le t \le \alpha n/2$ and $p = 2t/n^2$,

$$n_k(G_t) \le E_p(n_k) + (\ln n) n^{1/2}. \tag{67}$$

Then

$$E_t(w_a) = E_t(w_a|\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \tilde{\mathcal{G}}_2)P(\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \tilde{\mathcal{G}}_2) + E_t(w_a|\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \overline{\tilde{\mathcal{G}}_2}) \cdot P(G \in \tilde{\mathcal{G}}_1 \cap \overline{\tilde{\mathcal{G}}_2}) + E_t(w_a|\tilde{G} \in \overline{\tilde{\mathcal{G}}_1})P(\tilde{G} \in \overline{\tilde{\mathcal{G}}_1}). \tag{68}$$

By Theorem 13,

$$E_t(w_a|\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \tilde{\mathcal{G}}_2) \leq 1 + \epsilon(\alpha) + c_2(\ln n)^2/n^{1/2}. \tag{69}$$

By Theorem 9 and Corollary 12,

$$E_t(w_a|\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \overline{\tilde{\mathcal{G}}_2})P(\tilde{G} \in \tilde{\mathcal{G}}_1 \cap \overline{\tilde{\mathcal{G}}_2}) \leq 50 \ln n/(\ln n)^2 = 50(\ln n)^{-1}. \tag{70}$$

By Theorem 9,

$$E_t(w_a|\tilde{G} \in \overline{\tilde{\mathcal{G}}_1})P(\tilde{G} \in \overline{\tilde{\mathcal{G}}_1}) \leq n \cdot n^{-7} = n^{-6}. \tag{71}$$

The result follows from (68) to (71). □

An immediate corollary of Theorem 14 is that if $l = o(n)$, then the expected cost of a random graph process by QFW up to time l is at most $l(1 + o(1))$.

5. The unweighted algorithm. Since the analysis of the unweighted algorithm is in many respects similar to that of §3, we shall give a less detailed account.

In QF, the cost of an edge that joins a component of order k to a component of order l is $\frac{1}{2}k + \frac{1}{2}l$. We shall say that $\frac{1}{2}k$ is the *contribution* of the component of order k and $\frac{1}{2}l$ is the contribution of the component of order l. Thus, to analyze QF we need to estimate the contributions of components of any given order. To do this we shall rely on the following consequence of results in [10], which summarizes the key facts needed for the analysis. As before, we refer to a component as large if it has at least $n^{2/3}$ vertices; otherwise, it is small. Throughout this section, when we mention cost it means cost in QF.

THEOREM 15. *Let $t = n/2 + s$ and $k_1(s) = 3(n^2/s^2)\ln n$ for $-n/2 \leq s \leq N - n/2$, and set $s_1 = n^{2/3}(\ln n)^{2/3}$ with $t_1 = n/2 + s_1$. Let $\tilde{\mathcal{G}}_3$ be the set of graph processes such that*

(i) *For $0 \leq t \leq n/2 - s_1$, all the components of G_t are small.*

(ii) *For $t = n/2 + s$ and $|s| > s_1$, all the small components of G_t have order at most $k_1(s)$.*

(iii) *For $t_1 \leq t \leq N$, the graph G_t has exactly one large component called the giant component.*

(iv) *The giant component of G_{t_1} has at most $5n^{2/3}(\ln n)^{2/3}$ vertices.*

(v) *For $t \geq 2n \ln n$, G_t is connected.*

Then the probability that a random graph process is in $\tilde{\mathcal{G}}_3$ is at least $1 - n^{-1}$.

The proof of Theorem 15 is analogous to that of Theorem 1. Since every graph process costs at most $n(n-1)/2$, an immediate consequence of Theorem 15 is that the graph processes that are not in $\tilde{\mathcal{G}}_3$ contribute at most $O(n)$ to the expected cost of a random graph process. Hence, we may restrict our attention to graph processes that are in $\tilde{\mathcal{G}}_3$. Furthermore, Theorem 15 suggests breaking down the expected cost of such graph processes into three major parts: the expected contribution of the giant component, the expected contribution of the small components, and the expected contribution of the large components during the *transition period*, i.e., the period that goes from time $n/2 - s_1$ to $n/2 + s_1$.

THEOREM 16. *For a random graph process in $\tilde{\mathcal{G}}_3$ the expected contribution of the large components during the transition period is at most $O(n(\ln n)^2)$.*

Proof. Let $\tilde{G}$ be the random graph process in $\tilde{\mathcal{G}}_3$. Since the order of a component that contains any given vertex never decreases with t, by Theorem 15 every vertex that is in a large component during the transition period is in the giant component by time $t_1 = n/2+s_1$. Hence, the number of vertices on large components in the transition period is at most $5n^{2/3}(\ln n)^{2/3}$. Consequently the probability that the edge at time t in the transition period joins a large component to another component is at most

$$\frac{5n^{2/3}(\ln n)^{2/3}n}{N-t} = O((\ln n)^{2/3}n^{-1/3}).$$

Since the transition period lasts $2s_1 = 2n^{2/3}(\ln n)^{2/3}$ time, the expected number of such edges is at most $O(n^{1/3}(\ln n)^{4/3})$. The contribution of the large component for each such edge is at most $O(n^{2/3}(\ln n)^{2/3})$. Hence, the theorem follows. □

Knuth and Schönhage [22] state that the expected contribution of the small components is $\sim \frac{1}{3}n\ln n$. The following is a stronger version of this result.

THEOREM 17. *The expected contribution of the small components of a random graph process is* $1/3n\ln n + O(n\ln\ln n)$.

Proof. As before, let $X(k)$ be the number of components of order k of a graph G of order n. Then the probability that the edge to be added to G_t joins a given component of order k to another component is $k(n-k)/(N-t)$. Hence, the expectation of S, the contribution of the small components, is

$$\begin{aligned} E(S) &= \sum_{t=0}^{N-1}\sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)E_t(X(k))/(N-t) \\ &= \sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)\sum_{t=0}^{N-1}E_t(X(k))/(N-t). \qquad (72) \end{aligned}$$

Since with probability at least $1 - O(n^{-5})$ a random G_t is connected for $t \geq 2n\ln n$, by Lemma 3

$$\begin{aligned} E(S) &= \left\{\sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)N^{-1}\sum_{t=0}^{2n\ln n}E_t(X(k))\right\}(1+O(\ln n/n)) + O(n^{-2}) \\ &= \left\{\sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)N^{-1}\sum_{t=0}^{N-1}E_t(X(k))\right\}(1+O(\ln n/n)) + O(n^{-2}) \\ &= \left\{\sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)\int_0^1 E_p(X(k))dp\right\}(1+O(\ln n/n)) + O(n^{-2}). \qquad (73) \end{aligned}$$

By the methods used in Theorem 4, for $1 \leq k \leq n^{2/3}/\ln n$

$$E_p(X(k)) \leq E_p(T(k))(1+c_3(\ln n)^{-3/2}) \qquad (74)$$

and for $1 \leq k \leq n^{2/3}$

$$E_p(X(k)) \leq c_4E_p(T(k)). \qquad (75)$$

Furthermore, for $1 \leq k \leq n^{2/3}$ and $p = c/n$,

$$E_p(T(k)) = \frac{k^{k-2}}{k!}\frac{n}{c}(ce^{-c})^k\exp\left[\frac{k^2}{2n}(c-1) - \frac{k^3}{3n^2}\right](1+O(n^{-1/3})). \qquad (76)$$

Hence, by (25), integrating (76) by parts we get,

$$(77)\qquad \int_0^1 E_p(T(k))dp = n^{-1}\int_0^\infty E_p(T(k))dc = k^{-3}e^{-k^3/3n^2}(1+O(n^{-1/3})).$$

Therefore,

$$(78)\qquad \left\{\sum_{k=1}^{n^{2/3}-1}\frac{1}{2}k^2(n-k)\int_0^1 E_p(T(k))dp\right\}(1+O(\ln n/n)) = \frac{1}{3}n\ln n + O(n).$$

Consequently, our result follows if we show that the contribution of small nontree components is $O(n\ln\ln n)$. To see this note first that by (73) and (78), in the range where (74) applies, the contribution of the small nontree components is at most

$$(79)\qquad O(n\ln n)O((\ln n)^{-3/2}) = O(n(\ln n)^{-1/2}) = O(n).$$

Therefore, by Theorem 15 and (75), it suffices to show that the contribution of small tree components from time $n/2 - s_2$ to time $n/2 + s_2$ is $O(n\ln\ln n)$, where $s_2 = n^{2/3}\ln n$.

For $\frac{n}{2} - s_2 \le t \le \frac{n}{2} + s_2$ and $1 \le k \le n^{2/3}$,

$$(80)\qquad E_t(T(k)) = \binom{n}{k}k^{k-2}\binom{N_1}{t-k+1}\Big/\binom{N}{t},$$

where $N_1 = \binom{n-k}{2}$, and $N = \binom{n}{2}$. Since $N_1/N \le \left(1-\frac{k}{n}\right)^2$, and by standard computations, for $t = \frac{n}{2} + s$,

$$\binom{N_1}{t-k+1}\Big/\binom{N}{t} = \frac{(N_1)_{t-k+1}(t)_{k-1}}{(N)_{t-k+1}(N-t+k-1)_{k-1}}$$

$$(81)\qquad \le c_3 n^{-(k-1)}e^{-k+k^2/2n}\cdot e^{(sk^2-2s^2k)/n^2}.$$

Furthermore, $e^{(sk^2-2s^2k)/n^2} \le e^{1/8}$; therefore, from (80) and (81) we obtain that,

$$(82)\qquad E_t(T(k)) \le c_4 nk^{-5/2}e^{-s^2k/n^2},$$

where c_4 is an absolute constant.

The probability that the edge at time t will join a given tree component of order k to another component is $2k(n-k)/n^2$. Therefore, by (82), the expected contributions of small tree components from time $n/2 - s_2$ to $n/2 + s_2$ is

$$\begin{aligned}
&\sum_{t=\frac{n}{2}-s_2}^{\frac{n}{2}+s_2}\sum_{k=1}^{n^{2/3}-1} E_t(T(k))\frac{k^2(n-k)}{n^2}\\
&\le \sum_{t=\frac{n}{2}-s_2}^{\frac{n}{2}+s_2}\sum_{k=1}^{n^{2/3}-1} c_4 k^{-1/2}e^{-s^2k/n^2}\\
&\le 2c_4\left\{\sum_{s=0}^{n^{2/3}}\sum_{k=1}^{n^{2/3}-1}k^{-1/2} + \sum_{s=n^{2/3}}^{s_2}\sum_{k=1}^{n^{2/3}-1}k^{-1/2}e^{-\frac{s^2k}{n^2}}\right\}\\
&= O(n) + O(n\ln\ln n) = O(n\ln\ln n). \qquad \square
\end{aligned}$$

Since by Theorem 15, with probability at least $1 - n^{-1}$, all components of a random graph process are small for $0 \le t \le n/2 - s_1$, it follows immediately from Theorem 17 that the expected cost of a random graph process up to time $n/2 - s_1$ is $O(n \ln n)$. To complete the analysis, we determine the expected contribution of the giant component in our next result.

THEOREM 18. *For a random graph process in $\tilde{\mathcal{G}}_3$ the expected contribution of the giant component is $n^2/8 + O(n(\ln n)^2)$.*

Proof. We shall denote the order of the largest component by L_1, the number of vertices on small components by Y, and, as before, the number of components of order k by $X(k)$. Thus, by Theorem 15, for a random graph process in $\tilde{\mathcal{G}}_3$ and for $t = n/2+s$, with $k_1(s) = 3n^2 \ln n/s^2$, we have

$$Y = \sum_{k=1}^{k_1(s)} kX(k) \tag{83}$$

and

$$L_1 = n - Y. \tag{84}$$

The probability that the edge to be added to G_t joins the giant component to another component is $L_1Y/(N - t)$. Hence, by Theorem 15, the expected contribution of the giant component at time t is

$$\begin{aligned} g(t) &= E_t(L_1^2 Y/(2(N-t))) \\ &= E_t\left\{\frac{n^2Y - 2nY^2 + Y^3}{n^2}\right\} + O(\ln n) \\ &= \frac{n^2E_t(Y) - 2nE_t(Y^2) + E_t(Y^3)}{n^2} + O(\ln n). \end{aligned} \tag{85}$$

Similarly to Theorem 5, we can show that for $s \geq s_1$ and $1 \leq k < l \leq k_1(s)$

$$E_t(X(k)X(l)) = E_t(X(k))E_t(X(l))\left(1 + O\left(\frac{skl}{n^2}\right)\right), \tag{86}$$

and for $1 \leq k \leq k_1(s)$,

$$E_t(X(k)^2) = E_t(X(k))^2\left(1 + O\left(\frac{sk^2}{n^2}\right)\right) + E_t(X(k)). \tag{87}$$

Therefore, by (83), (86), and (87),

$$E_t(Y^2) = E_t(Y)^2 + O(s/n^2)\left(\sum_{k=1}^{k_1(s)} k^2E_t(X(k))\right)^2 + \sum_{k=1}^{k_1(s)} k^2E_t(X(k)). \tag{88}$$

As in (82), we get by standard approximations

$$k^2E_t(X(k)) \geq c_5nk^{-1/2}\exp(-2s^2k/n^2), \tag{89}$$

for an absolute constant c_5. Hence,

$$\sum_{k=1}^{k_1(s)} k^2E_t(X(k)) = O(n^2/s), \tag{90}$$

so (88) and (90) imply

$$E_t(Y^2) = E_t(Y)^2 + O(n^2/s). \tag{91}$$

Similarly,

$$E_t(Y^3) = E_t(Y)^3 + O(n^3/s). \tag{92}$$

From (85), (91), and (92),

$$g(t) = \frac{n^2 E_t(Y) - 2n(E_t(Y))^2 + (E_t(Y))^3}{n^2} + O(\ln n) + O(n/s). \tag{93}$$

Since

$$\sum_{s=s_1}^{2n\ln n} O(\ln n + n/s) = O(n(\ln n)^2),$$

we may ignore the error term in (93). Furthermore, if $E_t(Y) = Y_o + y_o$, where $y_o = O(Y_o)$ and $y_o = O(n - Y_o)$, then

$$\begin{aligned}(Y_o + y_o)(n - Y_o - y_o)^2 - Y_o(n - Y_o)^2 &= O(y_o Y_o(n - Y_o)) + O(y_o(n - Y_o)^2)\\ &= O(y_o(n - Y_o)n) = O(y_o s \cdot n),\end{aligned} \tag{94}$$

because $n - Y_o = O(s)$. Hence, we may ignore the error term y_o as well, provided that

$$\sum_{s=s_1}^{2n\ln n} \frac{y_o s}{n} = O(n(\ln n)^2). \tag{95}$$

The expected number of vertices on small nontree components is $O(n^2/s^2)$. The expected number of vertices on small tree components is

$$E_t(T) = \frac{n}{c}\sum_{k=1}^{n^{2/3}} \frac{k^{k-1}}{k!}(ce^{-c})^k \left(1 + O\left(\frac{sk^2}{n^2}\right) + O\left(\frac{k}{n}\right)\right), \tag{96}$$

where $c = 1 + 2s/n$. But, if $s \geq s_1$, then

$$\frac{1}{c}\sum_{k=n^{2/3}+1}^{\infty} \frac{k^{k-1}}{k!}(ce^{-c})^k = O(1)\sum_{k=n^{2/3}+1}^{\infty} k^{-3/2}\left(\left(1 + \frac{2s}{n}\right)e^{-2s/n}\right)^k = o(n^{-M}) \tag{97}$$

for all $M > 0$. So

$$E_t(Y) = t(c)n + y_1 + y_2 + y_3 \tag{98}$$

where,

$$t(c) = \frac{1}{c}\sum_{k=1}^{\infty} \frac{k^{k-1}}{k!}(ce^{-c})^k, \tag{99}$$

$$y_1 = O(n^2/s^2), \tag{100}$$

$$y_2 = O(s/n)\sum_{k=1}^{n^{2/3}} \frac{k^{k+1}}{k!}(ce^{-c})^k = O(s/n)\sum_{k=1}^{n^{2/3}} k^{1/2}(ce^{1-c})^k, \tag{101}$$

and

$$y_3 = O(1)\sum_{k=1}^{n^{2/3}} \frac{k^k}{k!}(ce^{-c})^k = O(1)\sum_{k=1}^{n^{2/3}} k^{-1/2}(ce^{1-c})^k. \tag{102}$$

Clearly,

$$\sum_{s=s_1}^{2n\ln n} y_1 s/n = O(n\ln n) \tag{103}$$

and

$$\begin{aligned}\sum_{s=s_1}^{2n\ln n} y_2 s/n &= O(n^{-2})\sum_{s=s_1}^{2n\ln n} s^2 \sum_{k=1}^{n^{2/3}} k^{1/2}(ce^{1-c})^k \\ &= O(n^{-2})\sum_{k=1}^{n^{2/3}} k^{1/2} \sum_{s=s_1}^{2n\ln n} s^2((1+2s/n)e^{-2s/n})^k. \end{aligned}\tag{104}$$

Since,

$$\ln((1+2s/n)e^{-2s/n})^k = O(ks^2/n^2),$$

we have that

$$\sum_{s=s_1}^{2n\ln n} s^2((1+2s/n)e^{-2s/n})^k = O(1)\sum_{s=s_1}^{s_2} s^2 = O(s_2^3), \tag{105}$$

where $s_2 = O(nk^{-1/2})$. Hence, from (103) and (104) we get

$$\sum_{s=s_1}^{2n\ln n} y_2 s/n = O(n^{-2})\sum_{k=1}^{n} s_2^3 k^{1/2} = O(n)\sum_{k=1}^{n} 1/k = O(n\ln n). \tag{106}$$

Similarly,

$$\sum_{s=s_1}^{2n\ln n} y_3 s/n = O(n). \tag{107}$$

Therefore, from (93), (95), (103), (106), and (107) we get that $\tilde{g}$, the expected contribution of the giant component, is given by

$$\tilde{g} = \sum_{s=s_1}^{2n\ln n} g(t) = n\cdot\sum_{s=s_1}^{2n\ln n}(t(c) - 2t(c)^2 + t(c)^3) + O(n(\ln n)^2). \tag{108}$$

By Theorem 16, $\sum_{s=0}^{s_1} g(t) = O(n(\ln n)^2)$. Therefore,

$$\begin{aligned}\tilde{g} &= n\sum_{s=0}^{2n\ln n}(t(c) - 2t(c)^2 + t(c)^3) + O(n(\ln n)^2) \\ &= \frac{n^2}{2}\left[\int_1^\infty t(c)dc - 2\int_1^\infty t(c)^2dc + \int_1^\infty t(c)^3dc\right] + O(n(\ln n)^2).\end{aligned}\tag{109}$$

Therefore, our result follows if we show that

$$\int_1^\infty t(c)dc - 2\int_1^\infty t(c)^2 dc + \int_1^\infty t(c)^3 dc = \frac{1}{4}. \tag{110}$$

Indeed, from [10, p. 97, Eq. 5]

$$t(c) = u/c, \tag{111}$$

where $u = u(c)$ is the unique solution to the equation

$$ue^{-u} = ce^{-c} \tag{112}$$

in the interval $0 < u \le 1$.

From (112),

$$\ln u - u = \ln c - c. \tag{113}$$

Taking derivatives with respect to c we get

$$\frac{u'}{u} - u' = \frac{1}{c} - 1. \tag{114}$$

From (114) multiplying by u/c,

$$\frac{uu'}{c} = \frac{u}{c} + \frac{u'}{c} - \frac{u}{c^2} = \frac{u}{c} + \left(\frac{u}{c}\right)' \tag{115}$$

and similarly,

$$\frac{u^2u'}{c^2} = \frac{u^2}{c^2} + \frac{uu'}{c^2} - \frac{u^2}{c^3} = \frac{u^2}{c^2} + \frac{1}{2}\left(\frac{u^2}{c^2}\right)'. \tag{116}$$

Hence, from (111) and (115) integrating by parts,

$$\begin{aligned}\int_1^\infty t(c)^2 dc &= \int_1^\infty \frac{u^2}{c^2}dc = \left[-\frac{u^2}{c}\right]_1^\infty + 2\int_1^\infty \frac{uu'}{c}dc \\ &= 1 + 2\int_1^\infty \frac{uu'}{c}dc = 1 + 2\int_1^\infty \frac{u}{c}dc + 2\left[\frac{u}{c}\right]_1^\infty \\ &= 2\int_1^\infty t(c)dc - 1. \end{aligned} \tag{117}$$

Similarly, from (111), (116), and (117),

$$\begin{aligned}\int_1^\infty t(c)^3 dc &= \int_1^\infty \frac{u^3}{c^3}dc = \left[-\frac{u^3}{2c^2}\right]_1^\infty + \frac{3}{2}\int_1^\infty \frac{u^2u'}{c^2}dc \\ &= \frac{1}{2} + \frac{3}{2}\int_1^\infty \frac{u^2}{c^2}dc + \left[\frac{3u^2}{4c^2}\right]_1^\infty \\ &= 3\int_1^\infty t(c)dc - 7/4. \end{aligned} \tag{118}$$

Therefore, (110) follows from (117) and (118) by simple substitution, concluding the proof. ◻

Acknowledgments. Major parts of this research were carried out at the University of Cambridge, England, at the Instituto de Matemática, Estatística, e Ciência da Computação, Universidade Estadual de Campinas, Brazil, at Louisiana State University, and at the California State University, Hayward. We thank the anonymous referees for their helpful suggestions and for catching a number of typos and small mistakes in the original manuscript.

REFERENCES

[1] A V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974.

[2] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, *An $O(n \log n)$ Sorting Network*, Proc. Fifteenth Annual ACM Symposium on Theory of Computing, (1983), pp. 1–9.

[3] D. ANGULIN AND L. G. VALIANT, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[4] L. BABAI, P. ERDÖS, AND S. M. SELKOW, *Random graph isomorphism*, SIAM J. Comput., 9 (1980), pp. 628–635.

[5] J. L. BENTLEY, D. S. JOHNSON, F. T. LEIGHTON, C. C. MCGEOCH, AND L. A. MCGEOCH, *Some unexpected expected behavior results for bin packing*, Proc. Sixteenth Annual ACM Symposium on Theory of Computing, (1984), pp. 279–288.

[6] B. BOLLOBÁS, *Graph Theory, An Introductory Course*, Springer-Verlag, New York, 1979.

[7] ———, *Random graphs*, in Combinatorics, London Mathematical Society Lecture Notes, 52, Cambridge University Press, Cambridge, 1981, pp. 80–102.

[8] ———, *Lectures on Random Graphs*, Proc. Waterloo Silver Jubilee Conference, 1982.

[9] ———, *The evolution of random graphs*, Trans. Amer. Math. Soc., 286 (1984), pp. 257–274.

[10] ———, *Random Graphs*, Academic Press, New York, 1985.

[11] ———, *The evolution of sparse graphs*, in Graph Theory and Combinatorics, A Volume in Honour of Paul Erdös, B. Bollobás, ed., Academic Press, London, 1984, pp. 35–57.

[12] B. BOLLOBÁS, A. BRODER, AND I. SIMON, *The cost of clustering in random probing*, J. Assoc. Comput. Mach., 37 (1990), pp. 224–237.

[13] B. BOLLOBÁS AND A. THOMASON, *Parallel Sorting*, Discrete Appl. Math., 6 (1983), pp. 1–11.

[14] J. DOYLE AND R. RIVEST, *Linear expected time of a simple union-find algorithm*, Inform. Process Lett., 5 (1976), pp. 146–148.

[15] P. ERDÖS AND A. RÉNYI, *On random graphs* I, Publ. Math., Debrecen, 6 (1959), pp. 290–297.

[16] ———, *On the evolution of random graphs*, Publ. Math. Inst. Hungar. Acad. Sci., 5 (1960), pp. 17–61.

[17] ———, *On the strength of connectedness of a random graph*, Acta Math. Acad. Sci. Hungar., 12 (1961), pp. 261–267.

[18] ———, *On the evolution of random graphs*, Bull. Inst. Internat. Statist. Tokyo, 38 (1961), pp. 343–347.

[19] H. N. GABOW AND R. E. TARJAN, *A linear time algorithm for a special case of disjoint set union*, Proc. Fifteenth Annual ACM Symposium on Theory of Computing, 1983, pp. 246–251.

[20] L. J. GUIBAS AND E. SZEMERÉDI, *The analysis of double hashing*, J. Comput. System Sci., 16 (1978), pp. 226–274.

[21] J. E. HOPCROFT AND J. D. ULLMAN, *Set merging algorithms*, SIAM J. Comput., 2 (1973), pp. 294–303.

[22] E. E. KNUTH AND A. SCHÖNHAGE, *The expected linearity of a simple equivalence algorithm*, Theoret. Comput. Sci., 6 (1978), pp. 281–315.

[23] L. PÓSA, *Hamiltonian circuits in random graphs*, Discrete Math., 14 (1976), pp. 359–364.

[24] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.

[25] ———, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.

[26] R. E. TARJAN AND J. VAN LEEUWEN, *Worst-case analysis of set union algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 245–281.

[27] A. C. YAO, *On the average behavior of set merging algorithms*, Proc. Eighth Annual ACM Symposium on Theory of Computing, 1976, pp. 192–195.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 1075–1086, October 1993

A PSEUDORANDOM ORACLE CHARACTERIZATION OF BPP*

JACK H. LUTZ†

Abstract. It is known from work of Bennett and Gill [*SIAM J. Comput.*, 10 (1981), pp. 96–113] and of Ambos-Spies [in *Proc. 1st Structure in Complexity Theory Conference*, 1986, pp. 23–34] that the following conditions are equivalent:

(i) $L \in \mathrm{BPP}$.

(ii) For almost all oracles A, $L \in \mathrm{P}^A$.

It is shown here that the following conditions are also equivalent to (i) and (ii):

(iii) The set of oracles A for which $L \in \mathrm{P}^A$ has pspace-measure 1.

(iv) For *every* pspace-random oracle A, $L \in \mathrm{P}^A$.

It follows from this characterization (and its proof) that almost every $A \in \mathrm{ESPACE}$ is $\leq_T^{\mathrm{P}}$-hard for BPP^A. Succinctly, the main content of the proof is that pseudorandom generators exist relative to every pseudorandom oracle.

Key words. probabilistic complexity, BPP, resource-bounded measure, random oracles, pseudorandom oracles, pseudorandom generators

AMS subject classifications. 68Q15, 03D15

1. Introduction.

The class BPP consists of those decision problems that are feasibly solvable by randomized algorithms. This class, defined by Gill [9], has been shown to admit a variety of equivalent definitions [1], [2], [11]–[13], [27], [28], [30]–[32]. A particularly elegant and useful characterization of BPP is the following.

THEOREM 1 (Bennett and Gill [2], Ambos-Spies [1]). *For a language $L \subseteq \{0, 1\}^*$ the following conditions are equivalent*:

(1) $L \in \mathrm{BPP}$.

(2) *For almost all oracles A, $L \in \mathrm{P}^A$.*

The "almost all" in condition (2) here refers to Lebesgue measure on the set of all oracles. (Oracles in this paper are languages $A \subseteq \{0, 1\}^*$.) That is, if an oracle A is chosen probabilistically, by using an independent toss of a fair coin to decide whether each string $x \in \{0, 1\}^*$ is in A, then condition (2) asserts that $L \in \mathrm{P}^A$ with probability one.

Interesting though it is, this characterization demands a more careful analysis. Since BPP is countable, Theorem 1 implies that *almost every* oracle is $\leq_T^{\mathrm{P}}$-hard for BPP. Nevertheless, Theorem 1 gives no information regarding *which* oracles are $\leq_T^{\mathrm{P}}$-hard for BPP. (The inclusion $\mathrm{BPP} \subseteq \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ of Sipser and Gács [25] implies that oracles that are $\leq_T^{\mathrm{P}}$-hard for $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ have this property, but by Theorem 1 this is only a measure-0 set of oracles, unless $\mathrm{BPP} = \Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$.)

In this paper we refine Theorem 1 by proving the following.

MAIN THEOREM. *For a language $L \subseteq \{0, 1\}^*$ the following conditions are equivalent*:

(1) $L \in \mathrm{BPP}$.

(2) *The set of oracles A for which $L \in \mathrm{P}^A$ has* pspace-*measure* 1.

(3) *For every* pspace-*random oracle A, $L \in \mathrm{P}^A$.*

(*Conditions* (2) *and* (3) *here refer to the resource-bounded measure theory and measure-theoretic pseudorandomness of Lutz* [18]; *see* §3 *below for details.*)

Intuitively, the Main Theorem says that *every* sufficiently random oracle is $\leq_T^{\mathrm{P}}$-hard for BPP and that pspace-randomness is sufficient here. Of course, every random oracle (i.e., every language whose characteristic sequence is algorithmically random in the equivalent

*Received by the editors April 18, 1990; accepted for publication (in revised form) April 18, 1992. This research was supported in part by National Science Foundation grants CCR-8809238 and CCR-9157382 and also by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), where the author was visiting while part of this work was performed.

†Department of Computer Science, Iowa State University, Ames, Iowa 50011.

senses of Martin-Löf [20], Levin [14], Schnorr [23], Chaitin [5], [6], Solovay [26], and Shen′ [24]) is pspace-random, so that it follows immediately from the Main Theorem that every random oracle is $\leq_T^P$-hard for BPP. Since almost every oracle is random [20], this in turn gives the (1) $\Longrightarrow$ (2) part of Theorem 1. However, the Main Theorem is much stronger than this. For example, since every pspace-measure-1 set has measure 1 in ESPACE = DSPACE(2^{linear}) [18], the (1) $\Longrightarrow$ (2) part of the Main Theorem tells us that for each $L \in$ BPP, L is $\leq_T^P$-reducible to almost every $A \in$ ESPACE. Similarly, since almost every language in E$_2$SPACE = DSPACE($2^{\text{polynomial}}$) is pspace-random [18], the (1) $\Longrightarrow$ (3) part of the Main Theorem tells us that almost every language in E$_2$SPACE is $\leq_T^P$-hard for BPP. In fact, our proof tells us more, namely, that almost every language $A \in$ ESPACE is $\leq_T^P$-hard for BPP^A.

2. Overview of proof.

The following notion of hardness relative to oracle circuits is central to the proof of the Main Theorem.

DEFINITION (Nisan and Wigderson [21], [22]). Given languages $L, A \subseteq \{0,1\}^*$, a real $\delta > 0$, and $n, s \in \mathbf{N}$, L is *$(\delta, s)^A$-hard at n* if

$$\left|L(\gamma^A) \triangle L_{=n}\right| > 2^{n-1}(1-\delta)$$

for every n-input oracle circuit γ with size$(\gamma) \leq s$. (Here $L_{=n}$ denotes $L \cap \{0,1\}^n$.) The *hardness of L relative to A* is the function $H_L^A : \mathbf{N} \to \mathbf{N}$ defined by

$$H_L^A(n) = \max\{h \in \mathbf{N} \mid L \text{ is } (h^{-1}, h)^A\text{-hard at } n\}.$$

(See [19] or [29] for details concerning oracle circuits.)

Thus a language L is $(\delta, s)^A$-hard at n if γ^A computes L incorrectly on at least $50(1-\delta)$ percent of the inputs in $\{0,1\}^n$, whenever γ is an n-input oracle circuit of size s.

For each real $0 < \alpha \leq 1$ and each oracle $A \subseteq \{0,1\}^*$ define the *relativized hardness class*

$$\mathrm{H}_\alpha^A = \{L \subseteq \{0,1\}^* \mid H_L^A(n) > 2^{\alpha n} \text{ a.e.}\}.$$

(We say that a condition $\Theta(n)$ holds *almost everywhere* (a.e.) if it holds for all but finitely many $n \in \mathbf{N}$. We say that $\Theta(n)$ holds *infinitely often* (i.o.) if it holds for infinitely many $n \in \mathbf{N}$.) Also define

$$\mathrm{H}^A = \bigcap_{0<\alpha<\frac{1}{3}} \mathrm{H}_\alpha^A.$$

If E^A = DTIME^A(2^{linear}) contains a hard language, then this language can be used to construct a pseudorandom bit generator that is quick enough and secure enough to achieve P^A = BPP^A. That is, we have the following.

THEOREM 2 (Nisan and Wigderson [21], [22]). *For every oracle A and every* $0 < \alpha < 1$, *if* $\mathrm{E}^A \cap \mathrm{H}_\alpha^A \neq \emptyset$, *then* $\mathrm{P}^A = \mathrm{BPP}^A$.

The proof of Theorem 2, a relativization of arguments in [21], [22], will not be given here.

The following result, which is the main technical content of this paper, will be proven in §4.

THEOREM 3. $\mu_{\text{pspace}}(\{A \mid \mathrm{E}^A \cap \mathrm{H}^A \neq \emptyset\}) = 1$. (*That is, the indicated set of oracles has* pspace-*measure* 1.)

COROLLARY 4. $\mu_{\text{pspace}}(\{A \mid \mathrm{P}^A = \mathrm{BPP}^A\}) = 1$.

The proof of the Main Theorem is now easy. If condition (1) holds, then P^A = BPP^A implies $L \in$ P^A, so that condition (2) follows by Corollary 4. If condition (2) holds, then condition (3) holds because every pspace-random language is, by definition, an element of every pspace-measure-1 set [18]. Finally, almost every oracle A is pspace-random [18], so that condition (1) follows from condition (3) by the (2) $\Longrightarrow$ (1) part of Theorem 1. $\square$

The relationship between pseudorandom generators and pseudorandom oracles is a particularly interesting aspect of this proof. A *pseudorandom generator* is a function $G : \{0,1\}^* \times \mathbf{N} \to \{0,1\}^*$ such that $|G(x,n)| = n$ for all x and n. Given a function $l : \mathbf{N} \to \mathbf{N}$ and an oracle $A \subseteq \{0,1\}^*$, a generator G is *A-quick* and *A-secure* on seeds of length l, and we write $G : l \xrightarrow{A} n$ if (i) $G(x,n)$ is deterministically computable in $2^{O(|x|)}$ time relative to A whenever $|x| = l(n)$; and (ii) for every family $\gamma = (\gamma_n)$ of oracle circuits with $\text{size}(\gamma_n) = O(n)$ we have

$$\left|\Pr\left[\gamma_n^A(G(x,n)) = 1\right] - \Pr\left[\gamma_n^A(y) = 1\right]\right| < \frac{1}{n} \quad \text{a.e.},$$

where $x \in \{0,1\}^{l(n)}$ and $y \in \{0,1\}^n$ are chosen according to the uniform distributions.

The main part of Nisan and Wigderson's proof of Theorem 2 shows that for every real $0 < \alpha < 1$ there exists $c \in \mathbf{N}$ such that for all $A \subseteq \{0,1\}^*$ if $\mathrm{E}^A \cap \mathrm{H}_\alpha^A \neq \emptyset$, then there is a generator $G : c\log n \xrightarrow{A} n$. Putting this together with Theorem 3 gives the following.

THEOREM 5. *There is a positive integer c such that for every* pspace*-random oracle $A \subseteq \{0,1\}^*$ there exists a pseudorandom generator $G : c\log n \xrightarrow{A} n$.*

Less formally, this says that pseudorandom generators exist relative to every pseudorandom oracle.

3. pspace-measure and pspace-randomness.

In this section we review some fundamentals of resource-bounded measure and pseudorandomness, where the resource bound is polynomial space. For more details, examples, and proofs, see [18].

We work with two alphabets, the binary alphabet $\{0,1\}$ and the extended binary alphabet $\Sigma = \{0,1,\perp\}$. The symbol $\perp$ (bottom) denotes an undefined bit. We fix the partial ordering $\sqsubseteq$ of Σ in which $\perp \sqsubseteq 0$, $\perp \sqsubseteq 1$, and 0 and 1 are incomparable. Given a string or sequence $x \in \Sigma^* \cup \Sigma^\infty$, we write $x[i]$ for the ith bit of x and $x[i..j]$ for the string consisting of the ith through jth bits of x. We also fix the standard enumeration $s_0 = \lambda$, $s_1 = 0$, $s_2 = 1$, $s_3 = 00, \ldots$ of $\{0,1\}^*$, and we write $x[w] = x[i]$ whenever $w = s_i$ and $0 \le i < |x|$. We extend $\sqsubseteq$ bitwise to strings and sequences, i.e., $x \sqsubseteq y$ if and only if $(\forall i \in \mathbf{N}) x'[i] \sqsubseteq y'[i]$, where $x' = x$ if $|x| = \infty$, $x' = x\perp^\infty$ if $|x| < \infty$, and y' is defined similarly. The *cylinder specified by* a string $x \in \Sigma^*$ is $C_x = \{A \subseteq \{0,1\}^* \mid x \sqsubseteq \chi_A\}$, where $\chi_A \in \{0,1\}^\infty$ is the *characteristic sequence* of A, i.e., each $\chi_A[i]$ is 1 if $s_i \in A$ and is 0 otherwise. We use the symbol $\top$ (top) to specify the empty set, i.e., $C_\top = \emptyset$. For $x, y \in \Sigma^*$ we let $x \wedge y$ be the shortest string such that $C_{x\wedge y} = C_x \cap C_y$. Note that $x \wedge y = \top$ if x and y are *incompatible*, i.e., if $C_x \cap C_y = \emptyset$. The *measure* $\mu(x)$ of a cylinder C_x is the probability that $A \in C_x$ when $A \subseteq \{0,1\}^*$ is chosen according to the random experiment in which an independent toss of a fair coin is used to decide whether each string $w \in \{0,1\}^*$ is in A. Thus if we let $\#(b,x)$ denote the number of occurrences of the symbol b in the string x and if we define

$$\|x\| = \begin{cases} \#(0,x) + \#(1,x) & \text{if } x \in \Sigma^*, \\ \infty & \text{if } x = \top, \end{cases}$$

then $\mu(x) = 2^{-\|x\|}$ for all $x \in \Sigma^* \cup \{\top\}$.

We fix once and for all a one-to-one pairing function $\langle,\rangle$ from $\{0,1\}^* \times \{0,1\}^*$ onto $\{0,1\}^*$ such that the pairing function and its associated projections $\langle x,y\rangle \mapsto x$ and $\langle x,y\rangle \mapsto y$ are computable in polynomial time. We insist further that this pairing function satisfy the following condition for all $x, y \in \{0,1\}^*$: $\langle x,y\rangle \in \{0\}^*$ if and only if $x, y \in \{0\}^*$. This condition canonically induces a pairing function $\langle,\rangle$ from $\mathbf{N} \times \mathbf{N}$ onto $\mathbf{N}$. We write $\langle x,y,z\rangle$ for $\langle x,\langle y,z\rangle\rangle$, etc., so that tuples of any fixed length are coded by the pairing function.

We let $\mathbf{D} = \{m2^{-n} \mid m, n \in \mathbf{N}\}$ be the set of nonnegative *dyadic rationals*. Many functions in this paper take their values in $\mathbf{D}$ or in $[0, \infty)$, the set of nonnegative real numbers. In fact, with the exception of some functions that map into $[0, \infty)$, our functions are of the form $f : X \to Y$, where each of the sets X, Y is $\mathbf{N}$, $\{0, 1\}^*$, $\mathbf{D}$, or some Cartesian product of these sets. Formally, in order to have uniform criteria for their computational complexities, we regard all such functions as mapping $\{0, 1\}^*$ into $\{0, 1\}^*$. For example, a function $f : \mathbf{N}^2 \times \{0, 1\}^* \to \mathbf{N} \times \mathbf{D}$ is formally interpreted as a function $\tilde{f} : \{0, 1\}^* \to \{0, 1\}^*$. Under this interpretation $f(i, j, w) = (k, q)$ means that $\tilde{f}(\langle 0^i, \langle 0^j, w\rangle\rangle) = \langle 0^k, \langle u, v\rangle\rangle$, where u and v are the binary representations of the integer and fractional parts of q, respectively. Moreover, we care only about the values of $\tilde{f}$ for arguments of the form $\langle 0^i, \langle 0^j, w\rangle\rangle$, and we insist that these values have the form $\langle 0^k, \langle u, v\rangle\rangle$ for such arguments.

For a function $f : \mathbf{N} \times X \to Y$ and $k \in \mathbf{N}$ we define the function $f_k : X \to Y$ by $f_k(x) = f(\langle 0^k, x\rangle)$. We then regard f as a uniform enumeration of the functions $f_0, f_1, f_2, \ldots$. For a function $f : \mathbf{N}^n \times X \to Y$ $(n \geq 2)$ we write $f_{k,l} = (f_k)_l$, etc. For a function $f : \{0, 1\}^* \to \{0, 1\}^*$ we write f^n for the n-fold composition of f with itself.

We work with the resource bound

$$\text{pspace} = \{f : \{0, 1\}^* \to \{0, 1\}^* \mid f \text{ is computable in polynomial space}\}.$$

(The length $|f(x)|$ of the output *is* included as part of the space used in computing f.)

Resource-bounded measure and pseudorandomness were originally developed in terms of "modulated covering by cylinders" [15]–[17]. Athough the main results of these papers are true, the underlying development was technically flawed. This situation was remedied in [18], in which resource-bounded measure was reformulated in terms of density functions. We review relevant aspects of the latter formulation here.

A *density function* is a function $d : \{0, 1\}^* \to [0, \infty)$ satisfying

$$d(x) \geq \frac{d(x0) + d(x1)}{2}$$

for all $x \in \{0, 1\}^*$. The *global value* of a density function d is $d(\lambda)$. An n-dimensional *density system* (n-DS) is a function $d : \mathbf{N}^n \times \{0, 1\}^* \to [0, \infty)$ such that $d_{\vec{k}}$ is a density function for every $\vec{k} \in \mathbf{N}^n$. It is sometimes convenient to regard a density function as a 0-DS.

A *computation* of an n-DS d is a function $\widehat{d} : \mathbf{N}^{n+1} \times \{0, 1\}^* \to \mathbf{D}$ such that

$$\left|\widehat{d}_{\vec{k},r}(x) - d_{\vec{k}}(x)\right| \leq 2^{-r} \tag{3.1}$$

for all $\vec{k} \in \mathbf{N}^n$, $r \in \mathbf{N}$, and $x \in \{0, 1\}^*$. A pspace-*computation* of an n-DS d is a computation $\widehat{d}$ such that $\widehat{d} \in$ pspace. An n-DS is pspace-*computable* if there exists a pspace-computation $\widehat{d}$ of d. (Note that (3.1) implies that

$$d_{\vec{k}}(x) = \lim_{r \to \infty} \widehat{d}_{\vec{k},r}(x)$$

for all $\vec{k} \in \mathbf{N}^n$ and $x \in \{0, 1\}^*$.)

The *set covered by* a density function d is

$$S[d] = \bigcup_{x \in \{0,1\}^* \wedge d(x) \geq 1} C_x.$$

A density function d *covers* a set X of languages if $X \subseteq S[d]$. A *null cover* of a set X of languages is a 1-DS d such that, for all $k \in \mathbf{N}$, d_k covers X with global value $d_k(\lambda) \leq 2^{-k}$. It is easy to show [18] that a set X of languages has classical Lebesgue measure 0 (i.e., probability 0 in the coin-tossing random experiment) if and only if there exists a null cover of X. In this paper we are interested in the situation in which the null cover d is pspace-computable.

DEFINITIONS. Let X be a set of languages and let X^c denote the complement of X.

(1) A pspace-*null cover* of X is a null cover of X that is pspace-computable.

(2) X has pspace-*measure* 0, and we write $\mu_{\text{pspace}}(X) = 0$, if there exists a pspace-null cover of X.

(3) X has pspace measure 1, and we write $\mu_{\text{pspace}}(X) = 1$, if $\mu_{\text{pspace}}(X^c) = 0$.

(4) X has *measure* 0 in ESPACE = DSPACE(2^{linear}), and we write $\mu(X \mid \text{ESPACE}) = 0$, if $\mu_{\text{pspace}}(X \cap \text{ESPACE}) = 0$.

(5) X has *measure* 1 *in* ESPACE, and we write $\mu(X \mid \text{ESPACE}) = 1$, if $\mu(X^c \mid \text{ESPACE}) = 0$. In this case, we say that X contains *almost every* language in ESPACE.

It is shown in [18] that these definitions endow ESPACE with internal measure-theoretic structure. Specifically, if $\mathcal{I}$ is either the collection $\mathcal{I}_{\text{pspace}}$ of all pspace-measure-0 sets or the collection $\mathcal{I}_{\text{ESPACE}}$ of all sets of measure 0 in ESPACE, then $\mathcal{I}$ is a pspace-ideal, i.e., it is closed under subsets, finite unions, and pspace-unions (countable unions that can be generated in polynomial space). More importantly, it is shown that the ideal $\mathcal{I}_{\text{ESPACE}}$ is a *proper* ideal, i.e., that ESPACE does *not* have measure 0 in ESPACE.

Our proof of Theorem 3 does not proceed directly from the above definitions. Instead we use a sufficient condition, proved in [18], for a set to have pspace-measure 0. To state this condition we need a polynomial notion of convergence for infinite series. All series here consist of nonnegative terms. A *modulus* for a series $\sum_{n=0}^{\infty} a_n$ is a function $m : \mathbf{N} \to \mathbf{N}$ such that

$$\sum_{n=m(j)}^{\infty} a_n \leq 2^{-j}$$

for all $j \in \mathbf{N}$. A series is p-*convergent* if it has a modulus that is a polynomial. A sequence

$$\sum_{k=0}^{\infty} a_{j,k} \qquad (j = 0, 1, 2, \ldots)$$

of series is *uniformly* p-*convergent* if there exists a polynomial $m : \mathbf{N}^2 \to \mathbf{N}$ such that, for each $j \in \mathbf{N}$, m_j is a modulus for the series $\sum_{k=0}^{\infty} a_{j,k}$. We will use the following sufficient condition for uniform p-convergence. (This well-known lemma is easily verified by routine calculus.)

LEMMA 6. *Let* $a_{j,k} \in [0, \infty)$ *for all* $j, k \in \mathbf{N}$. *If there exist a real* $\varepsilon > 0$ *and a polynomial* $g : \mathbf{N} \to \mathbf{N}$ *such that* $a_{j,k} \leq e^{-k^{\varepsilon}}$ *for all* $j, k \in \mathbf{N}$ *with* $k \geq g(j)$, *then the series*

$$\sum_{k=0}^{\infty} a_{j,k} \qquad (j = 0, 1, 2, \ldots)$$

are uniformly p-*convergent.*

The proof of Theorem 3 is greatly simplified by using the following special case (for pspace) of a uniform, resource-bounded generalization of the classical first Borel–Cantelli lemma.

LEMMA 7 (Borel [3], Cantelli [4], Lutz [18]). *If d is a* pspace-*computable* 2-DS *such that the series*

$$\sum_{k=0}^{\infty} d_{j,k}(\lambda) \qquad (j = 0, 1, 2, \ldots)$$

are uniformly p*-convergent, then*

$$\mu_{\text{pspace}}\left(\bigcup_{j=0}^{\infty}\bigcap_{t=0}^{\infty}\bigcup_{k=t}^{\infty} S[d_{j,k}]\right) = 0.$$

If we write $S_j = \bigcap_{t=0}^{\infty}\bigcup_{k=t}^{\infty} S[d_{j,k}]$ and $S = \bigcup_{j=0}^{\infty} S_j$, then Lemma 7 gives a sufficient condition for concluding that S has pspace-measure 0. Note that each S_j consists of those languages A that are in infinitely many of the sets $S[d_{j,k}]$.

Finally, we review the notion of pspace-randomness. A pspace-*test* is a set X of languages such that $\mu_{\text{pspace}}(X) = 1$. A language A *passes* a pspace-test X if $A \in X$. A language A is pspace-*random*, and we write $A \in \text{RAND(pspace)}$, if A passes all pspace-tests. That is,

$$\text{RAND(pspace)} = \bigcap_{\mu_{\text{pspace}}(X)=1} X.$$

Since every finite subset of ESPACE has pspace-measure 0 [18], it is immediate that

$$\text{RAND(pspace)} \cap \text{ESPACE} = \emptyset. \tag{3.2}$$

Moreover, every pspace-random language has essentially maximum circuit-size complexity and space-bounded Kolmogorov complexity [18]. Intuitively, pspace-random languages are random enough for all pspace-computable purposes. On the other hand, pspace-random languages may be computable. In fact, notwithstanding (3.2), almost every language in $\text{E}_2\text{SPACE} = \text{DSPACE}(2^{\text{polynomial}})$ is pspace-random [18].

4. Hardness under pseudorandom oracles. In this section we prove Theorem 3. For each $A \subseteq \{0, 1\}^*$ let

$$\text{ODD}(A) = \left\{u \in \{0,1\}^* \;\middle|\; |C(u, A)| \text{ is odd}\right\},$$

where

$$C(u, A) = \left\{uv \in A \;\middle|\; |v| = 2|u|\right\},$$

and let

$$X = \{A \mid \text{ODD}(A) \notin \text{H}^A\}.$$

Then $\text{ODD}(A) \in \text{E}^A$ for all A, so that it suffices to prove that $\mu_{\text{pspace}}(X) = 0$.

For each $j, k \in \mathbf{N}$ let

$$X_{j,k} = \begin{cases} \{A \mid H^A_{\text{ODD}(A)}(n) \leq 2^{\alpha(l)n}\} & \text{if } j = 2^{l^2} \text{ and } k = 2^n, \\ \emptyset & \text{if } j \text{ and } k \text{ are not of this form,} \end{cases}$$

where $\alpha(l) = \frac{l+1}{3l+4}$. (Note that $\alpha(0) = \frac{1}{4}$, $\alpha(l)$ is strictly increasing, and $\lim_{l\to\infty}\alpha(l) = \frac{1}{3}$.) It is clear that

$$X = \bigcup_{j=0}^{\infty}\bigcap_{t=0}^{\infty}\bigcup_{k=t}^{\infty} X_{j,k}. \tag{4.1}$$

We will use (4.1) and Lemma 7 to prove that $\mu_{\text{pspace}}(X) = 0$.

For all $l, n \in \mathbf{N}$ let $j = 2^{l^2}$, $k = 2^n$, and define the sets

$$\mathrm{OCIRC}(l,n) = \{\gamma \mid \gamma \text{ is a novel } n\text{-input oracle circuit with size}(\gamma) \leq k^{\alpha(l)}\},$$
$$\mathrm{DELTA}(l,n) = \left\{ D \subseteq \{0,1\}^n \;\middle|\; |D| \leq \tfrac{k}{2}(1 - k^{-\alpha(l)}) \right\}.$$

(An n-input oracle circuit γ is *novel* if it is functionally distinct from all those preceding it in a standard enumeration.) Then for all $\gamma \in \mathrm{OCIRC}(l,n)$ and $D \in \mathrm{DELTA}(l,n)$ let

$$Y_{\gamma,D} = \{A \mid L^A(\gamma) \mathbin{\triangle} D = \mathrm{ODD}(A)_{=n}\}.$$

Note that

$$X_{j,k} = \bigcup_{\gamma \in \mathrm{OCIRC}(l,n)} \bigcup_{D \in \mathrm{DELTA}(l,n)} Y_{\gamma,D} \tag{4.2}$$

for all $l, n \in \mathbf{N}$, where $j = 2^{l^2}$ and $k = 2^n$.

Define $d : \mathbf{N}^2 \times \{0,1\}^* \to [0,\infty)$ by

(4.3)
$$d_{j,k}(x) = \begin{cases} \displaystyle\sum_{\gamma \in \mathrm{OCIRC}(l,n)} \sum_{D \in \mathrm{DELTA}(l,n)} P(Y_{\gamma,D} \mid C_x) & \text{if } j = 2^{l^2} \text{ and } k = 2^n, \\ 0 & \text{if } j \text{ and } k \text{ are not of this form.} \end{cases}$$

The conditional probability

$$P(Y_{\gamma,D} \mid C_x) = \Pr_A[A \in Y_{\gamma,D} \mid A \in C_x]$$

in (4.3) is computed according to the uniform distribution on languages $A \subseteq \{0,1\}^*$, i.e., the random experiment in which A is chosen probabilistically by using an independent toss of a fair coin to decide whether each string $y \in \{0,1\}^*$ is in A. Note that $P(X_{j,k} \mid C_x) \leq d_{j,k}(x)$ for all $j, k \in \mathbf{N}$ and $x \in \{0,1\}^*$. (This inequality may be strict because the union (4.2) is not a disjoint union.)

By (4.1) and Lemma 7 it suffices to prove the following three claims.

CLAIM 1. *d is a* pspace-*computable* 2-DS.

CLAIM 2. *For all $j, k \in \mathbf{N}$, $X_{j,k} \subseteq S[d_{j,k}]$.*

CLAIM 3. *The series*

$$\sum_{k=0}^{\infty} d_{j,k}(\lambda) \qquad (j = 0, 1, 2, \ldots)$$

are uniformly p-*convergent.*

Proof of Claim 1. First note that each

$$\begin{aligned} P(Y_{\gamma,D} \mid C_x) &= \frac{P(Y_{\gamma,D} \cap C_x)}{P(C_x)} \\ &= \frac{P(Y_{\gamma,D} \cap C_{x0}) + P(Y_{\gamma,D} \cap C_{x1})}{P(C_x)} \\ &= \frac{P(Y_{\gamma,D} \cap C_{x0})}{2P(C_{x0})} + \frac{P(Y_{\gamma,D} \cap C_{x1})}{2P(C_{x1})} \\ &= \frac{P(Y_{\gamma,D} \mid C_{x0}) + P(Y_{\gamma,D} \mid C_{x1})}{2}, \end{aligned}$$

so that

$$d_{j,k}(x) = \frac{d_{j,k}(x0) + d_{j,k}(x1)}{2}$$

for all $j, k \in \mathbf{N}$ and $x \in \{0,1\}^*$. It follows that d is a 2-DS.

It is clear that we can use (4.3) to compute d, provided that we can compute the conditional probabilities $P(Y_{\gamma,D} \mid C_x)$. We thus focus on this computation.

Fix $\gamma \in \mathrm{OCIRC}(l,n)$ and $D \in \mathrm{DELTA}(l,n)$. Let $\mathrm{SOURCES}(n) = \{0,1\}^{k^3+k^2}$. For each $z \in \mathrm{SOURCES}(n)$ let a string $w \in \Sigma^*$ of length $2^{k^{\alpha(l)}+1} - 1$ and a set ODD $\subseteq \{0,1\}^n$ be constructed as follows. (For each $A \in Y_{\gamma,D}$ this process will, for some z, produce a string $w \sqsubseteq \chi_A$ and corresponding set $\mathrm{ODD} = \mathrm{ODD}(A) \cap \{0,1\}^n$.) Initially, $\mathrm{ODD} = \emptyset$ and w is all $\perp$'s. Then simulate γ on the successive inputs $u \in \{0,1\}^n$. Each time γ queries a string y in this simulation, do **if** $w[y] = \perp$ **then** $(w[y], z) := (\mathrm{head}(z), \mathrm{tail}(z))$. (Note that $|w|$ has been chosen large enough for $w[y]$ to exist here.) Then, in any case, use $w[y]$ as the response to the query. If $\gamma(u) = 1$ in this simulation, do $\mathrm{ODD} := \mathrm{ODD} \cup \{u\}$. After γ has been simulated on all inputs, do $\mathrm{ODD} := \mathrm{ODD} \mathbin{\triangle} D$. At this point note that at most $k^{1+\alpha(l)} < k^{4/3}$ of the bits $w[y]$ of w are in $\{0,1\}$; the rest are still $\perp$. Finally, use the remaining bits of z (actually use a portion of them, as needed) to complete the specification of w as follows. For each $u \in \{0,1\}^n$ first use bits of z to fill in all but one of the values $w[uv]$ for $v \in \{0,1\}^{2n}$; then define the remaining bit $w[uv]$ according to whether $u \in \mathrm{ODD}$. (The measure argument in Claim 3 below works precisely because these k bits—one for each u—are determined by ODD.) Finally, let z' be the initial segment of the original string $z \in \mathrm{SOURCES}(n)$ consisting of those bits actually used in this construction. Note that $|z'| < k^3 + k^{4/3}$ and that all but $|z'| + k$ bits of w are still $\perp$. Since w depends only on the prefix z' of z, we write $w = w(z')$.

Let $\mathrm{SOURCES}'(n) = \{z' \mid z \in \mathrm{SOURCES}(n)\}$. Since γ is a fixed oracle circuit (whose gates we simulate in a fixed topological order), we have $C_{w(z_1')} \cap C_{w(z_2')} = \emptyset$ for distinct $z_1', z_2' \in \mathrm{SOURCES}'(n)$. Moreover, it is clear that

$$Y_{\gamma,D} = \bigcup_{z' \in \mathrm{SOURCES}'(n)} C_{w(z')}. \tag{4.4}$$

It follows that for all $x \in \{0,1\}^*$

$$\begin{aligned} P(Y_{\gamma,D} \mid C_x) &= \sum_{z' \in \mathrm{SOURCES}'(n)} P(C_{w(z')} \mid C_x) \\ &= \sum_{z' \in \mathrm{SOURCES}'(n)} 2^{|x| - \|x \wedge w(z')\|}. \end{aligned} \tag{4.5}$$

This is the basis for our computation. Given j, k, x, γ, D, and z', it is clear that we can compute $2^{|x| - \|x \wedge w(z')\|}$ in space polynomial in $j + k + |x|$. (The string $w(z')$ has fewer than $k^3 + k^2 + k$ non-$\perp$ bits, so that it can be stored in space polynomial in k.) We can find the successive strings $z' \in \mathrm{SOURCES}'(n)$ by a depth-first search of $\{0,1\}^{\leq k^3+k^2}$, also in polynomial space. We can thus use (4.5) to calculate $P(Y_{\gamma,D} \mid C_x)$ in space polynomial in $j + k + |x|$. As already noted, we can then use (4.3) to calculate $d_{j,k}(x)$ in polynomial space. This proves that $d \in \mathrm{pspace}$, whence d is certainly pspace-computable, affirming Claim 1. □

Proof of Claim 2. Fix $j, k \in \mathbf{N}$. If j and k are not of the form $j = 2^{l^2}$ and $k = 2^n$, then $X_{j,k} \subseteq S[d_{j,k}]$ holds trivially. If $j = 2^{l^2}$ and $k = 2^n$, let $A \in X_{j,k}$. By (4.2) fix $\gamma \in \mathrm{OCIRC}(l,n)$ and $D \in \mathrm{DELTA}(l,n)$ such that $A \in Y_{\gamma,D}$. By (4.4) fix $z' \in \mathrm{SOURCES}'(n)$ such that $A \in C_{w(z')}$. Let $m = |w(z')|$. Then

$$\begin{aligned} d_{j,k}(\chi_A[0..m-1]) &\geq P(Y_{\gamma,D} \mid C_{\chi_A[0..m-1]}) \\ &= P(C_{w(z')} \mid C_{\chi_A[0..m-1]}) \\ &= 1, \end{aligned}$$

so that $A \in S[d_{j,k}]$ in any case. This proves Claim 2. □

Proof of Claim 3. We estimate the global values $d_{j,k}(\lambda)$. Fix $l, n \in \mathbf{N}$, and let $j = 2^{l^2}$, $k = 2^n$. Fix $\gamma \in \text{OCIRC}(l, n)$ and $D \in \text{DELTA}(l, n)$. By (4.5) and the fact that k bits of each $w(z')$ are determined by ODD, we have

$$\begin{aligned} P(Y_{\gamma,D} \mid C_\lambda) &= \sum_{z' \in \text{SOURCES}'(n)} 2^{-\|w(z')\|} \\ &= 2^{-k} \sum_{z' \in \text{SOURCES}'(n)} 2^{-|z'|} \\ &= 2^{-k}. \end{aligned}$$

(The last equality here holds because every string $z \in \text{SOURCES}(n)$ has exactly one prefix $z' \in \text{SOURCES}'(n)$.) Since γ and D are arbitrary here, it follows by (4.3) that

$$d_{j,k}(\lambda) \leq |\text{OCIRC}(l, n)| \ |\text{DELTA}(l, n)| \cdot 2^{-k}. \tag{4.6}$$

A routine counting argument shows that

$$|\text{OCIRC}(l, n)| \leq a(4ek^{\alpha(l)})^{k^{\alpha(l)}},$$

where $a = 2685$. (This is Lemma 4.2 of [19].) It follows that there is a constant $n_1 \in \mathbf{N}$ such that

$$|\text{OCIRC}(l, n)| \leq 2^{k^{\alpha(l)} \log k} \tag{4.7}$$

for all $l, n \in \mathbf{N}$ with $n \geq n_1$. (The constant n_1 does not depend on l here because $\alpha(l) < \frac{1}{3}$ for all l.) By the Chernoff bound (see [7], [8], [10]),

$$|\text{DELTA}(l, n)| \leq 2^k \rho^k, \tag{4.8}$$

where

$$\rho = \left[(1 - \varepsilon^2)^{-1} \left(\frac{1 - \varepsilon}{1 + \varepsilon}\right)^{\varepsilon}\right]^{1/2}, \qquad \varepsilon = k^{-\alpha(l)}. \tag{4.9}$$

Calculating with Taylor approximations, we have

$$\begin{aligned} \left(\frac{1 - \varepsilon}{1 + \varepsilon}\right)^{\varepsilon} &= (1 - 2\varepsilon + o(\varepsilon))^{\varepsilon} = e^{\varepsilon \ln(1 - 2\varepsilon + o(\varepsilon))} \\ &= e^{-2\varepsilon^2 + o(\varepsilon^2)} = 1 - 2\varepsilon^2 + o(\varepsilon^2) \end{aligned}$$

as $\varepsilon \to 0$. Since $(1 - \varepsilon^2)(1 - \frac{1}{2}\varepsilon^2) = 1 - \frac{3}{2}\varepsilon^2 + o(\varepsilon^2)$ as $\varepsilon \to 0$, it follows that

$$\left(\frac{1 - \varepsilon}{1 + \varepsilon}\right)^{\varepsilon} < (1 - \varepsilon^2)\left(1 - \frac{1}{2}\varepsilon^2\right) \tag{4.10}$$

for all sufficiently small ε. By (4.8), (4.9), and (4.10) there is a constant $n_2 \in \mathbf{N}$ such that

$$\begin{aligned}|\mathrm{DELTA}(l,n)| &\le 2^k(1-\tfrac{1}{2}\varepsilon^2)^{k/2}\\ &= 2^{k+(k/2)\log(1-(1/2)\varepsilon^2)}\\ &\le 2^{k-ck\varepsilon^2}\\ &\le 2^{k-ck^{1-2\alpha(l)}}\end{aligned} \tag{4.11}$$

for all $l, n \in \mathbf{N}$ with $n \ge n_2$, where $c = \frac{1}{4\ln 2}$. (The constant n_2 does not depend on l because $\varepsilon = k^{-\alpha(l)} \le k^{-\alpha(0)} = k^{-1/4}$ in any case.)

Let $k_0 = 2^{\max\{n_1,n_2\}}$. By (4.3), (4.6), (4.7), and (4.11) we have

$$d_{j,k}(\lambda) \le 2^{k^{\alpha(l)}\log k - ck^{1-2\alpha(l)}} \tag{4.12}$$

for all $j, k \in \mathbf{N}$ with $j = 2^{l^2}$ and $k \ge k_0$. Define a polynomial $g : \mathbf{N} \to \mathbf{N}$ by

$$g(j) = 2^{185}j^{381} + k_0$$

for all $j \in \mathbf{N}$. Writing $t = \ln k$ and $a = 3l + 4$, we have

$$\begin{aligned}k \ge g(j) \quad \Longrightarrow \quad t &\ge 185\ln 2 + 381l^2\ln 2\\ &\ge 128 + 264l^2\\ &\ge 8a^2.\end{aligned} \tag{4.13}$$

Examining the function $f(t) = e^{t/a} - 4t - 4$ and its derivative shows that $f(t) > 0$ for all $t \ge 8a^2$. By (4.13) then

$$\begin{aligned}k \ge g(j) &\Longrightarrow e^{t/a} - 4t - 4 \ge 0\\ &\Longleftrightarrow k^{1/a} - 4\ln k - 4 \ge 0\\ &\Longleftrightarrow ck^{1-3\alpha(l)} - \log k - \log e \ge 0\\ &\Longleftrightarrow 2^{k^{\alpha(l)}\log k - ck^{1-2\alpha(l)}} \le e^{-k^{\alpha(l)}}.\end{aligned} \tag{4.14}$$

By (4.12) and (4.14) we have

$$d_{j,k}(\lambda) \le e^{-k^{\alpha(l)}} \le e^{-k^{1/4}}$$

for all $j, k \in \mathbf{N}$ with $k \ge g(j)$. It follows by Lemma 6 that the series

$$\sum_{k=0}^{\infty} d_{j,k}(\lambda) \qquad (j = 0, 1, 2, \ldots)$$

are uniformly p-convergent, i.e., Claim 3 holds. □

By (4.1) and Claim 2 we have

$$X \subseteq \bigcup_{j=0}^{\infty}\bigcap_{t=0}^{\infty}\bigcup_{k=t}^{\infty} X_{j,k} \subseteq \bigcup_{j=0}^{\infty}\bigcap_{t=0}^{\infty}\bigcup_{k=t}^{\infty} S[d_{j,k}].$$

By Claims 1 and 3 and Lemma 7 it follows that $\mu_{\text{pspace}}(X) = 0$. This completes the proof of Theorem 3 (and the Main Theorem). □

5. Conclusion. We have used pseudorandom oracles to give a new characterization of BPP. If we write RAND(pspace) for the set of all pspace-random languages, then our characterization implies that $L \in \mathrm{P}^A$ for *every* $L \in$ BPP and *every* $A \in$ RAND(pspace). This result strengthens the intuition that pspace-random languages are adequate sources for all BPP problems. (Earlier, more asymptotic evidence for this view appears in [17].)

Our work also gives a more detailed analysis of the Bennett and Gill [2] result that $\mathrm{P}^A = \mathrm{BPP}^A$ for almost every oracle A. Specifically, under *every* pspace-random oracle A, E^A contains languages that are very hard to approximate with oracle circuits. Such a hard language can, by the work of Nisan and Wigderson [21], [22], be used to construct a pseudorandom generator that is quick enough and secure enough to establish $\mathrm{P}^A = \mathrm{BPP}^A$. Since almost every oracle A is pspace-random, the result of Bennett and Gill [2] follows.

Acknowledgments. I thank Ron Book and other participants in an informal workshop in structural complexity theory, held in 1990 at the University of California in Santa Barbara, for their helpful remarks on this work. I also thank David Juedes and Elvira Mayordomo for useful observations on resource-bounded measure.

REFERENCES

[1] K. AMBOS-SPIES, *Randomness, relativizations, and polynomial reducibilities*, in Proc. 1st Structure in Complexity Theory Conference, Berkeley, California, Lecture Notes in Computer Science, 223, 1986, Springer-Verlag, Berlin, pp. 23–34.

[2] C. H. BENNETT AND J. GILL, *Relative to a random oracle A*, $\mathrm{P}^A \neq \mathrm{NP}^A \neq co\text{-}\mathrm{NP}^A$ *with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[3] E. BOREL, *Sur les probabilités dénombrables et leurs applications arithmétiques*, Rend. Circ. Mat. Palermo, 26 (1909), pp. 247–271.

[4] F. P. CANTELLI, *La tendenza ad un limite nel senzo del calcolo della probabilità*, Rend. Circ. Mat. Palermo, 16 (1916), pp. 191–201.

[5] G. J. CHAITIN, *A theory of program size formally identical to information theory*, J. Assoc. Comput. Mach., 22 (1975), pp. 329–340.

[6] ———, *Incompleteness theorems for random reals*, Adv. in Appl. Math., 8 (1987), pp. 119–146.

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

[8] P. ERDÖS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[9] J. GILL, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput., 6 (1977), pp. 675–695.

[10] T. HAGERUP AND C. RÜB, *A guided tour of Chernoff bounds*, Inform. Process. Lett., 33 (1990), pp. 305–308.

[11] P. HINMAN AND S. ZACHOS, *Probabilistic machines, oracles and quantifiers*, in Proc. Oberwolfach Recursion-Theoretic Week, Lecture Notes in Mathematics, 1141, 1984, Springer-Verlag, Berlin, pp. 159–192.

[12] J. KÄMPER, *Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes*, in Proc. 8th Conference on Foundations of Software Technology and Theoretical Computer Science, Pune, India, Lecture Notes in Computer Science, 338, 1988, Springer-Verlag, Berlin, pp. 193–210.

[13] S. KURTZ, *A note on randomized polynomial time*, SIAM J. Comput., 16 (1987), pp. 852–853.

[14] L. A. LEVIN, *On the notion of a random sequence*, Soviet Math. Dokl., 14 (1973), pp. 1413–1416.

[15] J. H. LUTZ, *Almost everywhere high nonuniform complexity*, in Proc. 4th IEEE Structure in Complexity Theory Conference, IEEE Computer Society, Eugene, Oregon, 1989, pp. 37–53.

[16] ———, *Category and measure in complexity classes*, SIAM J. Comput., 19 (1990), pp. 1100–1131.

[17] ———, *Pseudorandom sources for* BPP, J. Comput. System Sci., 41 (1990), pp. 307–320.

[18] ———, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[19] J. H. LUTZ AND W. J. SCHMIDT, *Circuit size relative to pseudorandom oracles*, Theoret. Comput. Sci., 107 (1993), pp. 95–120.

[20] P. MARTIN-LÖF, *On the definition of random sequences*, Inform. and Control, 9 (1966), pp. 602–619.

[21] N. NISAN, *Using Hard Problems to Create Pseudorandom Generators*, MIT Press, Cambridge, MA, 1992.

[22] N. NISAN AND A. WIGDERSON, *Hardness vs. randomness*, in Proc. 29th Anual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1988, pp. 2–11.

[23] C. P. SCHNORR, *Process complexity and effective random tests*, J. Comput. System Sci., 7 (1973), pp. 376–388.

[24] A. K. SHEN′, *On relations between different algorithmic definitions of randomness*, Sov. Math. Dokl., 38 (1989), pp. 316–319.

[25] M. SIPSER, *A complexity-theoretic approach to randomness*, in Proc. 15th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1983, pp. 330–335.

[26] R. M. SOLOVAY, 1975, reported in [6].

[27] S. TANG AND R. BOOK, *Polynomial-time reducibilities and "almost-all" oracle sets*, Theoret. Comput. Sci., 81 (1991), pp. 35–47.

[28] S. TANG AND O. WATANABE, *On tally relativizations of* BP-*complexity classes*, SIAM J. Comput., 18 (1989), pp. 449–462.

[29] C. B. WILSON, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169–181.

[30] S. ZACHOS, *Probabilistic quantifiers and games*, J. Comput. System Sci., 36 (1988), pp. 433–451.

[31] S. ZACHOS AND M. FURER, *Probabilistic quantifiers vs. distrustful adversaries*, in Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, 1987, pp. 443–455.

[32] S. ZACHOS AND H. HELLER, *A decisive characterization of* BPP, Inform. and Control, 69 (1986), pp. 125–135.

SIAM J. COMPUT.
Vol. 22, No. 5, pp. 1087–1116, October 1993

POLYNOMIAL-TIME APPROXIMATION ALGORITHMS FOR THE ISING MODEL*

MARK JERRUM† AND ALISTAIR SINCLAIR†

Abstract. The paper presents a randomised algorithm which evaluates the partition function of an arbitrary ferromagnetic Ising system to any specified degree of accuracy. The running time of the algorithm increases only polynomially with the size of the system (i.e., the number of sites) and a parameter which controls the accuracy of the result. Further approximation algorithms are presented for the mean energy and the mean magnetic moment of ferromagnetic Ising systems.

The algorithms are based on Monte Carlo simulation of a suitably defined ergodic Markov chain. The states of the chain are not, as is customary, Ising spin configurations, but spanning subgraphs of the interaction graph of the system. It is shown that the expectations of simple operators on these configurations give numerical information about the partition function and related quantities.

The performance guarantees for the algorithms are rigorously derived and rest on the fact that the Markov chain in question is rapidly mixing, i.e., converges to its equilibrium distribution in a polynomial number of steps. This is apparently the first time that rapid mixing has been demonstrated at all temperatures for a Markov chain related to the Ising model.

Key words. the Ising model, statistical physics, ferromagnetism, spin-glasses, partition function, #P-completeness, approximation algorithms, Markov chains, rapid mixing, Monte Carlo simulation

AMS subject classifications. 05C85, 60J10, 60J20, 60K35, 68Q20, 68Q25, 82B20, 82B31, 82B80

1. Summary.

This paper is concerned with computational solutions to a classical combinatorial problem of statistical physics. Generally known as the *Ising model*, the problem has been the focus of much attention in the physics and mathematics communities since it was first introduced by Lenz [24] and Ising [14] in the early 1920s. We will not present a detailed historical account here; a very readable survey is given by Cipra [6], while Welsh [30] sets the Ising model in the context of other combinatorial problems in statistical physics.

The problem is easily stated. Consider a collection of sites $[n] = \{0, 1, \ldots, n-1\}$, each pair i, j of which has an associated *interaction energy* V_{ij}. In most cases of physical interest, the set E of pairs with nonzero interaction energies forms a regular lattice graph $([n], E)$. A *configuration* is an assignment of positive ($\sigma_i = +1$) and negative ($\sigma_i = -1$) spins to each site $i \in [n]$. The *energy* of a configuration $\sigma = (\sigma_i)$ is given by the Hamiltonian

$$H(\sigma) = -\sum_{\{i,j\}\in E} V_{ij}\sigma_i\sigma_j - B\sum_{k\in[n]}\sigma_k,$$

where B is an external field.

In the case where all interaction energies are nonnegative, such a system models the behaviour of a *ferromagnet*; in fact, it was towards an understanding of spontaneous magnetization that the model was first conceived. However, the Ising model has since become a powerful paradigm for the investigation of more general cooperative systems in which short-range interactions between elements can give rise to long-range order.

The central problem is to compute the *partition function*

$$Z = Z(V_{ij}, B, \beta) = \sum_{\sigma}\exp(-\beta H(\sigma)),$$

*Received by the editors February 10, 1992; accepted for publication (in revised form) March 30, 1992. An extended abstract of this paper appeared in the "Proceedings of the International Colloquium on Automata, Languages and Programming," Warwick, United Kingdom, July 1990; published by Springer-Verlag as Lecture Notes in Computer Science, Vol. 443.

†Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, Scotland.

where $\beta > 0$ is related to the temperature and the sum is over all possible configurations σ. Almost all the physical properties of the system can be computed from knowledge of Z. Essentially, Z is the normalising factor in the calculation of probabilities: according to the fundamental theory of statistical mechanics, the probability that the system in equilibrium is found in configuration σ is $\exp(-\beta H(\sigma))/Z$. Moreover, certain logarithmic derivatives of Z correspond to quantities such as the mean energy and the mean magnetic moment. Singularities in these derivatives generally correspond to *phase transitions*, when a small change in a parameter has an observable effect on the macroscopic properties of the system.

The search for efficient computational solutions to these problems has proved extremely hard and has generated a vast body of literature. A major breakthrough was achieved in the early 1960s by Kasteleyn [19] and Fisher [11], who reduced the problem of computing Z for any *planar* Ising system (i.e., one whose graph $([n], E)$ of nonzero interactions is planar) to the evaluation of a certain determinant. This must rank as one of the highlights in the field of combinatorial algorithms. It remains the state of the art as far as exact solutions are concerned; in particular, it does not appear to generalise to nonplanar systems. On the other hand, a huge amount of computational effort is poured into *numerical* solutions of the Ising model for three-dimensional regular lattices and other nonplanar systems. The problem is that the methods used here, while ingenious, generally lack a rigorous theoretical base and rely for their validity largely on physical intuition.

In this paper, we exhibit what we believe to be the first provably efficient approximation algorithm for the partition function of an arbitrary ferromagnetic Ising system. By "efficient" here we mean that the algorithm is guaranteed to run in time polynomial in the number of sites n. The algorithm is a *fully polynomial randomised approximation scheme* (fpras), i.e., it will produce solutions which, with very high probability, fall within arbitrarily small error bounds specified by the user, the price of greater accuracy being a modest increase in runtime. We also show that such an algorithm is essentially the best one can hope for, in the sense that the existence of an efficient *exact* algorithm for the problem, or even of an efficient approximation algorithm for the nonferromagnetic case, would have devastating and far-reaching consequences in the theory of computation.

From the point of view of theoretical computer science, our result provides a new example of a significant combinatorial enumeration problem which is #P-complete, and hence apparently intractable in exact form, but for which an efficient approximation algorithm exists. This is an intriguing class of problems, and includes the problems of computing the volume of a convex body [9], the partition function of a monomer-dimer system [16], and the permanent of a large class of 0–1 matrices [16]. Our algorithm is also of interest in its own right as a further application of the general technique of simulating an ergodic stochastic process whose rate of convergence can be analysed. This approach has recently attracted much attention, and its full algorithmic potential is only now becoming apparent.

The idea is the following. In order to compute weighted combinatorial sums, such as the Ising partition function, it is often enough to be able to sample configurations σ at random with probabilities proportional to their weights, in this case $\exp(-\beta H(\sigma))$. This can be achieved by setting up an ergodic Markov chain whose states are configurations and whose transitions correspond to small local perturbations. If the chain is designed so that the equilibrium distribution to which it converges is the desired weighted distribution over configurations, then we get a random sampling procedure by simulating the chain for some number of steps and outputting the final state. For such a procedure to be efficient, the chain must be *rapidly mixing* in the sense that it gets very close to equilibrium after a small (i.e., polynomial) number of steps. This is a highly nontrivial requirement, since the number of states is exponentially large. Recent developments have provided appropriate analytical tools for establishing the rapid mixing property for chains of this kind [27], [29], [7], [28].

The Markov chain simulation approach to the Ising model is far from new: Under the name of the Monte Carlo method, this technique has been applied extensively to a whole range of problems in statistical physics (see, e.g., [4]). The problem with the approach, however, is that it appears very difficult to define a Markov chain on Ising spin configurations σ which is rapidly mixing; indeed, the chains which are frequently used in practical simulation studies clearly do not have this property.

We overcome this obstacle by transforming the problem to an entirely new domain, where the configurations are spanning subgraphs of the interaction graph $([n], E)$. Each subgraph has an "energy" which is determined by weights attached to its edges and vertices. Although there is no direct correspondence between configurations in the two domains, and the subgraph configurations have no obvious physical significance, the two partition functions are, remarkably, very closely related. Moreover, and crucially, there is a natural Markov chain on the subgraphs with the appropriate equilibrium distribution which *is* rapidly mixing. Thus the Markov chain approach can be made to work efficiently in the new domain.

The above transformation is a classical result [26], often known as the "high-temperature expansion" of the Ising model partition function. However, the idea of viewing the graphs in this expansion as a statistical mechanical system which forms the basis of a Monte Carlo simulation appears to be new. To the best of our knowledge, our results represent the first rigorous proof of rapid mixing at all temperatures for a Markov chain related to the Ising model. Moreover, this property is entirely independent of the interaction topology and relies on no assumptions of any kind. We therefore believe that the chain deserves further investigation as a potentially powerful experimental tool.

The mechanism by which we use sampling of subgraph configurations to compute the partition function is perhaps of independent interest. This is achieved by subjecting an Ising system with fixed interactions and at a fixed temperature to varying external fields. By observing a small number of configurations, randomly selected at appropriately chosen values of the field, we are able to get an accurate estimate of Z. It is significant that this idea is motivated by combinatorial considerations and does not correspond to any obvious physical intuition.

As mentioned earlier, it is often derivatives of the partition function, rather than the function itself, which are of primary interest. For example, two important quantities are the *mean energy* $\mathcal{E} = -\partial(\ln Z)/\partial\beta$, and the *mean magnetic moment* $\mathcal{M} = \beta^{-1}\partial(\ln Z)/\partial B$. Our approximation algorithm for Z says nothing about our ability to compute these quantities accurately. However, it turns out that both $\mathcal{E}$ and $\mathcal{M}$ can be expressed in terms of expectations of certain simple operators on configurations in the subgraphs domain. Thus estimates of $\mathcal{E}$ and $\mathcal{M}$ can be read off from our configuration sampling algorithm, though again we may have to vary the external field in order to maximise the accuracy of the statistical experiment. As a result, we get a fpras for both $\mathcal{E}$ and $\mathcal{M}$ as well. We regard this as confirmation that our approach to the Ising model is robust and computationally effective.

The remainder of the paper is organised as follows. In §2 we describe the transformation of the Ising model to the new domain in which configurations are spanning subgraphs of the interaction graph. Section 4 is devoted to a discussion of the Markov chain on these configurations, and in particular to a proof that it is rapidly mixing. This fact is used in §3 to construct a fpras for the partition function of an arbitrary ferromagnetic Ising system, and in §5 to construct efficient approximation algorithms for the mean energy and the mean magnetic moment. Finally, in §6 we present strong evidence that our results are, in a precise theoretical sense, best possible.

2. The spins world and subgraphs world. Recall that our primary aim is to construct an algorithm for the following problem:

INSTANCE: A real symmetric matrix $(V_{ij} : i, j \in [n])$ of interaction energies, a real number B (the external field), and a positive real number β.

OUTPUT: The Ising partition function

$$Z = Z(V_{ij}, B, \beta) = \sum_{\sigma \in \{-1,+1)^n} \exp(-\beta H(\sigma)), \tag{1}$$

where the Hamiltonian $H(\sigma)$ is given by

$$H(\sigma) = -\sum_{\{i,j\}\in E} V_{ij}\sigma_i\sigma_j - B\sum_{k\in[n]} \sigma_k,$$

and E is the set of unordered pairs $\{i, j\}$ with $V_{ij} \neq 0$.

Our algorithm will address the *ferromagnetic* case of the Ising model, which is characterised by the interaction energies V_{ij} being nonnegative. Furthermore, rather than attempting to evaluate the partition function *exactly*, we shall content ourselves with a close approximation. The phrase "close approximation" will be given a precise meaning in the next section.

One strategy, which has been applied successfully to problems of this type and has, for example, been used to estimate the partition function of a general monomer-dimer system [27], [16], involves the simulation of an appropriately defined Markov chain. A direct application of this strategy to the Ising partition function would proceed as follows. View the configurations of the Ising system, namely, the 2^n possible spin vectors $\sigma \in \{-1, +1\}^n$, as the states of a Markov chain. Choose transition probabilities between states so that the Markov chain is ergodic and so that, in the stationary distribution, the probability of being in state σ is $Z^{-1}\exp(-\beta H(\sigma))$. A reasonable way to achieve this, and one which is often used in practice, is to allow transitions to occur between spin configurations which differ in just one component, and to choose transition probabilities according to the Metropolis rule [20]. If the resulting Markov chain is *rapidly mixing*, that is, if it converges rapidly to the stationary distribution regardless of the choice of initial state, then it can be used effectively to sample configurations σ from a distribution which is close to the stationary distribution. By collecting enough sample configurations, using different values of B and β, it should then be possible to estimate the partition function Z with good accuracy.

Unfortunately, it transpires that the Markov chain described, and which we refer to as the *spins-world* process, is *not* rapidly mixing. It is well known that ferromagnetic Ising systems typically exhibit a phase transition at a certain value of the parameter β; for values of β above this critical value, the system settles into a state in which there is a preponderance of spins of one or other sign. Transitions between the majority $+1$ states and majority -1 states occur very infrequently, simply because the stationary distribution assigns small total weight to the configurations with balanced spins. (Informally, the state space has a constriction separating the majority $+1$ states and the majority -1 states.) Although it could be argued that the barrier to rapid mixing just described is somewhat trivial, there exist other more subtle barriers that apparently cannot be surmounted.

The problem caused by the absence of rapid mixing in the spins-world process can be circumvented by simulating a different Markov chain, which we refer to as the *subgraphs-world* process. The two Markov chains are structurally very different; furthermore, the subgraphs-world process has, as far as we are aware, no direct physical significance. However, the subgraphs-world process has a close connection with the Ising partition function and, crucially in the current application, is rapidly mixing. For the time being, we content outselves with describing the subgraphs-world configurations and associated partition function. The description of the subgraphs-world process itself is deferred to §4.

We say that a subgraph is *spanning* if it includes all the vertices of the parent graph. (Note that spanning subgraphs are not, in general, connected.) The subgraphs-world configurations are spanning subgraphs of the interaction graph $([n], E)$. In the sequel we shall drop the adjective "spanning" where it seems safe to do so, and frequently identify a spanning subgraph $([n], X)$ with the set X of edges which define it. To simplify notation, let

$$(2) \qquad \lambda_{ij} = \tanh \beta V_{ij} \quad \text{and} \quad \mu = \tanh \beta B$$

Each configuration $X \subseteq E$ is assigned a *weight* according to the formula

$$(3) \qquad w(X) = \mu^{|\mathrm{odd}(X)|} \prod_{\{i,j\} \in X} \lambda_{ij},$$

where the notation $\mathrm{odd}(X)$ stands for the set of all odd-degree vertices in the graph X. The subgraphs-world partition function is simply

$$(4) \qquad Z' = \sum_{X \subseteq E} w(X).$$

The above sum is generally known as the "high-temperature expansion."

It is a surprising fact that the spins- and subgraphs-world partition functions Z and Z' are related in a simple way. Define

$$(5) \qquad A = (2 \cosh \beta B)^n \prod_{\{i,j\} \in E} \cosh \beta V_{ij},$$

and note that A is an easily computed function of the parameters that specify the Ising system. The following classical result [26] relates the two partition functions.

THEOREM 1. $Z = AZ'$.

In recognition of the central role it plays in our algorithm, we present a full proof of this result below.

Theorem 1 prompts us to consider a statistical mechanical system whose configurations are spanning subgraphs of $([n], E)$. We shall define a Markov chain whose states are these configurations, and whose stationary distribution assigns probability $\pi(X) = w(X)/Z'$ to configuration X. This subgraphs-world process will be analysed in detail in §4 and shown to be rapidly mixing. Hence it will provide us with an efficient means of sampling subgraphs-world configurations with probabilities roughly proportional to their weights. Since Z' is a weighted sum of the configurations, we might expect such a procedure to give us useful information about Z' itself, and hence about the original spins-world partition function Z. The next section confirms that this is indeed the case.

Proof of Theorem 1. Taking (1) as a starting point, apply the identity $e^x = \cosh x(1 + \tanh x)$ to recast the partition function in the form

$$Z = 2^{-n} A \sum_{\sigma \in \{-1,+1\}^n} \prod_{\{i,j\} \in E} \{1 + \tanh(\beta V_{ij} \sigma_i \sigma_j)\} \prod_{k \in [n]} \{1 + \tanh(\beta B \sigma_k)\},$$

where A is defined in (5). Note that the spin variables σ_k disappear from the expression for A because $\sigma_k = \pm 1$ for all k, and $\cosh x$ is an even function. Similarly, since $\tanh x$ is an odd function, the spin variables may be brought outside allowing Z to be rewritten as

$$Z = 2^{-n} A \sum_{\sigma} \prod_{\{i,j\} \in E} \{1 + \sigma_i \sigma_j \tanh \beta V_{ij}\} \prod_{k \in [n]} \{1 + \sigma_k \tanh \beta B\}.$$

Expanding the two products, and changing variables according to (2), we obtain

$$Z = 2^{-n} A \sum_{\sigma} \sum_{X \subseteq E} \left(\prod_{\{i,j\} \in X} \lambda_{ij} \sigma_i \sigma_j \right) \sum_{U \subseteq [n]} \left(\prod_{k \in U} \mu \sigma_k \right),$$

which, on interchanging the order of summation, yields

$$Z = 2^{-n} A \sum_{X \subseteq E} \sum_{U \subseteq [n]} \sum_{\sigma} W(U, X, \sigma), \tag{6}$$

where

$$W(U, X, \sigma) = \prod_{k \in U} \mu \sigma_k \prod_{\{i,j\} \in X} \lambda_{ij} \sigma_i \sigma_j .$$

Now we claim that $\sum_{\sigma} W(U, X, \sigma) = 0$ unless X is a graph in which all vertices in U have odd degree, and all vertices in $[n] - U$ have even degree. To see this, fix U and X, and let $k \in [n]$ be such that *either* $k \in U$ and has even degree in X *or* $k \in [n] - U$ and has odd degree in X. For any vector $\sigma \in \{-1, +1\}^n$, let $\sigma^{(k)}$ denote the vector derived from σ by inverting the sign of the kth component. Then the terms $W(U, X, \sigma)$ and $W(U, X, \sigma^{(k)})$ are equal in size but opposite in sign. Hence the terms of the sum $\sum_{\sigma} W(U, X, \sigma)$ cancel out in pairs.

Conversely, suppose that X *is* a graph in which all vertices in U have odd degree and all vertices in $[n] - U$ have even degree. Then, for all $\sigma \in \{-1, +1\}^n$ and $k \in [n]$, the terms $W(U, X, \sigma)$ and $W(U, X, \sigma^{(k)})$ are equal. Thus the value of $W(U, X, \sigma)$ is independent of σ and

$$\sum_{\sigma} W(U, X, \sigma) = 2^n \mu^{|U|} \prod_{\{i,j\} \in X} \lambda_{ij} = 2^n w(X).$$

Finally, substituting for $\sum_{\sigma} W(U, X, \sigma)$ in (6) we obtain the identity $Z = A \sum_{X \subseteq E} w(X)$, as required. □

3. Estimating the partition function. The aim of this section is to present an efficient approximation algorithm for computing the partition function Z of a ferromagnetic Ising system. The section is structured as follows. First, we define precisely what we mean by an efficient approximation algorithm. Then we state, without proof, the properties of the sampling procedure for subgraphs-world configurations which plays a key role in our algorithm: the construction and analysis of this procedure, based on a suitably defined Markov chain, is left to the next section. Finally, we explain how to use samples produced by this procedure to obtain a reliable approximation of Z.

Our definition of an efficient approximation algorithm is a very demanding one, following Karp and Luby [18] and others. For nonnegative real numbers $a, \tilde{a}, \epsilon$, we say that $\tilde{a}$ *approximates a within ratio* $1 + \epsilon$ if $a(1+\epsilon)^{-1} \leq \tilde{a} \leq a(1+\epsilon)$. Let f be any function from problem instances to real numbers. (The Ising partition function is an example of such a function.) A *randomised approximation scheme* for f is a probabilistic algorithm which, when presented with an instance x and a real number $\epsilon \in (0, 1]$, outputs a number which, with high probability, approximates $f(x)$ within ratio $(1 + \epsilon)$. We shall take the phrase "with high probability" to mean with probability at least 3/4. This is because a failure probability of 1/4 can be reduced to any desired value $\delta > 0$ by performing only $O(\log \delta^{-1})$ trials and taking the median of the results [17]. (This claim is also justified in the following proof of Lemma 3.) Of course, it is not enough just to obtain an accurate result with high reliability; the result must also be obtained efficiently. Accordingly, we call an approximation scheme *fully polynomial* if it runs in time polynomial in ϵ^{-1} and the size of the problem instance x. The reader will appreciate that a fully polynomial randomised approximation scheme, or *fpras*, embodies a strong notion of efficient approximation.

With an eye to simplicity of presentation, we shall not concern ourselves with the errors which arise through the inexact nature of computer arithmetic. Instead, we shall assume a computational model in which real arithmetic is performed with perfect accuracy, and in which arithmetic operations and standard functions, such as exp, are charged at unit cost. After all, we are aiming only at an *approximate* evaluation of the partition function, and it will become apparent that our technique does not rely on intermediate computations being carried out to untoward accuracy. Again with simplicity in mind, we will take n, the number of sites, as the size of the problem instance, even though the number of parameters to the model would be a more reasonable measure from an information-theoretic point of view.

As we already mentioned, our approximation algorithm for Z is based on a sampling procedure for subgraphs-world configurations. We must now be more precise about the properties of the sampling procedure. For a ferromagnetic Ising system $\langle \lambda_{ij}, \mu \rangle$, with λ_{ij} and μ as defined in (2) of the previous section, let Ω denote the set of subgraphs-world configurations, i.e., the set of spanning subgraphs of the interaction graph $([n], E)$, and define the probability distribution π over Ω by $\pi(X) = w(X)/\sum_{X'} w(X') = w(X)/Z'$, where w is the weight function defined in (3). (Note that, since the system is ferromagnetic, $w(X) \geq 0$ for all $X \in \Omega$, so π is a probability distribution.) We wish to formalise the notion of an algorithm which, given a ferromagnetic system, selects a configuration from a distribution which is "close to" π. A *generator* for subgraphs-world configurations is a probabilistic algorithm which takes as input a ferromagnetic Ising system in the form $\langle \lambda_{ij}, \mu \rangle$, plus a positive real tolerance δ, and outputs an element of Ω drawn from a distribution p satisfying

$$\|p - \pi\| \leq \delta.$$

Here $\|\cdot\|$ denotes variation distance, i.e.,

$$\|p - \pi\| = \frac{1}{2} \sum_{X \in \Omega} |p(X) - \pi(X)| = \max_{A \subseteq \Omega} |p(A) - \pi(A)|.$$

It turns out to be possible to construct an efficient generator for subgraphs-world configurations, as the following theorem states.

THEOREM 2. *There exists a generator for subgraphs-world configurations which, on inputs $\langle \lambda_{ij}, \mu \rangle$ and δ, runs in time bounded by a polynomial in n, μ^{-1}, and $\log \delta^{-1}$. Specifically, the runtime of the generator is $O(m^2 \mu^{-8}(\log \delta^{-1} + m))$, where $m = |E|$ is the number of nonzero interactions.*

Remarks. (i) The presence of μ^{-1} in the time bound implies that the generator is inefficient for systems with a very small external field. This dependence on the field is inessential and can be removed with a little extra work (see Theorem 10 of §5).

(ii) Efficient generators for combinatorial structures, of which the above is a particular example, are discussed in a general framework in [27], [29].

The construction of a generator with the above properties, based on simulation of a suitably defined Markov chain, is described and justified in detail in the next section. For the moment we will simply assume Theorem 2 and concentrate on showing how samples produced by the generator can be used to obtain an efficient approximation algorithm for the partition function $Z(V_{ij}, B, \beta)$. Our approach, which we now describe, is an instance of a computational technique which will be employed repeatedly in this paper.

Suppose we want to estimate the value of some physical quantity associated with a ferromagnetic Ising system. The first step is to express the quantity as the expectation of a suitably defined random variable over configurations in the subgraphs world. Then we can estimate the quantity by sampling configurations at random, with the aid of the generator of Theorem 2, and by computing the sample mean.

More precisely, let f be a nonnegative real-valued function defined on the set Ω of subgraphs-world configurations of a ferromagnetic Ising system. Viewing Ω as a sample space with probability distribution $\pi(X) = w(X)/Z'$, the function f becomes a random variable with expectation

$$\mathbf{E}(f) = \frac{1}{Z'} \sum_{X \in \Omega} w(X) f(X).$$

It is a simple matter to get an estimate of $\mathbf{E}(f)$ using the generator of Theorem 2: Construct an independent sample $\{X_i\}$ of configurations, of size s, and compute the sample mean $s^{-1} \sum_i f(X_i)$. Provided the tolerance input to the generator is small, this will be an almost unbiased estimator of $\mathbf{E}(f)$. By making the sample size s large enough, we can achieve any desired degree of accuracy with reasonable confidence. Moreover, we may drastically reduce the probability that the estimator falls outside the acceptable range of accuracy by repeating the entire process t times and taking the median of the t results.

The efficiency of such an experiment depends on how large the numbers s and t must be in order to achieve a specified accuracy with specified confidence. This in turn depends on the variance of the random variable f, or more precisely on the quantity $\max(f)/\mathbf{E}(f)$, where $\max(f)$ denotes the maximum value of f on Ω. The next lemma quantifies these effects; the proof is straightforward and is left until the end of the section.

LEMMA 3. *Let f be a nonnegative real-valued random variable defined on the set Ω of subgraphs-world configurations of a ferromagnetic Ising system, and let ξ, η be real numbers with $0 < \xi \leq 1$ and $0 < \eta \leq 1/2$. Then there is an experiment of the form described above which uses a total of $504\xi^{-2}\lceil \lg \eta^{-1} \rceil \max(f)/\mathbf{E}(f)$ samples from the generator, each with input $\langle \lambda_{ij}, \mu \rangle$ and tolerance $\delta = \xi \mathbf{E}(f)/8 \max(f)$, and produces an output Y satisfying*

$$\Pr(Y \text{ approximates } \mathbf{E}(f) \text{ within ratio } 1 + \xi) \geq 1 - \eta.$$

Lemma 3 makes it clear that, whenever we employ the above technique, we will need to ensure that the ratio $\max(f)/\mathbf{E}(f)$ is not too large for the random variable f under consideration. In particular, our criterion for efficiency demands that the ratio be bounded by a polynomial function of n, the size of the system.

We turn now to an explanation of how the technique can be applied to compute the partition function $Z(V_{ij}, B, \beta)$. Recall from Theorem 1 of the previous section that $Z = AZ'$, where A is simple to evaluate directly. We therefore concentrate on computing Z'. Our first step is to write Z' explicitly as a function of μ as follows:

$$Z' \equiv Z'(\mu) = \sum_{X \subseteq E} \mu^{|\mathrm{odd}(X)|} \prod_{\{i,j\} \in X} \lambda_{ij} = \sum_{k=0}^{\lfloor n/2 \rfloor} c_k \mu^{2k}. \tag{7}$$

Note that only even powers of μ need be included in the sum since the number of odd-degree vertices in a subgraph X is necessarily even. We are thus viewing Z' as a polynomial in μ^2 with coefficients

$$c_k = \sum_{X : |\mathrm{odd}(X)| = 2k} \prod_{\{i,j\} \in X} \lambda_{ij}. \tag{8}$$

In the ferromagnetic case all the c_k are positive, so $Z'(\mu)$ is an increasing function of μ.

Clearly the coefficients c_k actually depend on the λ_{ij}, and hence on the interactions V_{ij} of the system and on the parameter β. However, in what follows we will regard these quantities, and therefore also the coefficients, as fixed, and consider what happens when μ is varied. In spins-world terminology, this corresponds to subjecting a system with fixed interactions and at a fixed temperature to a varying external field. Our task is to evaluate the partition function at a specified external field value B. By the above discussion, this is reduced to evaluating the polynomial $Z'(\mu) = \sum c_k \mu^{2k}$ at the point $\mu = \tanh \beta B$.

Our starting point is the observation that the value of $Z'(\mu)$ at $\mu = 1$ can be computed directly. To see this, note from (7) and (8) that

$$Z'(1) = \sum_{k=0}^{\lfloor n/2 \rfloor} c_k = \sum_{X \subseteq E} \prod_{\{i,j\} \in X} \lambda_{ij} = \prod_{\{i,j\} \in E} (1 + \lambda_{ij}). \tag{9}$$

We are now going to relate the desired value $Z'(\tanh \beta B)$ to $Z'(1)$ using the values of $Z'(\mu)$ at certain intermediate points $\tanh \beta B < \mu < 1$.

The mechanism for relating the values of Z' at two points $\mu = \mu_0$ and $\mu = \mu_1$, with $1 \geq \mu_0 > \mu_1 \geq 0$, is the following. Consider the random variable $f(X) = (\mu_1/\mu_0)^{|\mathrm{odd}(X)|}$ over configurations of the system at $\mu = \mu_0$. The expectation of f is given by

$$\mathbf{E}_{\mu_0}(f) = \frac{1}{Z'(\mu_0)} \sum_{k=0}^{\lfloor n/2 \rfloor} c_k \mu_0^{2k} \left(\frac{\mu_1}{\mu_0} \right)^{2k} = \frac{Z'(\mu_1)}{Z'(\mu_0)}.$$

(Here and in the sequel we will use notation such as $\mathbf{E}_{\mu_0}(f)$ to indicate that the expectation is taken with respect to a particular value of μ, in this case μ_0). Hence we can estimate the quantity $Z'(\mu_1)/Z'(\mu_0)$ using the sampling technique discussed earlier. By Lemma 3, this process will be efficient provided the ratio $\max(f)/\mathbf{E}_{\mu_0}(f)$ is not too large. Clearly, this cannot be guaranteed for arbitrary values of μ_0 and μ_1. However, if the values are reasonably close together, then the ratio is bounded rather tightly, as we now show. First, note that certainly $\max(f) \leq 1$. It is therefore enough to obtain a lower bound on the expectation $\mathbf{E}_{\mu_0}(f)$. Such a bound is provided by the next lemma, whose proof we defer to the end of the section.

LEMMA 4. *Let μ_0 and μ_1 be arbitrary real numbers in the range* $[0, 1]$ *satisfying $\mu_1 < \mu_0 \leq \mu_1 + n^{-1}$. Then the ratio $Z'(\mu_1)/Z'(\mu_0)$ is bounded below by* $1/10$.

Lemma 4 suggests that we should be able to bootstrap the known value $Z'(1)$ to the desired value $Z'(\tanh \beta B)$ by performing statistical experiments at a sequence of intermediate values of μ which are a distance n^{-1} apart. Specifically, let $r < n$ be the natural number satisfying

$$\frac{n-r}{n} > \tanh \beta B \geq \frac{n-r-1}{n}, \tag{10}$$

and define the sequence (μ_k) for $0 \leq k \leq r+1$ by

$$\mu_k = \begin{cases} (n-k)/n & \text{for } \quad 0 \leq k \leq r; \\ \tanh \beta B & \text{for } \quad k = r+1. \end{cases} \tag{11}$$

Note that $\mu_k \in [0, 1]$ and $\mu_{k+1} < \mu_k \leq \mu_{k+1} + n^{-1}$. Hence by the above discussion we may estimate the ratio $Z'(\mu_{k+1})/Z'(\mu_k)$ efficiently for each k. This is enough to yield an estimate of $Z'(\tanh \beta B)$, since we have

$$Z'(\tanh \beta B) = Z'(1) \times \prod_{k=0}^{r} \frac{Z'(\mu_{k+1})}{Z'(\mu_k)}. \tag{12}$$

We are now in a position to give our approximation algorithm for the partition function Z. We assume that the input consists of a ferromagnetic Ising system in the form $\langle V_{ij}, B, \beta \rangle$, and a positive real $\epsilon \in (0, 1]$ which specifies the desired accuracy. As usual, we set $\lambda_{ij} = \tanh \beta V_{ij}$.

Step 1. Compute $A = (2 \cosh \beta B)^n \prod_{\{i,j\} \in E} \cosh \beta V_{ij}$, and $Z'(1) = \prod_{\{i,j\} \in E}(1 + \lambda_{ij})$.

Step 2. Define the sequence (μ_k) for $0 \le k \le r + 1$ as in (10) and (11) above. For each $k = 0, 1, \dots, r$ in turn, do the following:

> Let $f(X) = (\mu_{k+1}/\mu_k)^{|\mathrm{odd}(X)|}$ for each subgraphs-world configuration X, so that $\mathbf{E}_{\mu_k}(f) = Z'(\mu_{k+1})/Z'(\mu_k)$. Using the technique of Lemma 3 applied to the system at $\mu = \mu_k$, with $\xi = \epsilon/2n$ and $\eta = 1/4n$, compute a quantity Y_k satisfying
>
> $$\Pr(Y_k \text{ approximates } Z'(\mu_{k+1})/Z'(\mu_k) \text{ within ratio } 1 + \epsilon/2n) \ge 1 - 1/4n.$$

Step 3. Output the product

$$A \times Z'(1) \times \prod_{k=0}^{r} Y_k.$$

THEOREM 5. *The above algorithm is an* fpras *for the partition function Z of a ferromagnetic Ising system.*

Proof. The output of the algorithm is the product of the quantities A and $Z'(1)$, computed exactly in Step 1, together with $r + 1 \le n$ random variables Y_k arising from experiments in Step 2. From (12) and the property of the Y_k expressed in Step 2, it is immediate that the product approximates $Z(V_{ij}, B, \beta)$ within ratio $(1 + \epsilon/2n)^n \le 1 + \epsilon$ with probability at least $(1 - 1/4n)^n \ge 3/4$. It remains only to show that the runtime of the algorithm is bounded by a polynomial in n and ϵ^{-1}.

Steps 1 and 3 can clearly be executed in time $O(n^2)$. Now consider the operation of Step 2 for a particular value of k. Appealing to Lemmas 3 and 4, we see that the process of computing the estimate Y_k requires $N = 20160\epsilon^{-2}n^2\lceil \lg 4n \rceil$ calls to the generator of Theorem 2. Moreover, the tolerance supplied on each call is $\delta = \epsilon/160n$, and the value of μ is never less than n^{-1}. It follows from Theorem 2 that the runtime of each call is bounded by $q(n, \epsilon^{-1})$ for some polynomial $q(\cdot, \cdot)$. The total execution time of Step 2 is therefore $O(nNq(n, \epsilon^{-1}))$, which is a polynomial function of n and ϵ^{-1}. The algorithm therefore satisfies all the requirements of an fpras. □

Remarks. (i) The statement of Theorem 2 actually gives an upper bound on the polynomial q appearing at the end of the above proof. From this, it is easily seen that the overall runtime of the fpras of Theorem 5 is $O(\epsilon^{-2}m^2n^{11} \log n(\log(\epsilon^{-1}n)+m))$. Now we may assume, without loss of generality, that $\epsilon \ge 2^{-m}$, since otherwise we can evaluate Z exactly by brute force in time $O(m\epsilon^{-1})$. Hence the expression for the runtime simplifies to $O(\epsilon^{-2}m^3n^{11} \log n)$. (See also the remark following the proof of Theorem 7 at the end of the next section.)

(ii) Closer analysis reveals that the sequence of coefficients (c_k) of the polynomial expression (7) for Z' is *log-concave*, i.e.,

$$c_{k+1}c_{k-1} \le c_k^2 \quad \text{for} \quad k = 1, 2, \dots, \lfloor n/2 \rfloor - 1.$$

(The proof makes use of the ideas introduced in the proof of Theorem 7 of the next section.) This is a surprising result in its own right, since naturally occurring log-concave sequences are quite rare in combinatorics. It also suggests an alternative method for approximating Z: By log concavity, for each k it is possible to choose a value of μ which assigns to configurations with precisely k pairs of odd-degree vertices the largest aggregated weight. This in turn means that we can read off all significant coefficients c_k by sampling at appropriate values of μ, again using $Z'(1)$ as a reference value. (An analogous approach was used in [27], [16] to obtain the coefficients of a polynomial related to matchings, or monomer-dimer configurations, in a graph.) This method is both more complex and rather less efficient than the one presented in Theorem 5. However, it does supply more detailed information about Z, in the form of the coefficients of Z'. We have been unable to determine whether these quantities have any inherent physical significance, so we will not present the alternative algorithm in detail here.

We close the section by providing the missing proofs of Lemmas 3 and 4.

Proof of Lemma 3. Let $\mathrm{Var}(f)$ denote the variance of f, i.e., $\mathrm{Var}(f) = \mathbf{E}(f^2) - \mathbf{E}(f)^2$. The generator of Theorem 2 selects elements of Ω from a distribution p which is slightly different from π. Accordingly, define the mean and variance of f with respect to this distribution by

$$\mathbf{E}'(f) = \sum_{X\in\Omega} p(X)f(X);$$
$$\mathrm{Var}'(f) = \sum_{X\in\Omega} p(X)f(X)^2 - \mathbf{E}'(f)^2.$$

Since the variation distance satisfies $\|p - \pi\| \le \delta$, we have

$$\begin{aligned} |\mathbf{E}(f) - \mathbf{E}'(f)| &\le \delta \max(f) = \xi\mathbf{E}(f)/8; \\ |\mathrm{Var}(f) - \mathrm{Var}'(f)| &\le 3\delta \max(f)^2 = 3\xi\mathbf{E}(f)\max(f)/8. \end{aligned} \tag{13}$$

Now let $\{X_i\}$ be an independent sample of size s produced by the generator, and let $Y_0 = s^{-1}\sum_i f(X_i)$ be the sample mean. Clearly Y_0 has expectation $\mathbf{E}'(f)$ and variance $s^{-1}\mathrm{Var}'(f)$. Therefore, by Chebyshev's inequality we have

$$\Pr\left(|Y_0 - \mathbf{E}'(f)| > \tfrac{\xi}{3}\mathbf{E}'(f)\right) \le \frac{9}{\xi^2}\frac{\mathrm{Var}'(f)}{s\mathbf{E}'(f)^2}. \tag{14}$$

But if $|Y_0 - \mathbf{E}'(f)| \le \frac{\xi}{3}\mathbf{E}'(f)$ then, from (13),

$$\begin{aligned} |Y_0 - \mathbf{E}(f)| &\le |Y_0 - \mathbf{E}'(f)| + |\mathbf{E}'(f) - \mathbf{E}(f)| \\ &\le \tfrac{\xi}{3}\mathbf{E}'(f) + \tfrac{\xi}{8}\mathbf{E}(f) \\ &\le \tfrac{\xi}{3}\left(1 + \tfrac{\xi}{8}\right)\mathbf{E}(f) + \tfrac{\xi}{8}\mathbf{E}(f) \\ &\le \tfrac{\xi}{2}\mathbf{E}(f). \end{aligned} \tag{15}$$

Note that this in turn implies that Y_0 approximates $\mathbf{E}(f)$ within ratio $1+\xi$. Moreover, applying (13) again we have

$$\frac{\mathrm{Var}'(f)}{\mathbf{E}'(f)^2} \le \frac{\mathrm{Var}(f) + \frac{3}{8}\mathbf{E}(f)\max(f)}{\left(\frac{7}{8}\mathbf{E}(f)\right)^2} \le \frac{\frac{11}{8}\mathbf{E}(f)\max(f)}{\left(\frac{7}{8}\mathbf{E}(f)\right)^2} < \frac{2\max(f)}{\mathbf{E}(f)}, \tag{16}$$

where in the second inequality we have used the distribution-independent bound $\mathrm{Var}(f) \le \mathbf{E}(f)\max(f)$, valid for any nonnegative random variable f. Combining (15) and (16) with (14), and choosing sample size $s = 72\xi^{-2}\max(f)/\mathbf{E}(f)$, gives

$$\Pr(Y_0 \text{ approximates } \mathbf{E}(f) \text{ within ratio } 1+\xi) \le \frac{18}{\xi^2 s}\frac{\max(f)}{\mathbf{E}(f)} = \frac{1}{4}. \tag{17}$$

Now consider performing the above experiment an odd number t times, independently, and let Y denote the median of the resulting t values of Y_0. In view of (17), the probability that Y fails to approximate $\mathbf{E}(f)$ within ratio $1+\xi$ is at most

$$\sum_{i=(t+1)/2}^{t} \binom{t}{i} \left(\frac{1}{4}\right)^{i} \left(\frac{3}{4}\right)^{t-i} \le \left(\frac{1}{4}\right)^{t/2} \left(\frac{3}{4}\right)^{t/2} \sum_{i=(t+1)/2}^{t} \binom{t}{i} \le \left(\frac{3}{16}\right)^{t/2} 2^t = \left(\frac{3}{4}\right)^{t/2}.$$

Taking $t = 6\lceil \lg \eta^{-1} \rceil + 1$, this probability is bounded above by $\eta^{3\lg(4/3)} < \eta$. The random variable Y therefore satisfies the requirement of the lemma. The total number of samples required from the generator is st, which is bounded above by $504\xi^{-2}\lceil \lg \eta^{-1} \rceil \max(f)/\mathbf{E}(f)$ as claimed. $\square$

Proof of Lemma 4. We split the argument into two cases.

Case I. $\mu_0 \ge 3/4$. In this case, we have

$$\frac{\mu_1}{\mu_0} \ge 1 - \frac{1}{n\mu_0} \ge 1 - \frac{4}{3n}.$$

Therefore, since $Z'(\mu) = \sum_{k=0}^{\lfloor n/2 \rfloor} c_k \mu^{2k}$, and all coefficients c_k are positive,

$$\frac{Z'(\mu_1)}{Z'(\mu_0)} \ge \left(\frac{\mu_1}{\mu_0}\right)^{2\lfloor n/2 \rfloor} \ge \left(1 - \frac{4}{3n}\right)^n \ge \frac{1}{9},$$

assuming as we may that $n \ge 2$. (The problem is trivial otherwise.)

Case II. $\mu_0 < 3/4$. This case is handled by appealing to the original spins-world expansion of Z. First note from the definition (4) of Z' that

$$\frac{Z'(\mu_1)}{Z'(\mu_0)} = \frac{Z(V_{ij}, B_1, \beta)}{Z(V_{ij}, B_0, \beta)} \times \left(\frac{2\cosh \beta B_0}{2\cosh \beta B_1}\right)^n \ge \frac{Z(V_{ij}, B_1, \beta)}{Z(V_{ij}, B_0, \beta)}, \tag{18}$$

where $\beta > 0$ is arbitrary and B_0, B_1 are defined by $\mu_i = \tanh \beta B_i$. Note that $B_0 \ge B_1$. Moreover, the upper bound $\mu_0 - \mu_1 \le n^{-1}$ translates to a bound on $B_0 - B_1$ via the inequality $\tanh x - \tanh y \ge (x-y)\operatorname{sech}^2 x$, valid for $x \ge y \ge 0$. We get

$$\beta(B_0 - B_1) \le (\mu_0 - \mu_1)/\operatorname{sech}^2 \beta B_0 \le 16/7n, \tag{19}$$

where we used the fact that $\tanh \beta B_0 = \mu_0 < 3/4$ and $\operatorname{sech}^2 x = 1 - \tanh^2 x$. But from the definition (1) of the partition function Z we have

$$\frac{Z(V_{ij}, B_1, \beta)}{Z(V_{ij}, B_0, \beta)} \ge \min_{\sigma} \exp\left(\beta(B_1 - B_0) \sum_{k \in [n]} \sigma_k\right) \ge \exp(-n\beta(B_0 - B_1)),$$

which by (19) is bounded below by $\mathrm{e}^{-16/7}$ and hence by 1/10. Together with (18) this yields the desired bound on $Z'(\mu_1)/Z'(\mu_0)$. $\square$

4. An analysis of the subgraphs-world process.

4. An analysis of the subgraphs-world process. We shall assume that the reader is familiar with the elementary theory of finite Markov chains in discrete time: An introduction can be found, for example, in [10, Chap. XV].

Assume $\mu > 0$. Taking our cue from the form of (4), we define the subgraphs-world process, $\mathcal{MC}_{\text{Ising}}$, as follows. The state space, Ω, of the Markov chain $\mathcal{MC}_{\text{Ising}}$ is the set of all spanning subgraphs $X \subseteq E$; note that $|\Omega| = 2^m$, where $m = |E|$ is the number of unordered pairs $\{i, j\}$ with $\lambda_{ij} \neq 0$. For $X, X' \in \Omega$ with $X \neq X'$, the transition probability from X to X' is given by

$$p(X, X') = \begin{cases} 1/2m & \text{if} \quad |X \oplus X'| = 1 \quad \text{and} \quad w(X') \geq w(X); \\ w(X')/2mw(X) & \text{if} \quad |X \oplus X'| = 1 \quad \text{and} \quad w(X') < w(X); \\ 0 & \text{otherwise,} \end{cases}$$

where $X \oplus X'$ denotes the symmetric difference of X and X'. The self-loop probabilities $p(X, X)$ are defined implicitly by complementation, so that $p(X, X) = 1 - \sum_{X' \neq X} p(X, X')$. Thus, transitions in $\mathcal{MC}_{\text{Ising}}$ are perturbations in which a single edge is added to, or deleted from a subgraph. Note that exactly m transitions are available from any state, and all transition probabilities are bounded above by $1/2m$. Hence the transition probabilities are well defined, and the self-loop probabilities $p(X, X)$ for each state X are bounded below by 1/2.

We pause to observe that the above chain is very easy to simulate. Suppose the current state of the chain is $X \in \Omega$. Then the transitions from X can be selected according to the following model:

1. With probability 1/2 set $X' = X$, otherwise
2. select an edge $e \in E$ uniformly at random, and let $Y = X \oplus \{e\}$ (the symmetric difference of X and $\{e\}$);
3. if $w(Y) \geq w(X)$ then set $X' = Y$; if $w(Y) < w(X)$ then with probability $w(Y)/w(X)$ set $X' = Y$, otherwise set $X' = X$.

It will be seen that this procedure correctly models the transition probabilities specified earlier. It is worth remarking that there is no need to compute the weight functions $w(X)$ and $w(Y)$ from scratch at each iteration; since Y and X differ by a single edge, the quotient $w(Y)/w(X)$ can be computed using just two multiplications.

The Markov chain $\mathcal{MC}_{\text{Ising}}$ is irreducible (all states communicate via the empty state $\emptyset$) and aperiodic (the self-loop probabilities are nonzero). Thus there is a well-defined stationary distribution on Ω which is independent of the initial state. Define $\pi : \Omega \to \mathbb{R}$ by $\pi(X) = w(X)/\sum_{X'} w(X') = w(X)/Z'$. We shall see presently that π is indeed the stationary distribution on Ω. For $X, X' \in \Omega$ define $q(X, X') = \pi(X)p(X, X')$. We claim that q is symmetric in its two arguments. If $X = X'$ then there is nothing to prove. If $|X \oplus X'| > 1$, then $p(X, X') = 0$ and hence $q(X, X') = 0$. Finally, it is straightforward to verify from the definition of the transition probability $p(X, X')$ that

$$q(X, X') = (2m)^{-1} \min\{\pi(X), \pi(X')\}, \quad \text{if} \quad |X \oplus X'| = 1. \tag{20}$$

Since q is symmetric, the so-called *detailed balance* condition holds:

$$\pi(X)p(X, X') = q(X, X') = \pi(X')p(X', X). \tag{21}$$

Suppose, as is the case here, that the function $p(\cdot\,,\cdot)$ describes the transition probabilities of an ergodic Markov chain. It is a fact [16, Lem. 2.1] that if there is *any* function $\pi : \Omega \to \mathbb{R}$ satisfying detailed balance together with the normalisation condition $\sum_{X \in \Omega} \pi(X) = 1$, then the Markov chain is (*time-*) *reversible*, and π is its stationary distribution. Thus the stationary distribution of the Markov chain $\mathcal{MC}_{\text{Ising}}$ is indeed given by $\pi(X) = w(X)/Z'$, as claimed above, and we can use the chain to sample configurations $X \in \Omega$ with probabilities approximately proportional to $w(X)$.

As explained informally earlier, if the Markov chain $\mathcal{MC}_{\text{Ising}}$ is to be the basis of an efficient sampling procedure for configurations then it must be *rapidly mixing*, in the sense that, if it is allowed to evolve from a suitable initial state, the distribution of its final state will be very close to the stationary distribution after only polynomially many steps. Note that this is a highly nontrivial requirement: since the number of states in the chain is exponentially large, we are demanding that it converge after visiting only a tiny fraction of its state space. Our argument that the chain is rapidly mixing is in two parts: First, in Theorem 6, we state a general characterisation of the rapid mixing property in terms of a measure known as the *conductance*; then, in Theorem 7, we estimate the conductance of $\mathcal{MC}_{\text{Ising}}$.

For an ergodic reversible Markov chain, the conductance [27], [29] is defined by

$$\Phi = \min \left\{ \sum_{\substack{X \in S \\ X' \notin S}} q(X, X') \Bigg/ \sum_{X \in S} \pi(X) \right\},$$

where the minimisation is over all subsets S of states with $0 < \sum_{X \in S} \pi(X) \leq 1/2$. (Note that $0 < \Phi \leq 1$.) The conductance in some sense measures the rate at which the process can flow around the state space: specifically, it provides a lower bound on the conditional probability that the stationary process escapes from a small subset S of the state space in a single step, given that it is initially in S. Thus a chain with large conductance is unlikely to "get stuck" in any small region of the state space, so we might expect it to converge fast. This intuition is captured in the following theorem.

THEOREM 6. *Let Φ be the conductance of an ergodic, reversible Markov chain with stationary distribution π and* $\min_X p(X, X) \geq 1/2$. *Let $p^{(t)}$ denote the distribution of the state at time t given that the initial state is X_0. Then the variation distance $\| p^{(t)} - \pi \|$ satisfies*

$$\| p^{(t)} - \pi \| \leq \frac{(1 - \Phi^2)^t}{\pi(X_0)}.$$

(The requirement that $\min_X p(X, X) \geq 1/2$, i.e., that every state has a self-loop probability of at least 1/2, is a technical device which removes periodicity; note that $\mathcal{MC}_{\text{Ising}}$ satisfies this requirement by construction.)

Proof. The theorem is essentially a restatement of Theorem 3.4 of [29], to which the interested reader is referred for details; we mention here only the necessary modifications. The main difference stems from the fact that in the former result we used the stronger *relative pointwise distance* (r.p.d.), rather than the variation distance, as a measure of deviation from the stationary distribution. In similar fashion to the r.p.d., the variation distance at time t may be related, by elementary linear algebra, to the second eigenvalue λ_1 of the Markov chain: we get

$$\| p^{(t)} - \pi \| \leq \frac{\lambda_1^t}{\pi(X_0)}. \tag{22}$$

(See, for example, Proposition 2 of [7], which presents a marginally stronger result, with $2\sqrt{\pi(X_0)}$ replacing $\pi(X_0)$ in the denominator. Note that the presence of a self-loop probability of 1/2 on every state ensures that all eigenvalues are nonnegative.) The bound in (22) differs from that on the r.p.d. in Lemma 3.1 of [29] only in that $\pi(X_0)$ replaces $\min_X \pi(X)$.

Now the main result of §3 of [29], Lemma 3.3, relates λ_1 to the conductance via the bound $\lambda_1 \leq 1 - \Phi^2/2$, valid for an arbitrary reversible chain. It is easily seen from the proof of the lemma that the marginally stronger bound

$$\lambda_1 \leq 1 - \Phi^2 \tag{23}$$

holds for chains in which all self-loop probabilities are at least 1/2. Putting (22) and (23) together establishes the theorem. □

Remarks. (i) The heart of the above proof is the eigenvalue bound (23). This is a discrete analogue of Cheeger's inequality for Riemannian manifolds [5]. Related bounds have been observed by several authors; see, e.g., [8], [1], [23], [25].

(ii) Theorem 6 has a converse, which says that if a Markov chain is rapidly mixing then its conductance cannot be too small; see, e.g., [23], [27], [28]. Thus the conductance provides a *characterisation* of the rapid mixing property.

Theorem 6 allows us to investigate the rate of convergence of a reversible chain by examining its transition structure, as reflected in the conductance. In particular, if we wish to ensure a variation distance of at most δ then it is clear that $\Phi^{-2}(\ln \delta^{-1} + \ln \pi(X_0)^{-1})$ steps suffice. Thus the rapid mixing property will generally follow from an inverse polynomial lower bound on the conductance. Such a bound is available for the chain $\mathcal{MC}_{\text{Ising}}$ defined above. Specifically, we have the following theorem.

THEOREM 7. *The conductance of the Markov chain $\mathcal{MC}_{\text{Ising}}$ is bounded below by $\mu^4/4m$.*

The proof of Theorem 7 is the main content of this section. Before proceeding with it, however, let us first use the result to verify our claim in Theorem 2 of the previous section that an efficient generator for subgraphs-world configurations exists. This will complete the validation of our approximation algorithm for the partition function.

Proof of Theorem 2. The generator operates as follows. Given as input a ferromagnetic Ising system in the form $\langle \lambda_{ij}, \mu \rangle$, with $0 < \mu \leq 1$, and a tolerance $\delta \in (0, 1]$, simulate the associated Markov chain $\mathcal{MC}_{\text{Ising}}$ for $16m^2\mu^{-8}(\ln \delta^{-1} + m)$ steps, starting in state $X_0 = \emptyset$ (i.e., the empty graph on vertex set $[n]$). Since $\lambda_{ij} < 1$ for all i, j, and $\mu \leq 1$, it is clear that $w(X_0) \geq w(X)$ for all configurations X. Hence $\pi(X_0) \geq 2^{-m}$. Appealing to Theorem 6, we conclude that the specified number of simulation steps is enough to ensure a variation distance of at most δ. The theorem is therefore established. □

Proof of Theorem 7. The proof rests on a path counting argument, similar to those employed in previous applications [27], [16] of the conductance bound. We present a preliminary sketch map of the proof technique before considering the technical details which arise when applying the path counting argument to the particular chain under consideration.

For each pair of states $I, F \in \Omega$, a *canonical path* from I (the initial state) to F (the final state) is specified. The canonical path proceeds via a number of intermediate states using only valid transitions of the Markov chain. Each canonical path is assigned a weight which is the product of the stationary probabilities at the initial and final states; thus the weight of the path from I to F is $\pi(I)\pi(F)$, independent of the intermediate states in the path. A careful choice of canonical paths is essential to secure a good bound on conductance.

Suppose it can be shown that, for each transition $T \to T'$, the aggregated weight of canonical paths which use transition $T \to T'$ is bounded above by $bq(T, T')$, where q is as in (21). Consider any partition of the state space Ω into two sets S and $\bar{S}$ with $\sum_{X \in S} \pi(X) \leq 1/2$. Then, on the one hand, the total weight of canonical paths which cross the cut defined by S and $\bar{S}$ is at least $\sum_{I \in S} \sum_{F \in \bar{S}} \pi(I)\pi(F) = \pi(S)\pi(\bar{S}) \geq \pi(S)/2$. On the other hand, summing over transitions $T \to T'$ with $T \in S$ and $T' \in \bar{S}$, we have that the total weight of canonical paths which cross the cut is bounded above by $b \sum_{T \in S} \sum_{T' \in \bar{S}} q(T, T')$. Since S and $\bar{S}$ represent a general partition of the state space Ω, it follows immediately that the conductance of the Markov chain is bounded below by $1/2b$.

It will be perceived that the principal barrier to applying the above idea is likely to lie in obtaining a good value for the bound b. This is achieved using a combinatorial encoding technique, as follows. For each transition $T \to T'$, let $\text{cp}(T, T')$ denote the set of all pairs $(I, F) \in \Omega^2$ such that the canonical path from I to F includes the transition $T \to T'$. Fix any particular transition $T \rightarrow T'$. Then it turns out that we can define an *injective map* from $\mathrm{cp}(T, T')$ to the state space Ω. Since the map is injective, any state $U \in \Omega$ picks out at most one canonical path, from I to F, say, which uses the transition $T \rightarrow T'$; the state U can be thought of as an *encoding* of the canonical path. Moreover, the injective map can be chosen so that the weight of the canonical path, namely, $\pi(I)\pi(F)$, is roughly proportional to $\pi(U)$, the probability assigned to the encoding U by the stationary distribution. Since π is a probability distribution, the sum of $\pi(U)$ over all encodings U is at most one; this upper bound translates to an upper bound on the total weight of paths using $T \rightarrow T'$, and hence to a value for b.

All the above must now be specialised to the Markov chain $\mathcal{MC}_{\mathrm{Ising}}$. The first task is to specify a canonical path for each pair $I, F \in \Omega$. View I and F as graphs with vertex set $[n]$. Let $\Delta = I \oplus F$ be the symmetric difference of I and F, and suppose that the graph Δ has $2k$ vertices of odd degree. (The number of odd-degree vertices in a graph is necessarily even.) Cover Δ with a collection $C_1, C_2, \ldots, C_r$ of assorted trails and circuits which are pairwise edge disjoint, imposing the condition that the first k objects, $C_1, C_2, \ldots, C_k$, are all open *trails* (walks with no repeated edges) while the remainder, $C_{k+1}, C_{k+2}, \ldots, C_r$, are all *circuits* (closed trails).

That this can be done with so few (open) trails follows from a simple induction on k. When $k = 0$, every vertex of Δ is of even degree, so each connected component of Δ is Eulerian and can be covered by a single circuit. Now suppose $k > 0$ and i is a vertex in Δ of odd degree. The connected component of Δ containing i must contain at least one other odd-degree vertex, say, j. Connect i and j by a trail, letting this be one of the trails in the required decomposition of Δ. Deleting this trail from Δ yields a graph with $2(k-1)$ odd-degree vertices, which can be covered by $k-1$ trails (together with some number of circuits) by the induction hypothesis.

The covering of Δ by trails and circuits is not in general unique, and we assume that some rule is employed to pick out a particular choice of $C_1, C_2, \ldots, C_r$. We further assume that this rule also specifies a distinguished *initial vertex* for each trail or circuit, and a *direction* for each circuit. In the case of a trail the initial vertex must be an endpoint of the trail; in the case of a circuit the initial vertex may be arbitrary. The canonical path from I to F is now obtained by *unwinding* the trails and circuits $C_1, C_2, \ldots, C_r$ in sequence. Unwinding C_i involves processing each edge of C_i in sequence, starting at the initial vertex and, in the case of a circuit, following the assigned direction. Each processed edge, e, generates a single transition on the path from I to F. If e is in F (and hence not in I) the transition involves adding the edge e to the current state; if e is in I (and hence not in F) the transition involves deleting the edge e from the current state. It is clear that this process defines a canonical path of legal transitions from state I to state F.

The next task is to define the injective map (encoding) from the set of canonical paths using a given transition to Ω. Recall that $\mathrm{cp}(T, T')$ denotes the set of all pairs $(I, F) \in \Omega^2$ such that the canonical path from I to F employs the transition $T \rightarrow T'$. Define the map $\eta_{T\to T'} : \mathrm{cp}(T, T') \rightarrow \Omega$ by $\eta_{T\to T'}(I, F) = I \oplus F \oplus (T \cup T')$ for all $(I, F) \in \mathrm{cp}(T, T')$. The intention here is that the encoding should agree with I on the trails and circuits already processed, and with F elsewhere.

We verify that $\eta_{T\to T'}$ is injective by demonstrating that I and F are uniquely determined by $U = \eta_{T\to T'}(I, F)$. Indeed, given U, we can compute $U \oplus (T \cup T') = I \oplus F$ and hence the uniquely defined covering $C_1, C_2, \ldots, C_r$ of $I \oplus F$ by trails and circuits. The edge $e = T \oplus T'$, which is added or deleted by the transition $T \rightarrow T'$, points out which trail or circuit, C_i, is being unwound, and how far the unwinding of C_i has progressed. Starting at state T', we may complete the unwinding of C_i and successive trails/circuits to discover the final state F; equally well, we may use the reverse process to recover the initial state I. So the map $\eta_{T\to T'}$ is injective, as claimed.

The other property we require of the encoding is that its weight should be roughly proportional to that of the encoded path. Precisely, we require of $U = \eta_{T\to T'}(I, F)$ that

$$\pi(U)q(T, T') \geq (2m)^{-1}\mu^4\pi(I)\pi(F), \tag{24}$$

or, equivalently, multiplying through by $(Z')^2$ and using assertion (20),

$$w(U)w(T) \geq \mu^4 w(I)w(F) \quad \text{and} \quad w(U)w(T') \geq \mu^4 w(I)w(F). \tag{25}$$

The verification of the left-hand inequality will be treated in detail below; the right-hand inequality will then be shown to follow by symmetry.

For $X \in \Omega$, write $\Lambda(X) = \prod_{\{i,j\}\in X} \lambda_{ij}$, so that $w(X) = \Lambda(X)\mu^{|\mathrm{odd}(X)|}$. To verify the left-hand inequality of (25) it is enough to demonstrate separately that

$$\Lambda(U)\Lambda(T) \geq \Lambda(I)\Lambda(F), \tag{26}$$

and

$$|\mathrm{odd}(U)| + |\mathrm{odd}(T)| - |\mathrm{odd}(I)| - |\mathrm{odd}(F)| \leq 4. \tag{27}$$

(We used here the fact that $0 < \mu \leq 1$.) We deal first with (26), which is the more straightforward. From the construction of canonical paths we have $I \cap F \subseteq T \cup T' \subseteq I \cup F$, while the definition of U entails $U \oplus (T \cup T') = I \oplus F$. It follows by elementary set theory from these two observations that $U \cap (T \cup T') = I \cap F$ and $U \cup (T \cup T') = I \cup F$. Hence

$$\Lambda(U)\Lambda(T \cup T') = \Lambda(U \cup (T \cup T'))\Lambda(U \cap (T \cup T')) = \Lambda(I \cup F)\Lambda(I \cap F) = \Lambda(I)\Lambda(F),$$

which, together with $\Lambda(T \cup T') \leq \Lambda(T)$, implies (26).

We now turn to inequality (27). For $i \in [n]$ define

$$\alpha(i) = \chi_{\mathrm{odd}(U)}(i) + \chi_{\mathrm{odd}(T)}(i) - \chi_{\mathrm{odd}(I)}(i) - \chi_{\mathrm{odd}(F)}(i),$$

where χ_S denotes the characteristic function of a set S. Note that $-2 \leq \alpha(i) \leq 2$. Inequality (27) can be re-expressed as

$$\sum_{i\in[n]} \alpha(i) \leq 4. \tag{28}$$

We shall argue that $\alpha(i) \leq 0$ for all i outside a small set of exceptions. In order to discuss these exceptions, we give names to three vertices which have special significance. Denote by s the initial vertex of the circuit which is in the process of being unwound when the transition $T \to T'$ is made (s is undefined if the transition occurs on a trail). Denote by u and v the endpoints of the edge $e = T \oplus T'$ which is added or subtracted during the transition $T \to T'$; vertex u is distinguished from v by being the first to be encountered in the direction of unwinding. We shall see that the vertices s, u, and v are the only ones which can provide a positive contribution to the sum in (28).

Consider first the conditions under which $\alpha(i) = 2$ can occur. It must be the case that i has even degree in both I and F, and odd degree in both U and T. Now it is a consequence of the way canonical paths are constructed that a vertex which has even degree in both I and F will generally have even degree in the intermediate configuration T; the only exceptions are the vertex s (whose degree became odd when the unwinding of the circuit commenced) and the vertex u (whose degree was made odd by the previous transition, and whose parity is restored by the transition $T \to T'$ itself). To summarise: The case $\alpha(i) = 2$ can occur only when $i = s$ or $i = u$.

Consider now the conditions under which $\alpha(i) = 1$ can occur. This case is ruled out, with two exceptions, by simple parity considerations. Since $I \oplus F = U \oplus (T \cup T')$, the value of $\alpha(i)$ must be even unless $i = u$ or $i = v$. (These are the only points at which the set $T \cup T'$ may differ from T.) Combining this observation with the previous analysis for the case $\alpha(i) = 2$, we see that only three terms of the sum occurring in (28) can possibly be strictly positive and that the sum itself cannot exceed 5. (The worst case is achieved by setting $\alpha(s) = 2$, $\alpha(u) = 2$, and $\alpha(v) = 1$.) However, the sum cannot actually attain 5 on grounds of parity: Each of the terms appearing on the left-hand side of (27) is necessarily even. This completes the verification of (28), and hence of (27) and the left-hand inequality of (25). The right-hand inequality of (25) follows by a symmetrical argument, with the roles of T and T', and u and v, interchanged.

Finally, summing (24) over all canonical paths which employ the transition $T \to T'$ we obtain the following upper bound on the total weight of canonical paths which use $T \to T'$:

$$\sum_{(I,F)\in\mathrm{cp}(T,T')} \pi(I)\pi(F) \le \sum_{(I,F)\in\mathrm{cp}(T,T')} 2m\mu^{-4}\pi(\eta_{T\to T'}(I,F))q(T,T') \le 2m\mu^{-4}q(T,T'),$$

where the second inequality rests on the fact that $\eta_{T\to T'}$ is injective. In the notation of the sketch map presented at the beginning of the proof, $b = 2m\mu^{-4}$. Therefore, the conductance of $\mathcal{MC}_{\mathrm{Ising}}$ is bounded below by $1/2b = \mu^4/4m$, as claimed at the outset. □

Remark. The main task in the proof of Theorem 7 is to estimate the "bottleneck" measure b; this is then used to get a bound on conductance, and hence on the rate of convergence of $\mathcal{MC}_{\mathrm{Ising}}$. In fact, b can be used *directly* to obtain a significantly sharper bound on the rate of convergence; for the details, see [28]. Specifically, the runtime of the generator quoted in Theorem 2 is reduced by a factor $O(\mu^{-4})$, which improves the runtime of the approximation algorithm for the partition function (Theorem 5) by a factor of $O(n^4)$. Similar improvements apply to the runtimes of our other algorithms that make use of the generator.

5. The mean energy and mean magnetic moment. Of greater immediate importance to physicists than the partition function Z itself are the partial derivatives of ln Z with respect to β and B. The key quantities of interest are the *mean energy* $\mathcal{E} = -\partial(\ln Z)/\partial\beta$, and the *mean magnetic moment* $\mathcal{M} = \beta^{-1}\partial(\ln Z)/\partial B$. As their names suggest, both of these can be viewed in the spins world as expectations of the corresponding physical quantities.

There is no reason to suppose that an fpras for the partition function Z will directly yield an fpras for these derivatives of ln Z. However, we demonstrate in this section that polynomial-time approximation algorithms for $\mathcal{E}$ and $\mathcal{M}$ do indeed exist. The construction of these algorithms relies on the surprising fact that $\mathcal{E}$ and $\mathcal{M}$ can be viewed as expectations of appropriately defined random variables in the subgraphs world. Although some technical complications arise, it is possible to estimate these expectations more or less directly by simulating the subgraphs-world process for a polynomially bounded number of steps.

The mean magnetic moment is slightly easier to work with and we treat it first. The main result is preceded by a technical lemma, whose proof is deferred to the end of the section. Recall that in the subgraphs-world distribution, each configuration X occurs with probability $w(X)/Z'$.

LEMMA 8. *Suppose the configuration $X \in \Omega$ is randomly selected according to the subgraphs-world distribution. Then*:

(i) $\Pr(|\mathrm{odd}(X)| > 0) \le \mu^2/2$, *provided* $\sum\lambda_{ij} \ge 1$;

(ii) $\Pr(|\mathrm{odd}(X)| = 2) \ge \mu^2/10$, *provided* $\sum\lambda_{ij} \ge 1$ *and* $\mu \le n^{-1}$.

THEOREM 9. *There exists an* fpras *for the mean magnetic moment* $\mathcal{M} = \beta^{-1}\partial(\ln Z)/\partial B$, *where* Z *is the partition function of a ferromagnetic Ising system.*

Proof. We shall express the quantity $\mathcal{M}$ as an expectation in the subgraphs world by differentiating the logarithm of the expansion given in Theorem 1 with respect to B. Before doing this, it is convenient to perform some preparatory computations. Since $\mathcal{M} = 0$ when $B = 0$, we may assume that $B > 0$. Recall that $w(X) = \Lambda(X)\mu^{|\text{odd}(X)|}$, where $\mu = \tanh\beta B$ by definition, and $\Lambda(X)$ is independent of B. Then

$$\begin{aligned}\frac{\partial}{\partial B}w(X) &= \Lambda(X)|\text{odd}(X)|\mu^{|\text{odd}(X)|-1}(\text{sech}\beta B)^2\beta \\ &= \beta w(X)|\text{odd}(X)|(\tanh\beta B)^{-1}(\text{sech}\beta B)^2 \\ &= w(X)\frac{2\beta|\text{odd}(X)|}{\sinh 2\beta B}.\end{aligned}$$

Furthermore, from the definition of A in (5),

$$\frac{\partial}{\partial B}\ln A = \frac{\partial}{\partial B}n\ln\cosh\beta B = n\beta\tanh\beta B.$$

With these identities in mind, we compute $\mathcal{M}$ using the expansion of Theorem 1 as the starting point:

$$\begin{aligned}\mathcal{M} = \frac{1}{\beta}\frac{\partial}{\partial B}\ln Z &= \frac{1}{\beta}\frac{\partial}{\partial B}\ln A + \frac{1}{\beta}\frac{\partial}{\partial B}\ln Z' \\ &= n\tanh\beta B + \frac{1}{\beta Z'}\sum_X \frac{\partial}{\partial B}w(X) \\ &= n\tanh\beta B + \frac{1}{Z'}\sum_X w(X)\frac{2|\text{odd}(X)|}{\sinh 2\beta B}.\end{aligned}$$

Using, as before, the notation $\mathbf{E}(f) = (Z')^{-1}\sum_X w(X)f(X)$ to express the expectation of a random variable f in the subgraphs world, the above identity can be written more compactly as

$$\mathcal{M} = n\tanh\beta B + \frac{2}{\sinh 2\beta b}\mathbf{E}|\text{odd}(X)|. \tag{29}$$

Note that to approximate $\mathcal{M}$ within ratio $1+\epsilon$ it is enough, since both terms of (29) are positive, to estimate $\mathbf{E}|\text{odd}(X)|$ within ratio $1+\epsilon$. We propose to do this by using the Markov chain $\mathcal{MC}_{\text{Ising}}$, analysed in §4, to provide a polynomial number of sample configurations X from the subgraphs-world distribution, and returning the average of $|\text{odd}(X)|$ over the sample. As noted in the discussion preceding Lemma 3, this approach will yield an fpras for $\mathcal{M}$ provided the ratio max $|\text{odd}(X)|/\mathbf{E}|\text{odd}(X)|$ is bounded by a polynomial in n. Although such a bound often holds, a more refined approach is necessary in certain circumstances. We proceed by case analysis.

Case I. $\sum\lambda_{ij} \geq 1$. We identify two subcases, according to the size of μ.

Case I(a). $\mu \geq n^{-1}$. In this range, we may estimate $\mathcal{M}$ by direct experiment. From Lemma 8,

$$\mathbf{E}|\text{odd}(X)| \geq 2\Pr(|\text{odd}(X)| > 0) \geq \mu^2 \geq n^{-2},$$

while, clearly, $\max|\mathrm{odd}(X)| \le n$. Thus the ratio $\max|\mathrm{odd}(X)|/\mathbf{E}|\mathrm{odd}(X)|$ is bounded above by n^3.

Case I(b). $\mu < n^{-1}$. Intuitively, the problem when μ is small is that a randomly sampled configuration may, with high probability, satisfy $|\mathrm{odd}(X)| = 0$; in this situation, very many trials may be required to obtain a sufficiently accurate estimate of the expectation of $|\mathrm{odd}(X)|$. The solution is to perform experiments at an increased value of μ, say $\hat{\mu}$, at which the event $|\mathrm{odd}(X)| > 0$ occurs with sufficiently high probability. Since we shall be allowing μ to vary, it is convenient, as before, to refine our notation to make the dependence on μ explicit. In particular, $\mathbf{E}_{\hat{\mu}}(f)$ will denote the expectation of the random variable f when experiments are undertaken with μ set to some revised value $\hat{\mu}$. The unsubscripted notation $\mathbf{E}(f)$ will be reserved for expectations with respect to the original value of μ.

Set $\hat{\mu} = n^{-1}$ and define

$$f(X) = |\mathrm{odd}(X)| \left(\frac{\mu}{\hat{\mu}}\right)^{|\mathrm{odd}(X)|}.$$

Straightforward manipulations yield the identity

$$\mathbf{E}|\mathrm{odd}(X)| = \frac{Z'(\hat{\mu})}{Z'(\mu)}\mathbf{E}_{\hat{\mu}}(f), \tag{30}$$

which relates the quantity we are attempting to estimate to the expectation of f at the revised value of μ. Since $Z'(\hat{\mu})$ and $Z'(\mu)$ may be estimated by the techniques of §3, we concentrate here on the estimation of $\mathbf{E}_{\hat{\mu}}(f)$. From part (ii) of Lemma 8 we have $\Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2) \ge \hat{\mu}^2/10$; this inequality allows the expectation of f to be bounded below:

$$\mathbf{E}_{\hat{\mu}}(f) \ge 2\left(\frac{\mu}{\hat{\mu}}\right)^2 \Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2) \ge \frac{\mu^2}{5}.$$

The maximum of f, meanwhile, satisfies the crude bound $\max(f) \le (\mu/\hat{\mu})^2 n$. Thus the ratio $\max(f)/\mathbf{E}_{\hat{\mu}}(f)$ is bounded above by $5n\hat{\mu}^{-2} = 5n^3$. *Case* II. $\sum \lambda_{ij} < 1$. In this rather pathological case, the essential problem we face is that a randomly sampled configuration may, with high probability, be the empty set. As before, we shall conduct experiments at an artificially inflated value of μ and use (30) to relate the results of these experiments to the value we are attempting to estimate. This time, however, we choose to work with $\hat{\mu} = 1$.

Unfortunately, even with $\hat{\mu}$ set to 1, the highest possible value, it may still happen that the empty configuration $X = \emptyset$ occurs with very high probability. We overcome this problem by sampling from the distribution obtained by conditioning on the event $X \ne \emptyset$. With f as before, and noting that $\hat{\mu} = 1$ and $f(\emptyset) = 0$, we have

$$\begin{aligned}\mathbf{E}_{\hat{\mu}}(f) &= \sum_{k=0}^{\lfloor n/2 \rfloor} 2k\mu^{2k} \Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2k) \\ &= \Pr_{\hat{\mu}}(X \ne \emptyset) \sum_{k=1}^{\lfloor n/2 \rfloor} 2k\mu^{2k} \Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2k \mid X \ne \emptyset).\end{aligned}$$

Now $Z'(\hat{\mu}) = \prod(1 + \lambda_{ij})$, and hence

$$\Pr_{\hat{\mu}}(X \ne \emptyset) = \frac{Z'(\hat{\mu}) - 1}{Z'(\hat{\mu})} \tag{31}$$

is easy to evaluate directly. It is enough, therefore, to estimate the expectation of f with respect to the distribution of *conditional* probabilities, in which each nontrivial configuration $X \neq \emptyset$ occurs with probability $(Z'(\hat{\mu}) - 1)^{-1} \prod_{\{i,j\} \in X} \lambda_{ij}$. This conditional distribution may be sampled without recourse to Markov chain simulation, the direct method being as follows. Start with $X = \emptyset$ and perform a sequence of m trials, each trial determining whether a particular edge is to be added to X. The probability governing each trial has one of two values, depending on whether any of the previous trials have contributed an edge to X. Define $p_{ij} = \lambda_{ij}(1 + \lambda_{ij})^{-1}$, and suppose that we are about to decide whether to add a certain edge $\{i, j\}$ to X. If $X \neq \emptyset$, the edge $\{i, j\}$ is added to X with probability p_{ij}; otherwise, the edge is added to X with probability $p_{ij}[1 - \prod(1 - p_{hk})]^{-1}$, where the product is over all edges $\{h, k\}$ whose fate is yet to be decided, including edge $\{i, j\}$ itself. It is straightforward to verify that this procedure yields the required distribution. It only remains to verify that a polynomially bounded number of sample configurations suffice to provide an accurate estimate of the expectation of f. Again, we do this by demonstrating an upper bound on the ratio between the maximum of f and the mean of f. Observe that

$$(32) \qquad Z'(\hat{\mu}) = \prod(1 + \lambda_{ij}) \leq \exp\left(\sum \lambda_{ij}\right) \leq 1 + 2\sum \lambda_{ij},$$

and

$$(33) \qquad \Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2) \geq \frac{1}{Z'(\hat{\mu})} \sum \lambda_{ij},$$

where in (32) we used the fact that $e^x \leq 1 + 2x$ in the range $0 \leq x \leq 1$. Combining (31) and (33), we obtain a lower bound on the probability of $|\mathrm{odd}(X)| = 2$ conditional on $X \neq \emptyset$.

$$\begin{aligned} \Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2 \mid X \neq \emptyset) &= \frac{\Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 2)}{\Pr_{\hat{\mu}}(X \neq \emptyset)} \\ &\geq \frac{Z'(\hat{\mu})}{Z'(\hat{\mu}) - 1} \frac{1}{Z'(\hat{\mu})} \sum \lambda_{ij} \geq \frac{1}{2}; \end{aligned}$$

the final step here relies on (32). It follows immediately that the expectation of f with respect to the conditional distribution is at least μ^2. Using the crude bound $\max(f) \leq n\mu^2$, we see that the ratio of the maximum to the mean of f is bounded above by n. This completes the analysis of Case II.

We conclude by analysing the time complexity of the proposed approximation scheme for $\mathcal{M}$. The worst case is realised by Case I(b), where our method demands that the three quantities appearing on the right-hand side of identity (30) be known with sufficiently high accuracy. To obtain an fpras for $\mathcal{M}$, it is enough to estimate each of these three quantities within ratio $1 + \epsilon/4$ and with failure probability $1/12$. Setting $\xi = \epsilon/4$ and $\eta = 1/12$ in Lemma 3, we see that $O(\epsilon^{-2}n^3)$ samples from the generator suffice to estimate $\mathbf{E}_{\hat{\mu}}(f)$ within ratio $1 + \epsilon/4$ and with failure probability $1/12$. The production of each sample requires time $O(m^2\hat{\mu}^{-8}(\log \delta^{-1} + m))$, where $\delta^{-1} = O(\epsilon^{-1}n^3)$. We may assume that $\epsilon \geq 2^{-m}$; otherwise there would be time enough to evaluate $\mathcal{M}$ exactly using a brute force algorithm. With this simplifying assumption, and noting that $\hat{\mu} = n^{-1}$, the time to produce each sample is seen to be $O(m^3n^8)$, and the total time to estimate $\mathbf{E}_{\hat{\mu}}(f)$ is $O(\epsilon^{-2}m^3n^{11})$. The overall execution time for the algorithm is thus dominated by the time taken to estimate $Z'(\mu)$ and $Z'(\hat{\mu})$ within ratio $1 + \epsilon/4$ and with failure probability $1/12$; from Remark (i) following Theorem 5 this dominant term is seen to be $O(\epsilon^{-2}m^3n^{11} \log n)$. $\square$

We turn now to the mean energy $\mathcal{E}$. Up to this point, we have always sampled configurations with μ set to some value which is at least n^{-1}. In the sequel, we shall need to sample configurations at smaller values of μ; at these values Theorem 2 no longer guarantees an execution time for the sampling procedure which is polynomial in n. However, efficient sampling is possible, even at $\mu = 0$.

THEOREM 10. *There exists a generator for subgraphs-world configurations which, on inputs* $\langle \lambda_{ij}, \mu \rangle$ *and* δ, *runs in* expected *time bounded by a polynomial in* n *and* $\log \delta^{-1}$. *Specifically, the expected execution time of the generator is* $O(m^2 n^8 (\log \delta^{-1} + m))$.

Proof. The result for $\mu \geq n^{-1}$ follows directly from Theorem 2, so assume that $\mu < n^{-1}$. We employ the generator of Theorem 2 but with μ artificially boosted to $\hat{\mu} = n^{-1}$, and δ decreased to $\hat{\delta} = \delta/10$. To sample a configuration X from the distribution corresponding to the *original* value of μ, perform a sequence of trials of the following form. First produce a random configuration X using the generator of Theorem 2 (with the modified parameters); then, with probability $(\mu/\hat{\mu})^{|\mathrm{odd}(X)|}$, declare the trial successful; otherwise, declare the trial a failure. The sequence of trials is halted at the first successful trial, and the configuration X produced by that trial is returned as the result.

The probability that a trial is declared successful is at least $\Pr_{\hat{\mu}}(|\mathrm{odd}(X)| = 0)$, which by Lemma 4 (setting $\mu_1 = 0$ and $\mu_0 = n^{-1}$) is at least $1/10$. Thus the expected number of trials required to generate a configuration is at most 10. It is straightforward to check that the procedure described above, viewed as a generator with respect to the *original* value of μ, has tolerance at most $10\hat{\delta} = \delta$. □

As in the case of the mean magnetic moment, the main result rests on a technical lemma, whose proof is deferred.

LEMMA 11. *Suppose* $B = 0$, *i.e., that there is no external field. If* $w(X) \leq 1/32nm^2$ *for all* $X \neq \emptyset$, *then* $Z' = \sum_X w(X) \leq m$.

THEOREM 12. *There exists an* fpras *for the* (*negation of the*) *mean energy* $-\mathcal{E} = \partial(\ln Z)/\partial\beta$, *where* Z *is the partition function of a ferromagnetic Ising system.*

Proof. We shall assume that $B = 0$, i.e., that there is no external field; the proof in the general case introduces extra technical complications, but requires no new ideas. At the end of the proof, we sketch the modifications required to deal with a nonzero external field.

When $B = 0$ the partition function, Z, simplifies to $Z = A \sum_X w(X)$, where

$$A = \prod_{\{i,j\} \in E} \cosh \beta V_{ij}, \qquad w(X) = \prod_{\{i,j\} \in X} \tanh \beta V_{ij},$$

and the sum is over all closed $X \subseteq E$. (A graph X is said to be *closed* if every vertex of X has even degree.) Define

$$c = \sum_{\{i,j\} \in E} V_{ij} \tanh \beta V_{ij}, \qquad g(X) = \sum_{\{i,j\} \in X} 2V_{ij}/\sinh 2\beta V_{ij},$$

and let $f(X) = c + g(X)$. Then

$$\begin{aligned} \frac{\partial}{\partial\beta} w(X) &= \frac{\partial}{\partial\beta} \prod_{\{i,j\} \in X} \tanh \beta V_{ij} \\ &= \prod_{\{i,j\} \in X} \tanh \beta V_{ij} \sum_{\{i,j\} \in X} 2V_{ij}/\sinh 2\beta V_{ij} \\ &= w(X) g(X). \end{aligned}$$

Using this identity, and starting from the expansion given in Theorem 1:

$$\begin{aligned}-\mathcal{E} = \frac{\partial}{\partial\beta}(\ln Z) &= \frac{\partial}{\partial\beta}\ln A + \frac{\partial}{\partial\beta}\ln Z' \\ &= \frac{\partial}{\partial\beta}\sum_{\{i,j\}\in E}\ln\cosh\beta V_{ij} + \frac{1}{Z'}\frac{\partial}{\partial\beta}\sum_X w(X) \\ &= \sum_{\{i,j\}\in E} V_{ij}\tanh\beta V_{ij} + \frac{1}{Z'}\sum_X w(X)g(X) \\ &= \frac{1}{Z'}\sum_X w(X)(c+g(X)).\end{aligned}$$

Thus, the mean energy can be expressed as an expectation: $-\mathcal{E} = \mathbf{E}(f)$. This expression for $-\mathcal{E}$ immediately suggests that we attempt to estimate the mean energy by taking an average of $f(X)$ over some polynomially bounded set of sample configurations X. This basic idea can be made to work, with a little refinement. As before, we proceed by case analysis.

Case I. $\beta V_{ij} > 1$ for some pair i, j. The condition guarantees that the constant c is not too small in relation to $g(X)$, and hence that the ratio $\max(f)/\mathbf{E}(f)$ is not too large. This, as we have seen, implies that $\mathbf{E}(f)$ can be estimated by direct experiment. First note that the existence of a pair i, j with $\beta V_{ij} > 1$ entails

$$c \geq V_{ij}\tanh\beta V_{ij} > \frac{3V_{ij}}{4} > \frac{3}{4\beta}. \tag{34}$$

Then observe that the inequality $x/\sinh x < 1$, valid for $x > 0$, implies

$$g(X) = \sum_{\{i,j\}\in X} 2V_{ij}/\sinh 2\beta V_{ij} < \sum_{\{i,j\}\in X}\frac{1}{\beta} \leq \frac{m}{\beta}.$$

Thus $\max(f) = c + \max(g) \leq c + m/\beta \leq c(1+4m/3)$, where the final step relies on (34). Since $\mathbf{E}(f)$ is certainly bounded below by c, it follows that $\max(f)/\mathbf{E}(f) \leq 1 + 4m/3 \leq 7m/3$.

Case II. $\beta V_{ij} \leq 1$ for all i, j. To estimate $\mathbf{E}(f)$ within ratio $1+\epsilon$ it is enough, since c is positive, to estimate $\mathbf{E}(g) = \mathbf{E}(f) - c$ within ratio $1+\epsilon$. For simplicity, we shall, in the sequel, work with g instead of f. Using the bounds $1/2 < x/\sinh x < 1$, valid for x in the range $0 < x \leq 2$, we have $1/2\beta < 2V_{ij}/\sinh 2\beta V_{ij} < 1/\beta$, implying

$$1/2\beta < g(X) < m/\beta, \quad \text{for} \quad X \neq \emptyset. \tag{35}$$

Let $C \neq \emptyset$ be a closed subgraph which maximises $w(C)$. Note that C is necessarily a cycle, and can be found in polynomial time using a standard shortest paths algorithm. There are two subcases, depending on the magnitude of $w(C)$.

Case II(a). $w(C) \geq 1/32nm^2$. In this subcase we may estimate $\mathbf{E}(g)$ by direct experiment. Since $Z' \geq w(\emptyset) + w(C) = 1 + w(C)$, it follows that

$$\Pr(X = \emptyset) = \frac{w(\emptyset)}{Z'} = \frac{1}{Z'} \leq (1+w(C))^{-1} \leq 1 - \frac{1}{64nm^2}.$$

Combining this inequality with inequality (35), we obtain $\mathbf{E}(g) > (1/2\beta)\Pr(X \neq \emptyset) \geq 1/128\beta nm^2$. Then a further application of (35) yields the required bound: $\max(g)/\mathbf{E}(g) \leq 128nm^3$.

Case II(b). $w(C) < 1/32nm^2$. The essential problem in this case is that we have no lower bound on the expectation of g. The solution is to increase the expectation artificially by adjusting the weight function w. Naturally, a new weight function induces a new probability distribution on configurations, which in turn alters the expectation of g. It is therefore important to adjust the weights systematically, so that it is possible to recover the expectation of g with respect to the *original* distribution from the knowledge of the expectation of g with respect to the *new* distribution.

The new weight function w_α is parameterised by a real number α in the range $0 \le \alpha \le 1$. We define adjusted edge weights $\lambda_{ij}^{(\alpha)} = \lambda_{ij}^{1-\alpha}$ which induce a new weight function w_α:

$$w_\alpha(X) = \prod_{\{i,j\}\in X} \lambda_{ij}^{(\alpha)} = \prod_{\{i,j\}\in X} \lambda_{ij}^{1-\alpha} = w(X)^{1-\alpha}.$$

Note that the original weight function corresponds to setting $\alpha = 0$, i.e., $w(X) = w_0(X)$. Note also that $0 < \lambda_{ij}^{(\alpha)} \le 1$, so the new edge weights $\lambda_{ij}^{(\alpha)}$ correspond to a well-defined subgraphs-world process; as a consequence, it is possible to sample configurations according to the distribution which assigns probability $w_\alpha(X)/Z'_\alpha$ to configuration X, where $Z'_\alpha = \sum_X w_\alpha(X)$. Let $\mathbf{E}_\alpha(\cdot)$ denote the expectation operator with respect to the new distribution, i.e., $\mathbf{E}_\alpha(h) = (Z'_\alpha)^{-1}\sum_X w_\alpha(X)h(X)$.

Now fix α so that $w_\alpha(C) = 1/32nm^2$; the required value of α is the solution to the equation $w(C)^{1-\alpha} = 1/32nm^2$, and lies in the range $0 < \alpha < 1$. For any $X \ne \emptyset$, maximality of C implies $w(C) \ge w(X)$, which in turn implies $w_\alpha(C) = w(C)^{1-\alpha} \ge w(X)^{1-\alpha} = w_\alpha(X)$; thus C remains a maximum weight nontrivial configuration under the new weight function w_α. Now the quantity we wish to evaluate, namely, $\mathbf{E}(g)$, can be written as an expectation with respect to the new distribution:

$$\begin{aligned}\mathbf{E}(g) &= \frac{1}{Z'}\sum_X w(X)g(X) = \frac{1}{Z'}\sum_X w_\alpha(X)w(X)^\alpha g(X)\\ &= \frac{Z'_\alpha}{Z'}\mathbf{E}_\alpha(w(X)^\alpha g(X)).\end{aligned} \tag{36}$$

Since Z' and Z'_α can be computed by the fpras of §3, we merely have to show that the remaining factor $\mathbf{E}_\alpha(w(X)^\alpha g(X))$ can be approximated in polynomial time. As before, this can be achieved by bounding the ratio of the maximum to the mean.

By maximality of C and inequality (35),

$$\max_X(w(X)^\alpha g(X)) \le w(C)^\alpha \frac{m}{\beta}.$$

Also, by Lemma 11 and (35),

$$\begin{aligned}\mathbf{E}_\alpha(w(X)^\alpha g(X)) &\ge \frac{w_\alpha(C)}{Z'_\alpha}(w(C)^\alpha g(C))\\ &\ge \frac{1}{32nm^3}\frac{w(C)^\alpha}{2\beta} = \frac{w(C)^\alpha}{64\beta nm^3}.\end{aligned}$$

Putting these inequalities together, we obtain

$$\frac{\max(w(X)^\alpha g(X))}{\mathbf{E}_\alpha(w(X)^\alpha g(X))} \le 64nm^4.$$

This completes the analysis of Case II(b).

It is this final case which determines the execution time of the proposed fpras for $-\mathcal{E}$. Consider the cost of evaluating the three factors appearing in (36), within ratio $1+\epsilon/4$ and with failure probability $1/16$. From §3, $O(\epsilon^{-2}m^3n^{11}\log n)$ time suffices to obtain satisfactory estimates for the factors Z' and Z'_α. Setting $\xi = \epsilon/4$ and $\eta = 1/16$ in Lemma 3, we see that $O(\epsilon^{-2}nm^4)$ samples from the generator suffice to evaluate the remaining factor, $\mathbf{E}_\alpha(w(X)^\alpha g(X))$, within ratio $1+\epsilon/4$ and with failure probability $1/16$. By Theorem 10, the generation of each sample requires expected time $O(m^3n^8)$. (As before, we are justified in assuming that $\epsilon \geq 2^{-m}$.) Thus the expected time required to obtain a sufficiently accurate estimate of $\mathbf{E}_\alpha(w(X)^\alpha g(X))$ is $O(\epsilon^{-2}m^7n^9)$; it can be seen that this term dominates those which arise in the other steps of the algorithm. Now set a definite upper bound on the aggregated execution time of the generator which is 16 times the *expected* execution time. By this means the quoted *average case* time complexity is converted into a *worst case* time complexity, at the cost of introducing an additional failure probability of $1/16$. This additional failure probability may be absorbed into the overall failure probability of $1/4$ which the definition of fpras allows.

Finally, we sketch the modifications required to handle the case of nonzero external field, i.e., $B > 0$. Starting with the general subgraphs-world expansion for Z, and differentiating $\ln Z$ with respect to β, the mean energy $-\mathcal{E}$ can again be expressed as the expectation of an appropriately defined random variable $f(X)$. Naturally, the form of f is now more complex. Lemma 11 continues to hold, but with $1/64n^2m^2$ replacing $1/32nm^2$ as the upper bound on $w(X)$. The proof of Lemma 11 increases in technical complexity, but the main idea remains as before. The complications arise from the fact that the maximum weight nontrivial subgraph may be either a cycle (as before) or a single edge (previously excluded). The case analysis in the proof of the theorem proceeds as before, but the upper bound on overall execution time rises to $O(\epsilon^{-2}m^7n^{10})$. □

In this section, we have presented fully polynomial randomised approximation schemes for the first derivatives of $\ln Z$ with respect to β and B. The *second* derivatives of $\ln Z$ also have physical significance: $\mathcal{C} = k\beta^2\partial^2(\ln Z)/\partial\beta^2$ is the *specific heat*, and $\mathcal{X} = \beta^{-1}\partial^2(\ln Z)/\partial B^2$ the *magnetic susceptibility*. (Here, k denotes Boltzmann's constant.) It is natural to ask whether the techniques presented in this section can be extended to these quantities. With a certain amount of computational effort, it is possible to express $\mathcal{C}$ and $\mathcal{X}$ as expectations of appropriate random variables in the subgraphs world. Unfortunately, however, these random variables are not necessarily positive, and the proof techniques of Theorems 9 and 12 are therefore not applicable. At present, the question of the existence of an fpras for $\mathcal{C}$ and $\mathcal{X}$ remains open.

We close the section by presenting proofs of the technical lemmas.

Proof of Lemma 8. We demonstrate, by a simple mapping argument, that

$$\Pr(|\mathrm{odd}(X)| = 2) \geq \mu^2 \Pr(|\mathrm{odd}(X)| = 0). \tag{37}$$

Let Ω_k denote the set $\{X \in \Omega : |\mathrm{odd}(X)| = 2k\}$. Associate with each configuration $X \in \Omega_0$ the set $S(X) = \{X' \in \Omega : |X' \oplus X| = 1\} \subseteq \Omega_1$. It is straightforward to verify that the subsets $\{S(X) : X \in \Omega_0\}$ are pairwise disjoint, and that $\sum_{X' \in S(X)} w(X') \geq w(X)\mu^2$ for all $X \in \Omega_0$. (For $X = \emptyset$ we need the condition $\sum \lambda_{ij} \geq 1$.) Thus $\sum_{X\in\Omega_1} w(X) \geq \mu^2 \sum_{X\in\Omega_0} w(X)$, and (37) follows by dividing through by Z'.

It follows from (37) that $\Pr(|\mathrm{odd}(X)| > 0) \geq \mu^2/(1+\mu^2) \geq \mu^2/2$; this deals with the first part of the lemma. Furthermore, Lemma 4 assures us that $\Pr(|\mathrm{odd}(X)| = 0) \geq 1/10$ whenever $\mu \leq n^{-1}$. Combining this observation with (37) yields the second part of the lemma. □

Proof of Lemma 11. Since $B = 0$, it is only the closed subgraphs $X \subseteq E$ which have nonzero weight: $w(X) = \prod_{\{i,j\}\in X} \lambda_{ij}$. Let $X_0, X_1, X_2, \ldots, X_{s-1}$ be an enumeration of the closed subgraphs of E in order of nonincreasing weight; thus $X_0 = \emptyset$, $w(X_0) = 1$, and $w(X_1) \leq 1/32nm^2$. For each edge $e = \{i, j\} \in E$ define $l(e)$, the *length* of e, to be $-\ln \lambda_{ij}$. Extend the length function to subsets of E by summation, so that $l(X) = -\ln w(X)$. (Clearly, these "lengths" are merely weights which combine additively rather than multiplicatively. Even so, they will prove to be a convenient notational and conceptual aid.)

Define $L = l(X_1) = -\ln w(X_1)$. Let $C = (e_1, e_2, \cdots, e_r)$ be any circuit in E; here, each $e_i = \{v_{i-1}, v_i\}$ is an edge, and $v_r = v_0$. Define $d_0 = 0$, and $d_k = \sum_{i=1}^{k} l(e_i)$ for $1 \leq k \leq r$. Call a directed edge $e_k = (v_{k-1}, v_k)$ of C a *pier* if there exists an integer h with $d_{k-1} < hL/2 \leq d_k$. We make two observations about piers. First, the circuit C is completely determined by the start vertex v_0 and the set of all piers. This is because the total length of edges in C which form a span between two consecutive piers is strictly less than $L/2$. Thus, the existence of two distinct spans between consecutive piers would imply the existence of a circuit of length less than L, and hence of a nontrivial closed subgraph of weight greater than $e^{-L} = w(X_1)$; this would contradict the assumption that X_1 is maximal. Second, the total number of piers in C is at most $2d_r/L = 2l(C)/L$. Intuitively, the role of piers is to provide a compact encoding of circuits.

Now suppose $\alpha \geq 0$, and let $X \subseteq E$ be a general closed subgraph with $l(X) \leq \alpha L$. Decompose X into its connected components; each of these components is Eulerian and hence can be regarded as a circuit with specified start vertex. Encode each component of X as a sequence consisting of the start vertex of the circuit followed by the piers of the circuit in sequence. Encode X itself by concatenating the codes for the various components; note that X is completely determined by the code so formed. Since each connected component of X has length at least L, the total number of vertices in the code (which is equal to the number of components) is at most $l(X)/L = \alpha$. Furthermore, the total number of directed edges in the code (which is the total number of piers) is at most $2l(X)/L = 2\alpha$. These observations yield an upper bound on the number of distinct codes for closed X with $l(X) \leq \alpha L$. The number of ways of selecting a sequence of at most α distinct vertices is bounded by n^α; that of selecting a sequence of at most 2α distinct directed edges by $(2m)^{2\alpha}$; that of interleaving the vertex and edge sequences by $2^{3\alpha}$. Thus the number of distinct codes, and hence the number of closed X with $l(X) \leq \alpha L$, is bounded above by $(32nm^2)^\alpha$.

Now consider the subsequence $X_0, X_1, \ldots, X_{k-1}$, consisting of the k closed subgraphs of greatest weight (i.e., shortest length), and let $\alpha = l(X_{k-1})/L$. Then the coding argument implies $(32nm^2)^\alpha \geq k$. On the other hand, from the definition of α, and using the bound on $w(X_1)$ guaranteed in the statement of the lemma,

$$(32nm^2)^\alpha \leq \frac{1}{w(X_1)^\alpha} = \frac{1}{w(X_{k-1})}.$$

Combining these two inequalities we obtain $w(X_{k-1}) \leq k^{-1}$, and hence

$$Z' = \sum_{k=1}^{s} w(X_{k-1}) \leq \sum_{k=1}^{s} \frac{1}{k} \leq \sum_{k=1}^{2^m} \frac{1}{k}.$$

By a well-known asymptotic result [21, p. 74], the latter sum is bounded by m for all sufficiently large m; indeed it is enough that $m \geq 3$. The lemma holds trivially for $m < 3$. □

6. Completeness results. In this paper we have restricted our attention to the *ferromagnetic* case of the Ising model; moreover we have contented ourselves with solutions which are *approximate* only. The results of this section aim to justify these apparently limited goals. Since we are concerned here with *negative* results, it will be an advantage to work with a simplified version of the Ising problem.

INSTANCE: A symmetric matrix $(V_{ij} : i, j \in [n])$ with entries in $\{-1, 0, +1\}$, and a natural number, β, presented in unary notation.

OUTPUT: The partition function

$$Z = Z(V_{ij}, \beta) = \sum_{\sigma} 2^{-\beta H(\sigma)},$$

where $H(\sigma) = -\sum_{\{i,j\}\in E} V_{ij}\sigma_i\sigma_j$, and the sum is over $\sigma \in \{-1, +1\}^n$. (As usual, E is the set of pairs $\{i, j\}$ with $V_{ij} \neq 0$.)

We refer to the problem in this form as ISING. The main points to note are that the external field is zero, and that powers of e have been replaced by powers of 2. The latter modification merely amounts to a scaling of β, and is made to avoid problems which would arise from the introduction of real arithmetic. The restrictions imposed on the various quantities appearing in an instance of ISING ensure that the output is a rational number whose numerator is a binary integer with a polynomially bounded number of digits, and whose denominator is a certain power of two. The output can thus be considered as a fixed-point binary number with an explicit "binary point." Adopting this viewpoint, it is not difficult to locate ISING within the class #P of combinatorial enumeration problems. (See Garey and Johnson's book [12, p. 167] for an explanation of #P and its completeness class.)

The two problems which form the starting point for the intractability results of this section are MAXCUT:

INSTANCE: An undirected graph G and a positive integer b.

QUESTION: Is there a cut set in G of size b? That is to say, is there a partition of the vertex set of G into two subsets such that the number of edges which span the two subsets is at least b?

and the related #MAXCUT:

INSTANCE: An undirected graph G.

OUTPUT: The number of cut sets in G of maximum size.

The following is a slight extension of a known result.

LEMMA 13. MAXCUT *is* NP-*complete, and* #MAXCUT *is* #P-*complete.*

Proof. NP-completeness of MAXCUT is proved in [13]. The reductions used there are not "parsimonious" [12, p. 169], and hence do not immediately imply #P-completeness of #MAXCUT. As usual, however, the reductions (given in the proofs of Theorems 1.1 and 1.2 of that paper) can be modified, without great difficulty, to yield parsimonious versions. For those who wish to follow the argument in detail, the necessary modifications are presented below.

In [13, Thm. 1.1], new variables $\{e_i : 1 \leq i \leq m\}$ should be introduced, and the definition of the clause set S' amended to read

$$\begin{aligned} S' = \bigcup_{i=1}^{m} \{ &(d_i \vee \neg e_i), \\ &(a_i), (b_i), (c_i), (d_i), (e_i), \\ &(a_i \vee b_i), (a_i \vee c_i), \ldots, (d_i \vee e_i), \\ &(\neg a_i \vee \neg b_i), (\neg a_i \vee \neg c_i), \ldots, (\neg d_i \vee \neg e_i)\}, \end{aligned}$$

where each ellipsis stands for seven omitted disjunctions. Note that there are 26 clauses in S' arising from the ith clause, $(a_i \vee b_i \vee c_i)$, of S. If the ith clause of S is satisfied, then there is precisely one way to choose truth values for the variables d_i and e_i so that 20 of these 26 clauses of S' are satisfied. Conversely, if the ith clause of S is not satisfied then, however d_i

and e_i are chosen, at most 19 of the 26 clauses can be satisfied. Thus, setting $k = 20m$, the original proof goes through. Note that the reduction is now parsimonious.

In [13, Thm. 1.2], duplicate clauses should be removed by replacing each clause $C'_i = (u_i \vee v_i)$ by the seven clauses

$$(u_i \vee \neg c_i), (\neg u_i \vee c_i), (u_i \vee \neg d_i), (\neg u_i \vee d_i), (c_i \vee \neg d_i), (\neg c_i \vee d_i), \text{and } (c_i \vee v_i),$$

where c_i and d_i are new variables. For a given assignment to u_i and v_i, one must set $c_i = d_i = u_i$ in order to maximise the total number of satisfied clauses within these seven. Now, if C'_i is satisfied then all seven clauses may be satisfied; however, if C'_i is not satisfied then at most six of the clauses may be satisfied. The existing proof goes through with k set to $k' + 6q$. Again, the modified reduction is parsimonious. □

The first theorem of the section presents evidence that our restriction to the ferromagnetic case of the Ising model cannot be relaxed.

THEOREM 14. *There can be no* fpras *for* ISING *unless* RP = NP.

RP is the class of decision problems which can be solved in polynomial time by a certain type of randomised algorithm which is allowed one-sided errors. (See [2, p. 138] for a precise definition.) It is widely conjectured that RP is strictly contained in NP. Thus Theorem 14 can be interpreted as strong evidence that no approximation algorithm exists for the Ising partition function in the nonferromagnetic, or "spin-glass" case. Indeed, the existence of such an algorithm would imply, by virtue of Theorem 14, the existence of efficient randomised algorithms for such hard problems as testing satisfiability of a Boolean formula and the Travelling Salesman Problem.

Proof of Theorem 14. Let $G = ([n], E)$ be a graph and b a positive integer defining an instance of MAXCUT. Construct an instance of ISING by setting $\beta = n$, and $V_{ij} = -1$ when $\{i, j\} \in E$. (As usual, $V_{ij} = 0$ when $\{i, j\} \notin E$.) Each spin-vector σ partitions $[n]$ into two subsets, and hence defines a cut set of G: $\text{cut}(\sigma) = \{\{i, j\} \in E : \sigma_i\sigma_j = -1\}$. Note that $H(\sigma) = m - 2|\text{cut}(\sigma)|$, where $m = |E|$. Let N_k denote the number of spin vectors σ for which $|\text{cut}(\sigma)| = k$. Then the simplified partition function can be re-expressed as $Z = \sum_{k=0}^{m} N_k 2^{\beta(2k-m)}$.

Note that if a cut set of size b exists in G then $Z \geq 2^{\beta(2b-m)} = 2^{n(2b-m)}$; in contrast, if no such cut set exists, $Z \leq 2^n 2^{\beta(2b-2-m)} = 2^{-n}2^{n(2b-m)}$. Now the existence of an fpras for ISING would imply that these two cases could be distinguished in polynomial time, with failure probability at most $\frac{1}{4}$; in other words, MAXCUT $\in$ BPP. From this it would follow—since MAXCUT is NP-complete and BPP is closed under polynomial time reductions—that NP $\subseteq$ BPP. However, the inclusion NP $\subseteq$ BPP entails RP = NP [22]. □

Our final theorem states that ISING is a complete problem for the class #P. Thus a polynomial time algorithm which solves it *exactly* would yield similar algorithms for a range of presumably intractable problems, such as counting the number of satisfying assignments of a Boolean formula and counting optimal Travelling Salesman tours. We should therefore not be too disappointed that we have obtained only *approximation* algorithms for the Ising problem.

THEOREM 15. ISING *is #P-complete even when the matrix* V_{ij} *is nonnegative* (*i.e., even in the ferromagnetic case*).

Proof. We present an easy polynomial-time (Turing) reduction from #MAXCUT. Let $G = ([n], E)$ be an instance of #MAXCUT. Set $V_{ij} = +1$ when $\{i, j\} \in E$, and $V_{ij} = 0$ otherwise. Note that $H(\sigma) = 2|\text{cut}(\sigma)| - m$. With N_k as before we have $Z = \sum_{k=0}^{m} N_k 2^{\beta(m-2k)} = 2^{\beta m} p(4^{-\beta})$, where $p(x) = \sum_k N_k x^k$ is a polynomial of degree m. Suppose that the value of p is known at the points $\beta = 0, 1, \ldots, m$, i.e., at $x = 1, 4^{-1}, 4^{-2}, \ldots, 4^{-m}$. Then the coefficients of p can be recovered in polynomial time from these values by interpolation. Using Newton's formula, this process can be carried out using only rational arithmetic; moreover, the lengths of the numerators and denominators all remain polynomially bounded. The leading (nonzero) coefficient of p is twice the number of maximum cut sets in G. (Note that each bipartition of $[n]$ corresponds to a pair of opposite spin-vectors.) $\square$

Further completeness results related to the ones in this section can be found in [3], [15].

REFERENCES

[1] N. ALON, *Eigenvalues and expanders*, Combinatorica, 6 (1986), pp. 83–96.

[2] J. L. BALCÁZAR, J. DIAZ, AND J. GABARRÓ, *Structural Complexity, Volume* I, Springer-Verlag, Berlin, New York 1988.

[3] F. BARAHONA, *On the computational complexity of Ising spin glass models*, J. Physics A, 15 (1982), pp. 3241–3253.

[4] K. BINDER, *Monte Carlo investigations of phase transitions and critical phenomena*, in Phase Transitions and Critical Phenomena, Volume 5b, C. Domb and M. S. Green, eds., Academic Press, London, 1976, pp. 1–105.

[5] J. CHEEGER, *A lower bound for the smallest eigenvalue of the Laplacian*, in Problems in Analysis, R. C. Gunning, ed., Princeton University Press, NJ, 1970, pp. 195–199.

[6] B. CIPRA, *An introduction to the Ising model*, Amer. Math. Monthly, 94 (1987), pp. 937–959.

[7] P. DIACONIS AND D. STROOCK, *Geometric bounds for eigenvalues of Markov chains*, Ann. of Appl. Prob., 1 (1991), pp. 36–61.

[8] J. DODZIUK, *Difference equations, isoperimetric inequality and transience of certain random walks*, Trans. Amer. Math. Soc., 284 (1984), pp. 787– 794.

[9] M. E. DYER, A. M. FRIEZE, AND R. KANNAN, *A random polynomial time algorithm for approximating the volume of convex bodies*, J. ACM, 38 (1991), pp. 1–17.

[10] W. FELLER, *An Introduction to Probability Theory and Its Applications, Volume* I, 3rd ed., John Wiley, NY, 1968.

[11] M. E. FISHER, *On the dimer solution of planar Ising models*, J. Math. Phys., 7 (1966), pp. 1776–1781.

[12] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*completeness*, Freeman, San Francisco, 1979.

[13] M. R. GAREY, D. S. JOHNSON, AND L. STOCKMEYER, *Some simplified* NP-*complete problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.

[14] E. ISING, *Beitrag zur Theorie des Ferromagnetismus*, Z. Phys., 31 (1925), pp. 253–258.

[15] M. R. JERRUM, *Two-dimensional monomer-dimer systems are computationally intractable*, J. of Statist. Phys., 48 (1987), pp. 121–134.

[16] M. R. JERRUM AND A. J. SINCLAIR, *Approximating the permanent*, SIAM J. Comput., 18 (1989), pp. 1149–1178.

[17] M. R. JERRUM, L. G. VALIANT, AND V. V. VAZIRANI, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.

[18] R. M. KARP AND M. LUBY, *Monte-Carlo algorithms for enumeration and reliability problems*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 56–64.

[19] P. W. KASTELEYN, *Dimer statistics and phase transitions*, J. Math. Phys., 4 (1963), pp. 287–293.

[20] S. KIRKPATRICK, C. GELATT, AND M. VECCHI, *Optimisation by simulated annealing*, Science, 220 (May 1983), pp. 671–680.

[21] D. E. KNUTH, *The Art of Computer Programming, Volume* 1*: Fundamental Algorithms*, Addison-Wesley, New York, 1975.

[22] KER-I KO, *Some observations on probabilistic algorithms and* NP-*hard problems*, Inform. Process. Lett., 14 (1982), pp. 39–43.

[23] G. F. LAWLER AND A. D. SOKAL, *Bounds on the L^2 spectrum for Markov chains and Markov processes: A generalization of Cheeger's inequality*, Trans. Amer. Math. Soc., 309 (1988), pp. 557–580.

[24] W. LENZ, *Beitrag zum Verständnis der magnetischen Erscheinungen in festen Körpern*, Z. Phy., 21 (1920), pp. 613–615.

[25] B. MOHAR, *Isoperimetric numbers of graphs*, J. Combinatorial Theory, Ser. B, 47 (1989), pp. 274–291.

[26] G. F. NEWELL AND E. W. MONTROLL, *On the theory of the Ising model of ferromagnetism*, Rev. Modern Phys., 25 (1953), PP. 353– 389.

[27] A. SINCLAIR, *Algorithms for Generation and Counting: A Markov Chain Approach*, Birkhäuser, Boston, 1993.

[28] ———, *Improved bounds for mixing rates of Markov chains and multicommodity flow*, Combin. Probab. Comput. 1(1992), pp. 351–370.

[29] A. J. SINCLAIR AND M. R. JERRUM, *Approximate counting, uniform generation, and rapidly mixing Markov chains*, Inform. Comput., 82 (1989), pp. 93–133.

[30] D. J. A. WELSH, *The computational complexity of some classical problems from statistical physics*, in Disorder in Physical Systems, Oxford University Press, February 1990, pp. 307–321.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1117–1141, December 1993

MINIMAL NFA PROBLEMS ARE HARD*

TAO JIANG† AND B. RAVIKUMAR‡

Abstract. Finite automata (FA's) are of fundamental importance in theory and in applications. The following basic minimization problem is studied: Given a DFA (deterministic FA), find a *minimum* equivalent nondeterministic FA (NFA). This paper shows that the natural decision problem associated with it is PSPACE-complete. More generally, let A $\rightarrow$ B denote the problem of converting a given FA of type A to a minimum FA of type B. This paper also shows that most of these problems are computationally *hard*. Motivated by the question of how much nondeterminism suffices to make the decision problem involving an NFA computationally hard, the authors study the complexity decision problems for FA's and present several intractability results, even for cases in which the input is deterministic or nondeterministic with a very limited nondeterminism. For example, it is shown that it is PSPACE-complete to decide if $L(M_1) \cdot L(M_2) = L(M_3)$, where M_1, M_2, and M_3 are DFAs. These problems are related to some classical problems in automata theory (such as deciding whether an FA has the finite power property), as well as recent ones (such as determining the diversity of a given FA).

Key words. finite automation, minimization, NP-complete, PSPACE-complete

AMS subject classifications. 68Q15, 68Q45

1. Introduction.

Regular languages and their generating devices, the finite-state machines, occupy a central place in theoretical computer science. Among the reasons for their appeal are the elegant and diverse ways in which they can be characterized (e.g., in terms of finite automata, regular expressions, two-way finite automata) and the nice structural properties they possess. Finite automata are easier to design and analyze than are other models of computation. Yet they offer many challenging problems and provide insights into some fundamental concepts in computation theory, such as nondeterminism. In fact, the notion of nondeterminism was first introduced in automata theory by Rabin and Scott in their study of finite automata [Ra59].

Among the central problems in this theory are the complexity of decision problems. In a pioneering work Stockmeyer and Meyer [St73], [Me72] and Hunt, Rosenkrantz, and Szymanski [Hu73], [Hu74], [Hu76] classified the complexity of a number of decision problems for finite automata. Their results indicate that these decision problems (with the exceptions of finiteness and emptiness) are computationally hard[1] when the input is a nondeterministic finite automaton (NFA). Hunt and Rosenkrantz [Hu74] also presented meta-theorems from which hardness results for a wide class of properties of NFA's can be deduced. However, very few hardness results involving deterministic finite automata (DFA's) have been shown. Of course, many decision problems involving DFA's are efficiently solvable. But when they are not efficiently solvable, proving hardness is considerably more difficult. One of the main contributions of the present work is to establish such results.

Another popular area of study dating back to Myhill [My57], Nerode [Ne58], and Rabin and Scott [Ra59] is the problem of minimizing a finite automaton or of converting one type

*Received by the editors January 11, 1991; accepted for publication (in revised form) March 22, 1992. An extended abstract of this work was presented at the 18th International Colloquium on Automata, Languages and Programming, Madrid, Spain, July 1991.

†Department of Computer Science and Systems, McMaster University, Hamilton, Ontario L8S 4K1, Canada. The research of this author was supported in part by a grant from the Science and Engineering Research Board, McMaster University, and by Natural Sciences and Engineering Research Council of Canada operating grant OGP 0046613.

‡Department of Computer Science and Statistics, University of Rhode Island, Kingston, Rhode Island 02881. The research of this author was supported in part by National Science Foundation grant CCR-9111980.

[1] In this paper a problem is computationally hard if it is NP-hard or PSPACE-hard.

of automaton to a minimum-state automaton of another type (which we call the minimum-conversion problem). Some well-known results of this kind are the following: (i) Every NFA with n states can be converted to a DFA of size 2^n [Ra59], and, in general, this bound cannot be improved [Me71]. (Thus the problem of converting an NFA to a minimal equivalent DFA is probably of exponential complexity.) (ii) Every DFA with n states can be converted to a minimal equivalent DFA in $O(n \log n)$ time [Ho71]. This naturally raises the question, How about NFA minimization? It readily follows from the work of Hunt, Rosenkrantz, and Szymanski [Hu76] and of Stockmeyer and Meyer [St73] that NFA minimization is PSPACE-hard for a binary alphabet and NP-hard for a unary alphabet. One problem that is conspicuously absent in all of this study is the problem of converting a DFA to a minimal equivalent NFA. This is one of the main problems studied here. Although the problem's fundamental nature is sufficient to justify its study, we think that it has practical appeal as well. Regular languages are used in many applications, and to optimize the space requirements we would like to store the language in a succinct form. Since an NFA can offer exponential saving in space, it is of interest to find a minimal equivalent NFA for a given DFA. Note further that an NFA is a good representation of a regular language when the principal operation performed is a membership test. A well-known application that uses this idea is presented in [Th68]. It was suggested by a referee of this paper that an additional motivation for finding the minimal NFA equivalent to a DFA comes from computational learning theory [Na91] since in the learning theory setting the inferred machine can be an NFA. (On a related note, it is observed in [Tz89] that an equivalence algorithm for the class of probabilistic finite automata leads to an efficient learning of that class.) Unfortunately, as shown in Theorem 3.2 below, this problem is PSPACE-hard.

Our study was also motivated by a set of problems, some of which are related to the DFA $\rightarrow$ NFA conversion problem stated above:

(i) Minimization of *unambiguous* finite automata (UFA's). A UFA is an NFA in which there is a unique accepting computation for every accepted string. UFA's were first studied by Mandel and Simon [Ma78] and by Reutenauer [Re77]. Stearns and Hunt [St85] showed that equivalence and containment problems are decidable in polynomial time when the inputs are UFA's. It is therefore of interest to know whether there is a polynomial-time algorithm for minimizing a UFA. We show that UFA minimization is NP-complete.

(ii) Complexity of decision problems, such as equivalence, containment, and minimization, when the NFA involved is structurally simple, e.g., reverse deterministic.[2]

Two main contributions of this work (corresponding to §§3 and 4, respectively) can be summarized as follows:

(i) We provide a complete treatment of the complexity of minimal-conversion problems involving various types of finite automata.

(ii) We prove results that substantially strengthen the previous results on the complexity of decision problems related to NFA's.

The conclusion that extremely weak forms of nondeterminism suffice to encode hard problems provides motivation for reorienting the goal towards approximate minimization problems and a careful study of heuristic methods. It should be observed that the only result that establishes the hardness of an approximation problem in automata theory is the recent result of Pitt and Warmuth [Pi89]. Our results also raise some complexity questions in classical automata theory and about some recently developed notions in finite automata. For example, by using the techniques presented in this paper it can be shown that the problem of determining the exact rank of a finite automaton is PSPACE-hard. (See Salomaa [Sa81] for a definition of the rank of a regular language.) The complexity of determining whether the rank is finite remains open. The decidability of this well-known problem is a jewel presented in [Sa81].

[2]An NFA M is reverse deterministic if the reverse of M is deterministic.

This paper consists of four sections, not including this introduction and an appendix. In §2 we introduce the technical terms and the background needed for reading the rest of the paper. In §3 we consider the problem of converting a given DFA to a minimal UFA and to a minimal NFA. We show in Theorems 3.1 and 3.2 that these problems (more precisely, their decision versions) are NP-complete and PSPACE-complete, respectively. In §4 our aim is twofold: in §4.1 we study the equivalence and containment problems involving DFA's augmented by just one nondeterministic operation (such as concatenation) and show that these problems are hard. Note that this claim is stronger than the previously known result that one nondeterministic state in the input automaton is sufficient to make a problem computationally hard. In §4.2 we pursue another variation of the DFA $\rightarrow$ NFA conversion problem in which we require that the resulting NFA be structurally simple. For illustration, a problem of this type may be stated as follows: Inputs are a DFA M_1 and an integer k. The question is, Are there DFA's M_2 and M_3 such that $|M_2| + |M_3| \leq k$ and $L(M_1) = L(M_2) \cdot L(M_3)$? In §5 we present some open problems arising from our work. In the appendix we formally state the decision problems introduced and studied in this paper (as well as those used in our reductions) in a standard format.

2. Preliminaries. We begin by introducing the technical terms and concepts from formal languages, finite automata theory, and computational complexity. Our main goal here is to introduce the necessary technical terms. We assume that the reader is familiar with fundamental notions in the above topics, and we refer the reader to standard references, such as [Ho79] and [Ga78], for unexplained terms.

This work is primarily concerned with two classes of finite-state acceptors—DFA, the deterministic finite automata, and NFA, the nondeterministic finite automata, over a finite alphabet (which is usually implicit).[3] Our definitions of these are standard; we use a 5-tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$. But we will usually provide informal descriptions of machines and avoid formalism whenever possible. The size of a machine M, denoted by $|M|$, is its number of states. The conventional definition of DFA requires a DFA to be *completely specified*, i.e., a move must be defined for each pair of state and symbol. This forces us to include a dead state in most of the DFA's. (A dead state is a state from which no final state can be reached.) One can also allow a DFA to have some unspecified moves. Such a DFA is called a *partially specified* DFA. Although all of our results hold for both completely and partially specified DFA's, to simplify the presentation we will consider only partially specified DFA's in this paper. It is easy to see that all the proofs (possibly with some minor modifications) also work for completely specified DFA's. For partially specified DFA's we can further assume without loss of generality that each DFA considered consists of live states only. (A state is called live if it is not dead.) For convenience, an NFA is assumed to be ϵ-free except in the proof of Theorem 3.2, where we use ϵ-moves. (Note that the requirement that it be ϵ-free will not increase the size of a minimal NFA.) We also study a less familiar class of finite automata—UFA, the collection of unambiguous NFA's [Ma78], [Re77], [St85]. In §3 we briefly describe their significance.

We use the operations on languages in the usual sense. The standard ones are union, intersection, concatenation, Kleene star, and reversal, denoted respectively by $\cup$, $\cap$, $\cdot$, *, and rev. As is customary, we drop the operator $\cdot$ except where its absence could cause discomfort in reading the expression. We also use the quotient operation (/) defined as follows: Let L be a language and x a string. Then L/x is defined as

$$L/x = \{y | y \cdot x \in L\}.$$

[3]DFA, NFA, etc., denote the collection of machines and also an individual machine. The context will easily resolve this ambiguity.

It is well known [Ho79] that regular languages are closed under all these operations.

A DFA (or NFA or UFA) M is said to be minimum or minimal (we use these two terms interchangeably) if there is no DFA with fewer than $|M|$ states accepting $L(M)$.

The basic complexity theory notions used in this work are Turing machine (TM), NP-completeness, PSPACE-completeness and polynomial-time many–one reductions, instantaneous descriptions (ID's), halting TM, acceptance, and time and space complexities. We refer the readers to [Ho79] for definitions of these terms. Throughout this paper a *reduction* means a polynomial-time many–one reduction. We use the word *hard* in the technical sense that the problem is NP-hard, PSPACE-hard, or intractable in some other sense. The problems studied in this paper and the known hardness results used in the reductions are all listed in the appendix in a format popularized by Garey and Johnson [Ga78].

3. Complexity of finding minimal NFA. In this section we study the minimal-conversion problems. Let A and B be two classes of finite-state acceptors. For example, A may be the collection of NFA, and B may be the collection of DFA. $A \to B$ denotes the problem of converting a type-A finite automaton to a minimal type-B finite automaton. It is more convenient to state this as a decision problem: The input will be M, a type-A machine, and an integer k (in binary). The decision question we want to answer is, Is there a k-state type-B machine accepting $L(M)$? We will be primarily interested in the decision versions of the minimal-conversion problems. Since all the results we present are (conditional) lower-bound results (NP-completeness or PSPACE-completeness), they imply the same lower bound for the optimization versions. Our study involves three important classes of finite-state acceptors, namely, DFA, NFA, and UFA.

The two problems DFA $\to$ DFA and NFA $\to$ DFA are classical. Hopcroft's well-known algorithm [Ho71] for the former problem runs in $O(n \log n)$ time. The latter problem is known to have an exponential lower bound (on space and time), as is seen from the simple fact that the minimal equivalent DFA of an n-state NFA can have 2^n states [Me71]. The decision version of the latter problem is also hard; it can be shown to be PSPACE-hard by using the fact that the universe problem for NFA is PSPACE-hard [Me72]. In fact, the problem is NP-hard even over a one-letter alphabet [St73]. Note that these claims hold even if we fix $k = 1$. The problem NFA $\to$ UFA also requires exponential space (and thus time) since there is an unavoidable exponential blowup when NFA is converted to UFA [St85]. The exponential blowup holds even in the case of a unary alphabet [Ra89]. The decision version of this problem is also PSPACE-hard since the argument for NFA $\to$ DFA holds in this case as well.

This leaves us with four basic problems: DFA $\to$ NFA, DFA $\to$ UFA, UFA $\to$ UFA, and UFA $\to$ DFA. We will study the decision versions of these problems, and we will show that the first problem is PSPACE-complete and that the second and third are NP-complete. These results hold for any alphabet of size at least 2. The unary case is studied in a companion work [Ji90] (see §5 for some details regarding [Ji90]). Since UFA is exponentially more succinct than DFA [St85], the fourth problem requires exponential space and time. Unfortunately, we do not know the complexity of the decision version of this problem at the moment. (In fact, we do not know whether it is in NP or PSPACE-complete.)

From now on $A \to B$ denotes the decision version of the problem of converting a type-A finite automaton to a minimal type-B finite automaton. We need the following lemmas in the proofs of Theorems 3.1 and 3.2, where the claimed *hardness* results are proved. Our first two lemmas are from [St85]. They are needed to prove that DFA $\to$ UFA is in NP. For the sake of completeness we will present a sketch of proof of these lemmas. In fact, our proof of Lemma 3.1 is simpler than the original proof.

LEMMA 3.1 (Stearns and Hunt [St85]). *There is a deterministic polynomial-time algorithm for deciding whether the two given* UFA, *M_1 and M_2, are equivalent.*

Proof (sketch). Since a polynomial-time algorithm for containment immediately implies a polynomial-time algorithm for the equivalence problem, we concentrate on the former problem: Given M_1 and M_2, determine whether $L(M_1) \subseteq L(M_2)$. We show that the containment problem reduces to the following proper-containment problem stated as follows: Given as input two UFA's M_3, M_4 such that $L(M_3) \subseteq L(M_4)$, determine whether the containment is proper. (Reduction: Given M_1 and M_2, design a UFA M_3 for $L(M_1) \cap L(M_2)$. Note that M_3 can be obtained from M_1 and M_2 by using the standard construction [Ho79] in polynomial time. Choose $M_4 = M_1$. The reduction holds because the latter problem has a "yes" answer if and only if the former has a "no" answer.) The ideas described so far are directly from [St85]. Our simplification comes in the polynomial-time algorithm for the proper containment. Stearns and Hunt developed this algorithm by using difference equations. We use the adjacency matrix representation A and B of M_3 and M_4, respectively. It can be observed that the proper-containment problem has a "yes" answer if and only if the number of strings in $L(M_3)$ of length i is exactly equal to the number of strings of length i for each $i = 1, 2, \ldots, n_1 + n_2$, where n_1 and n_2 are the number of states in M_3 and M_4. The proof now follows from the fact that the number of strings of length i can be readily obtained from the matrix A^i. A^i can be computed fast by using the standard repeated-squaring technique. $\square$

LEMMA 3.2 (Stearns and Hunt [St85]). *There is a deterministic polynomial-time algorithm that, given an* NFA *M as input, decides whether M is unambiguous.*

Proof (sketch). Let L' be the set of strings that can be derived by at least two distinct accepting paths in M. It is easy to design an NFA M' to accept L' (from M) in polynomial time. The claim follows from the fact that M is unambiguous if and only if $L(M')$ is empty and that emptiness of an NFA can be tested in polynomial time. $\square$

In the proof of Theorem 3.1 we will use a variation of the set basis problem, which has been shown to be NP-complete by Stockmeyer [St76]. We call our problem the *normal set basis problem.* The problem is stated below.

Let C and B be collections of finite sets. B is said to be a *basis* of C if for each $c \in C$ there is a subcollection of B whose union is exactly c. B is said to be a *normal basis* of C if for each $c \in C$ there is a pairwise disjoint subcollection of B whose union is exactly c. The (normal) set basis problem is as follows: Given a collection C of finite sets and positive integer $k \leq |C|$, decide whether C has a (normal) basis with cardinality at most k.

Since it is not obvious how the original proof of Stockmeyer in [St76] can be modified to show the NP-completeness of the normal set basis problem, here we include an independent proof of this result. Note that our proof will also show the NP-completeness of the set basis problem.

LEMMA 3.3. *The normal set basis problem is* NP-*complete.*

Proof. Clearly, the normal set basis is in NP. We show that it is NP-hard by transforming the Vertex Cover problem to it. Let $G = \langle V, E \rangle$, and k be an instance of the Vertex Cover Problem, where $V = \{v_1, v_2, \ldots, v_n\}$. We design the collection C of sets as follows.

For each vertex v_i, let $c_i = \{x_i, y_i\}$, $i = 1, 2, \ldots, n$. Let $\langle v_i, v_j \rangle$ be in E with $i < j$. We define

$$\begin{aligned}
c_{i,j}^1 &= \{x_i, a_{i,j}, b_{i,j}\},\\
c_{i,j}^2 &= \{y_j, b_{i,j}, d_{i,j}\},\\
c_{i,j}^3 &= \{y_i, d_{i,j}, e_{i,j}\},\\
c_{i,j}^4 &= \{x_j, e_{i,j}, a_{i,j}\},\\
c_{i,j}^5 &= \{a_{i,j}, b_{i,j}, d_{i,j}, e_{i,j}\}.
\end{aligned}$$

Finally, we let $C = \{c_i | 1 \leq i \leq n\} \cup \{c^t_{i,j} | \langle v_i, v_j \rangle \in E, 1 \leq t \leq 5\}$, and we let $s = n + 4|E| + k$. Obviously, C and s can be constructed from G and k in polynomial time. We complete the proof by showing that G has a vertex cover of size at most k if and only if C has a normal basis of cardinality at most s.

Intuitively, the idea behind the proof is as follows: To cover the set c_i, the basis B must contain either c_i or both singleton sets $\{x_i\}$ and $\{y_i\}$. Let $V_1 = \{v_i |$ both $\{x_i\}$ and $\{y_i\}$ are in $B\}$. We can show for a fixed $\langle v_i, v_j \rangle \in E$ that at least four sets (in addition to sets $c_i, c_j, \{x_i\}$, and $\{x_j\}$) are necessary to cover the five sets $c^t_{i,j}$, $t = 1, \ldots, 5$, and that four sets (in addition to sets $c_i, c_j, \{x_i\}$, and $\{x_j\}$) are sufficient if and only if at least one of v_i or v_j is in V_1. Thus if there is a vertex cover of size k, we choose the normal basis B as follows: For every v_i in the cover we include both $\{x_i\}$ and $\{y_i\}$ in B; otherwise, we include c_i in B. Conversely, if there is a normal basis of cardinality s, we will show that V_1 (defined above) can be extended to a vertex cover of size k.

Now we give the formal details. Let G have a vertex cover V_1 of size k. We will show that there is a normal basis B of size s. Define a collection of sets B as follows: For $v_i \in V_1$ include both $\{x_i\}$ and $\{y_i\}$ in B; else include $\{c_i\}$ in B. The number of sets included in B so far is $k + n$. Let $e = \langle v_i, v_j \rangle$ (where $i < j$) be an arbitrary edge in G. Since V_1 is a vertex cover, either v_i or v_j (or both) is in V_1. Assume, e.g., that v_i is in V_1. Include the sets $\{a_{i,j}, b_{i,j}\}$, $\{y_j, b_{i,j}, d_{i,j}\}$, $\{d_{i,j}, e_{i,j}\}$, and $\{x_j, d_{i,j}, e_{i,j}\}$ in B. (We omit the similar case corresponding to $v_j \in B$.) This completes the definition of B. Note that $c^t_{i,j}$ can be expressed as a union of members of B as

$$\begin{aligned} c^1_{i,j} &= \{x_i\} \cup \{a_{i,j}, b_{i,j}\}, \\ c^3_{i,j} &= \{y_i\} \cup \{d_{i,j}, e_{i,j}\}, \\ c^5_{i,j} &= \{a_{i,j}, b_{i,j}\} \cup \{d_{i,j}, e_{i,j}\} \end{aligned}$$

and that the other two sets are members of B. Since the total number of sets included in B for each edge is four, the cardinality of B is $(k + n) + 4|E| = s$. From the foregoing argument it is also obvious that B is a normal basis of C.

Conversely, suppose that there is a normal basis B of cardinality at most $s = n + 4|E| + k$. We can assume without loss of generality that no proper subcollection of B is a normal basis. We show that G has a vertex cover of size at most k. Define V' $\{v_i |$ both $\{x_i\}$ and $\{y_i\}$ are in $B\}$. Let $|V'| = k'$. The number of sets in B consisting of only x_i and/or y_i is at least $n + k'$. (This can be seen from the fact that B must have the subset c_i for all i such that $v_i \notin V'$. Thus there are $n - k'$ such sets, in addition to $2k'$ singleton sets corresponding to i's such that $v_i \in V'$.) Let $E' \subseteq E$ be the set of edges covered by V', i.e., $E' = \{\langle v_i, v_j \rangle | v_i$ or v_j is in $V'\}$. The following observation can be easily shown:

Observation 3.1. For any $e \in E$ at least four sets of B (in addition to sets $c_i, c_j, \{x_i\}$, and $\{x_j\}$) are required to cover the sets $c^t_{i,j}$, $t = 1, \ldots, 5$. Further, at least five sets in addition to sets $c_i, c_j, \{x_i\}$, and $\{x_j\}$ are required to cover them if $e \notin E'$.

Now the total number of sets needed to cover C is at least

$$n + k' + 4|E'| + 5(|E| - |E'|) = n + k + 4|E| + (|E| - |E'| + k' - k).$$

Thus $|E| - |E'| \leq k - k'$. Thus $|E| - |E'| + k' \leq k$. We conclude the proof by showing that there is a vertex cover V of size $|E| - |E'| + k'$: Add one of the end vertices of each edge $e \in E - E'$ to V'. This vertex cover is of size $= |E| - |E'| + k' \leq k$. □

We now present the main results of this section.

THEOREM 3.1. *The* DFA $\rightarrow$ UFA *problem is* NP-*complete.*

Proof. It is easy to show, by using Lemmas 3.1 and 3.2, that the problem is in NP. Let M and k be inputs. The nondeterministic algorithm will guess an NFA M' with at most k states.

It will then check, by using Lemma 3.2, that M' is unambiguous. Then, by using Lemma 3.1, it verifies that they are equal and accepts.

To prove NP-hardness we reduce the Normal Set Basis problem to our problem. Let C and s be an instance of the Normal Set Basis problem, where $C = \{c_1, c_2, \ldots, c_n\}$ and $c_i = \{a^i_1, a^i_2, \ldots, a^i_{n_i}\}$. Construct a DFA M as follows: M is defined over an alphabet $\Sigma = \{t \mid t \in c_i$ for some $i\} \cup \{b_i \mid i = 1, 2, \ldots, n\}$. The state set of M is $Q = \{q_0, q_1, \ldots, q_n, q_f\}$. The start and accepting states are q_0 and q_f, respectively. The transition function $\delta : Q \times \Sigma \to Q$ is defined as

$$\begin{aligned} \delta(q_0, b_i) &= q_i, \qquad 1 \le i \le n,\\ \delta(q_i, a^i_j) &= q_f, \qquad 1 \le i \le n, \quad 1 \le j \le n_i. \end{aligned}$$

Let $k = s + 2$. We claim that C has a normal basis of cardinality s if and only if $L(M)$ is accepted by a k-state UFA. Suppose C has a normal basis of cardinality at most s. Let $r_1, r_2, \ldots, r_s$ be a normal basis of C. The following is a description of a UFA M' with at most k states accepting $L(M)$: Let $Q' = \{q_0, q_f, s_1, \ldots, s_s\}$ be the states of M' where s_j corresponds to the basic set r_j. To describe the transition function δ' we first fix (arbitrarily) a representation of each c_i as a disjoint union of the basic sets. Say that each basic member in this representation belongs to c_i. Now we describe δ':

$$\begin{aligned} s_j \in \delta'(q_0, b_i) &\quad \text{iff } r_j \text{ belongs to } c_i,\\ \delta'(s_j, a) = q_f &\quad \text{iff } a \in r_j. \end{aligned}$$

It is easy to see that M' is a UFA and that it accepts $L(M)$.

To show the converse assume that $L(M)$ can be accepted by a UFA M' with k or fewer states. Assume M' is a minimal UFA. Let q'_0 be the starting of M'. Since $L(M)$ contains a finite set of strings of length 2, the states $Q - \{q'_0\}$ can be partitioned as Q_1 and Q_2 such that the only transitions in M' are from q'_0 to states in Q_1 and from the states of Q_1 to those in Q_2. From the minimality of M' it follows that $|Q_2| = 1$ (if it is assumed that the instance is nontrivial). For each state $q \in Q_1$ define a set $B_q = \{a \mid q_f \in \delta(q, a)\}$. It is easy to see that the collection $\{B_q\}$, $q \in Q_1$, is a normal basis of C of size at most $k - 2$. □

The next corollary easily follows from Lemma 3.1 and Theorem 3.1.

COROLLARY 3.1. *The* UFA $\to$ UFA *problem is* NP-*complete.*

We next consider the DFA $\to$ NFA problem and show that it is PSPACE-complete. We prove this result by reducing the Universe Problem for Multiple DFA to the problem DFA $\to$ NFA. The Universe Problem for Multiple DFA is the following: Given a collection of DFA $M_1, \ldots, M_n$ over a finite alphabet Σ, decide whether $\bigcup_i L(M_i) = \Sigma^*$. This problem is readily seen to be PSPACE-complete by reduction from the Finite-State Automata Intersection Problem, which was shown by Kozen [Ko77] to be PSPACE-complete. (All we need to do is to complement the languages $L(M_i)$ by interchanging the accepting and nonaccepting states.)

THEOREM 3.2. *The* DFA $\to$ NFA *problem is* PSPACE-*complete.*

Proof. This problem is in PSPACE since the equivalence problem for NFA's is in PSPACE. We show that it is PSPACE-hard by reducing the universe problem for multiple DFA to our problem. Let DFA $M_1, \ldots, M_n$ (over an alphabet Σ) form an instance of the universe problem. (The answer to this instance is "yes" if $\bigcup_i L(M_i) = \Sigma^*$ and is "no" otherwise.) We want to transform this to a DFA M and an integer k such that the answer to the universe problem is "yes" if and only if some NFA with k or fewer states can accept $L = L(M)$. Intuitively, the idea is as follows. We can assume that M_i are minimal. The alphabet over which L is defined includes, in addition to the symbols of Σ, additional symbols $b_{i,j}$, corresponding to state $q_{i,j}$ of M_i, and a_i, corresponding to the start state of M_i. M is chosen so as to accept the marked versions of the strings accepted by M_i's, where the marks correspond to states of M_i's. This ensures that any NFA accepting L must essentially possess all the states of M_i's. In addition, L includes Σ^* and some additional strings used as enforcers. The enforcers will be chosen in such a way that an NFA N accepting L will require at least k states and at most $k+1$ states, where $k = 4 + \sum_i |M_i|$. Since all states of M_i's are required to be present in N, if $\bigcup_i L(M_i) = \Sigma^*$, then with no additional state, N can accept all strings in Σ^*. The enforcers will make sure that neither the states of M_i's nor the states needed to accept the enforcer strings can be used for accepting Σ^*. Thus at least one more state is needed to accept L.

We now present a formal proof. We assume with no loss of generality the following properties of M_i: (i) M_i's are minimal, (ii) for each i $L(M_i)$ is neither empty nor equals Σ^*, and (iii) $\Sigma \cup \{\epsilon\}$ is contained in $\bigcup_i L(M_i)$. Let t_i be the number of states in M_i, let $Q_i = \{q_{i,1}, \ldots, q_{i,t_i}\}$ be the state set of M_i, let $q_{i,1}$ be the start state, and let $F_i \subseteq Q_i$ be the accepting states of M_i. For a state $q_{i,j}$ let $P(i, j)$ be defined as the set of strings accepted by M_i if F_i were redefined to be the singleton $\{q_{i,j}\}$. More formally,

$$P(i, j) = \{w | \delta(q_{i,1}, w) = q_{i,j}\}.$$

Clearly, $P(i, j) \cap P(i, k) = \emptyset$ for $j \neq k$.

For each i, $1 \leq i \leq n$, let a_i be a new symbol, and for each i, j, $1 \leq i \leq n$, $1 \leq j \leq t_i$, let $b_{i,j}$ be a new symbol. Let $B = \{b_{i,j} | 1 \leq i \leq n,\ 1 \leq j \leq t_i\}$. Let c, d, and f be three additional new symbols. For each i define a language $P(i)$ as

$$P(i) = a_i \bigcup_{j=1}^{t_i} [P(i, j) b_{i,j}].$$

Also define

$$Q(i) = \{b_{i,1}\} \cup \{x b_{i,j} | x \in \Sigma, \delta(q_{i,1}, x) = q_{i,j}\}$$

and

$$R = (\{c\} \cup \Sigma)(\{d\} \cup \Sigma)\Sigma^*(\{f\} \cup B).$$

Finally, let

$$L = \bigcup_i [P(i) \cup a_i L(M_i) \cup Q(i)] \cup \Sigma^* \cup R,$$

and let $k = \Sigma_i t_i + 4$. It is easy to show that a DFA accepting L can be constructed in polynomial time, given M_i. To conclude the proof of Theorem 3.2 it suffices to prove Claim 3.2 below which asserts that the reduction is correct. Before we prove this, we need some observations concerning the languages defined above. In particular, we will establish a lower bound of k and upper bound of $k+1$ on the size of a minimal NFA accepting L (irrespective of whether the answer to the original instance is "yes" or "no"). With this intention we describe (informally) an NFA N' with $k+1$ states accepting L. N' is obtained by taking the machines M_i's and adding five more states q_0, p_1, p_2, p_3, and f. The start state is q_0, and the accepting states of N' are the accepting states of M_i's along with p_1 and f. The transitions include all the original transitions of M_i's, with the following additions: $q_{i,1} \in \delta'(q_0, \epsilon)$, and $q_{i,1} \in \delta'(q_0, a_i)$, $\delta'(q_0, a) \in f$ for all $a \in \Sigma$, $p_1 \in \delta'(q_{i,j}, b_{i,j})$, $p_2 \in \delta'(q_0, c)$, $p_2 \in \delta'(q_0, a)$ for all $a \in \Sigma$, $p_3 \in \delta'(p_2, d)$, $p_3 \in \delta'(p_2, a)$ for all $a \in \Sigma$, $p_3 \in \delta'(p_3, a)$ for all $a \in \Sigma$, and, finally, $p_1 \in \delta'(p_3, f)$, $p_1 \in \delta'(p_3, b)$ for all $b \in B$. Figure 3.1 presents this NFA. Please note that

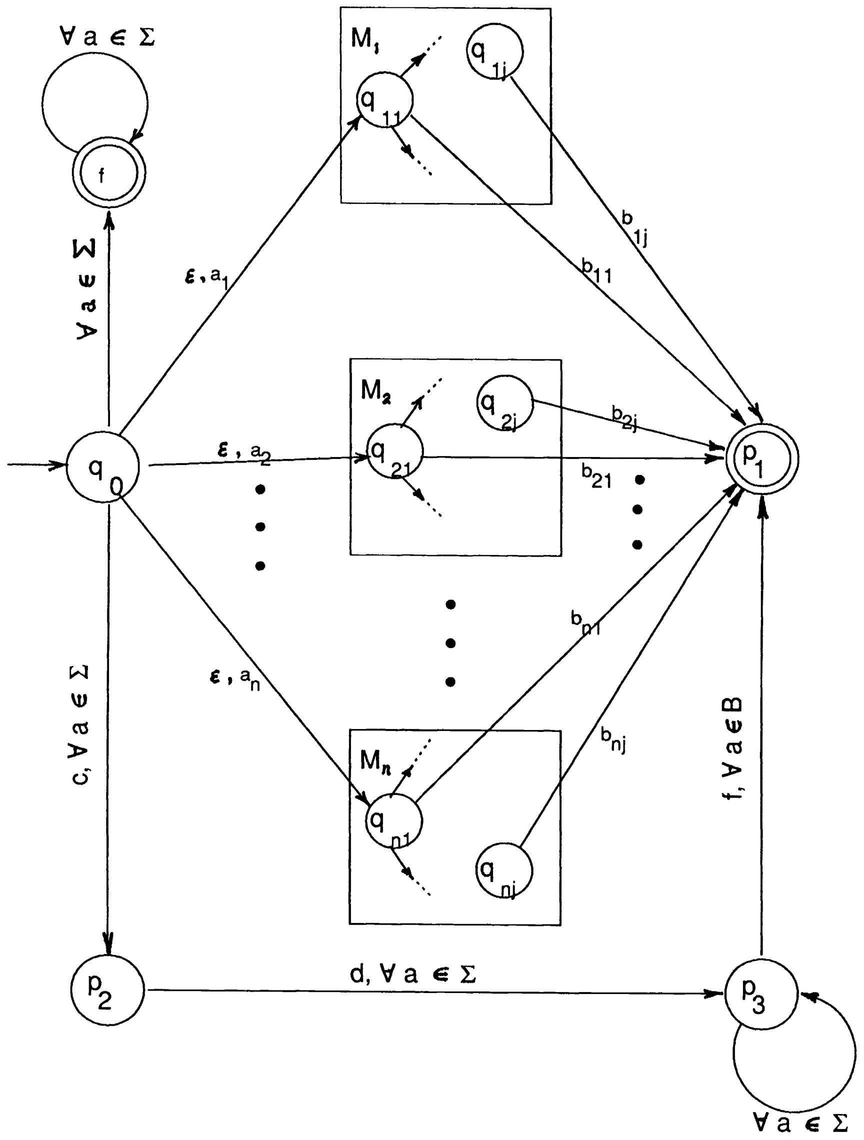


FIG. 1. NFA *in the proof of Theorem* 3.2. (*N.B. M_i's are given* DFA'*s, with all the arcs and only the additional edges connecting M_i's to the other states are shown.*)

this figure omits most of the details of the internal structure of M_i's and concentrates only on the external connections, connecting M_i's to the remaining five states.

Thus if N is a minimal NFA accepting L, $|N| \leq k + 1$. Let Q_N be the set of states of N, let F_N be the accepting states of N, and let q_0 be the start state. Define $S(i, j)$ as the set of states p reachable following a string in $a_i P(i, j)$ such that some final state is reachable from p via an arc labeled $b_{i,j}$. More succinctly,

$$S(i, j) = \{q \in Q_N | \delta(q_0, w) = q \text{ for some } w \in a_i P(i, j) \text{ and } \delta(q, b_{i,j}) \cap F_N \neq \emptyset\}.$$

Clearly, $S(i, j)$ is nonempty for all pairs (i, j). We now prove the following claim:

CLAIM 3.1. *For $(i, j) \neq (i', j')$, $S(i, j) \cap S(i', j') = \emptyset$.*

Proof. Suppose not. Then there is a $u \in S(i, j) \cap S(i', j')$. We consider two cases.

Case 1: $i \neq i'$. Clearly, there exist strings x and y such that $u \in \delta(q_0, a_i x)$, $u \in \delta(q_0, a_{i'} y)$, and $\delta(u, b_{i,j}) \cap F \neq \emptyset$. This implies that $a_{i'} y b_{i,j} \in L$, a contradiction.

Case 2: $i = i'$. Clearly, $j \neq j'$. Let w be a string in $P(i, j)$ such that $u \in \delta(q_0, a_i w)$. Note that $w \notin P(i, j')$ since $P(i, j) \cap P(i, j') = \emptyset$ for $j \neq j'$. It follows that $a_i w b_{i,j'} \in L$, a contradiction. This concludes the proof of Claim 3.1. □

From Claim 3.1 it follows that N must have at least $\sum_{i=1}^{n} t_i = k - 4$ states. Also, in the above proof our argument in Case 1 implies that each state in $S(i, j)$ is not reachable from any state in $S(i', j')$ if $i \neq i'$. Our next step is to show that N needs four more states in addition to the $k - 4$ established above. The need for these additional four states is established in (i)–(iv) below:

(i) Clearly, the start state q_0 is not in any $S(i, j)$.

(ii) Consider the set of states Q_c reachable from q_0 on c. Clearly, Q_c is not empty and is different from each state in $S(i, j)$.

(iii) The same argument is used for Q_{cd} (the set of states reachable following cd from q_0). Note also that $Q_{cd} - Q_c \neq \emptyset$.

(iv) Consider the set of states with an incoming arc labeled $b_{i,j}$ for some (i, j). This set is nonempty and cannot overlap any of the states described above. Denote this set by Q_f.

Summarizing what we have shown so far, a minimal NFA accepting L will have at least k states and at most $k + 1$ states.

CLAIM 3.2. $\bigcup L(M_i) = \Sigma^*$ *if and only if L can be accepted by an* NFA *with at most k states.*

Proof. (only if) We exhibit a k-state NFA accepting L: Simply remove the state f (and all the transitions associated with it) from the NFA of Fig. 3.1. The resulting NFA is easily seen to accept L.

(if) Suppose that a minimal NFA N accepting L has k states. Define $S(i, j)$, q_0, Q_c, Q_{cd}, and Q_f as defined above for this N. It is easily seen that the state set of N is $\bigcup_{i,j} S(i, j) \cup \{q_0\} \cup Q_c \cup Q_{cd} \cup Q_f$. Also, $|S(i, j)| = 1$ for all i and j, $|Q_c| = 1$, $|Q_{cd}| = 1$, and $|Q_f| = 1$. This is true because our lower bound of k in Claim 3.1 counted only one for each $S(i, j)$ and each of Q_c, Q_{cd}, and Q_f. By the definition of R, Q_c and Q_{cd} cannot lead to a final state following any string in Σ^*. We also know from the argument presented above that a string in Σ^* can be accepted only by using some states in $\bigcup_{i,j} S(i, j) \cup \{q_0\} \cup Q_f$.

The definitions of $Q(i)$'s and R restrict the states in $\bigcup_{i,j} S(i, j)$ that N can move to in the first step on a symbol in $\Sigma \cup \{\epsilon\}$. Clearly, in the first step N can only move to $S(i, 1)$ for some i on ϵ. On a symbol $x \in \Sigma$, N can only move to some $S(i, j)$, where $q_{i,j} = \delta(q_{i,1}, x)$.

Let L_{0f} be the set of strings in Σ^* accepted by using states q_0 and Q_f only. Since N cannot move to q_0 (Q_f) from q_0 (Q_f, respectively) on any symbol,

$$L_{0f} \subseteq \Sigma \cup \{\epsilon\}.$$

For each i let L_i be the set of strings in Σ^* accepted by using states in $\{q_0\} \cup \bigcup_j S(i, j)$. For each $S(i, j)$ let $L(i, j)$ be the subset of Σ^* accepted if $S(i, j)$ is considered the start state. Then for each i

$$L_i = \bigcup \{x L(i, j) | x \in \Sigma \cup \{\epsilon\}, N \text{ can move to } S(i, j) \text{ on symbol } x\}.$$

The definition of $S(i, j)$ means that if N can move to $S(i, j)$ from q_0 on $a_i x$, then $x \in P(i, j)$. Thus

$$a_i L(M_i) = a_i \bigcup_j P(i,j)L(i,j)$$
$$\supseteq a_i L_i.$$

Thus $L_i \subseteq L(M_i)$. Hence

$$L(N) \cap \Sigma^* = \bigcup_i L_i \cup L_{0f}$$
$$\subseteq \bigcup_i L(M_i) \cup L_{0f}.$$

Since $\Sigma^* \subseteq L = L(N)$ and $L_{0f} \subseteq \Sigma \cup \{\epsilon\} \subseteq \bigcup_i L(M_i)$,

$$L(N) \cap \Sigma^* = \Sigma^*$$
$$\subseteq \bigcup_i L(M_i) \cup L_{0f}$$
$$\subseteq \bigcup_i L(M_i).$$

That is, $\bigcup L(M_i) = \Sigma^*$. This concludes the proof of Claim 3.2 and Theorem 3.2. □

It is interesting to observe that the results of Theorems 3.1 and 3.2 and Corollary 3.1 hold even when restricted to a binary alphabet. We sketch a proof of this result:

COROLLARY 3.2. *The results of Theorems* 3.1 *and* 3.2 *and Corollary* 3.1 *hold even when the input* DFA *is defined over* $\Sigma = \{0, 1\}$.

Proof (sketch). The essential idea is to use a binary encoding in the above proofs. This is very easy to do in Theorem 3.1 and Corollary 3.1. So we consider Theorem 3.2. Let the alphabet over which L of Theorem 3.2 is defined be Σ' and $m = |\Sigma'|$. For convenience, relabel the symbols of Σ' as $\{a_1, \dots, a_m\}$. We encode a_i as $0^i 1$. Formally, let h be the homomorphism $h(a_i) = 0^i 1$. To complete the reduction we need to introduce some additional enforcers. The following motivates the need for the enforcers. Consider a minimum NFA N accepting L, and let $q_i \to q_j$ be a transition on symbol a_k in N. This transition can be realized in N', the corresponding minimum NFA over the binary alphabet, by a chain of states $q_i, p_{i,1}, p_{i,2}, \dots, p_{i_k}, q_j$ as

$$q_i \xrightarrow{0} p_{i,1} \xrightarrow{0} p_{i,2} \xrightarrow{0} \cdots \xrightarrow{0} p_{i,k} \xrightarrow{1} q_j.$$

We want the new states $p_{i,1}, p_{i,2}, \dots, p_{i,k}$ to be private states of q_i, i.e., they should not be shared by any other states of N. This can be achieved by introducing additional strings in L'. For each i let $P(i)$ denote the language accepted N if q_i were considered to be the only accepting state. Define

$$L' = h(L) \cup_i (h(P(i))0^m 1^{i+1}).$$

Then each state q_i is forced to have m private states $p_{i,1}, p_{i,m}, \dots, p_{i,m}$ in N'. The following claim, which we state without proof, completes the proof of the corollary:

CLAIM 3.3. $\bigcup_i h(L(M_i)) = h(\Sigma^*)$ *if and only if* L' *can be accepted by an* NFA *with at most* $(m + 1)(|N| + 1)$ *states.* □

4. Complexity of finding minimal NFA's of restricted types. Much effort has been spent in classifying the complexity of decision problems for finite-state machines. The results of Stockmeyer and Meyer [St73], Hunt, Rosenkrantz, and Szymanski [Hu76], and others indicate that the complexity of most of the fundamental decision problems (except membership, emptiness, and finiteness) involving NFA's are hard. A way to understand the nature of nondeterminism is to make the NFA as simple as possible and determine whether the decision problems are still hard. Two well-known results of this kind are that of Stockmeyer and Meyer [St73], who showed that the universe problem is coNP-complete even for a unary alphabet, and that of Hunt, Rosenkrantz, and Szymanski [Hu76], who showed that the equivalence problem is coNP-complete even for NFA's with no loops. The former result, combined with the existence of the normal form [Ch86] for unary NFA's (see §1), implies that the universe problem is hard even for NFA's with only *one* nondeterministic state. We will present several interesting hardness results on equivalence, containment, etc., in which the NFA involved will be structurally very simple (such as the concatenation of two DFA's).

Another collection of problems studied in this section is in keeping with the central theme of this paper, namely, the complexity of finding optimal NFA's equivalent to a given DFA. Here we will require the resulting NFA to be of a special type. To give the flavor of these problems, let us consider the Concatenation Equivalence problem: Given three DFA's M_1, M_2, and M_3, decide whether $L(M_1) \cdot L(M_2) = L(M_3)$. An analogous optimization problem (which we call the *Minimum Concatenation Generation problem*) is the following: Given a DFA M and an integer k, find two DFA's M_1, M_2, if possible, such that $|M_1| + |M_2| \leq k$ and $L(M) = L(M_1) \cdot L(M_2)$. It turns out that both of these problems are PSPACE-complete. It follows that the presence of just one nondeterministic operation, such as concatenation, makes the equivalence problem extremely *hard.* Of course, one can replace concatenation by any other regularity-preserving operator Op (unary or binary) and study the Op *Equivalence problem* and the *Minimum* Op *Generation problem.* We study these problems for all the fundamental regularity-preserving unary and binary operations.

The Op Equivalence problems were motivated by the fact that when certain operations, such as concatenation or Kleene star, are applied, the resulting languages require DFA's with exponentially more states than those of the DFA for the original language(s) [Ra89]. Thus the standard method of converting to DFA and testing equivalence will not work. We are therefore led to seek other methods to answer these decision questions or demonstrate that none is likely to exist.

It should be remarked that the results presented in this section are not related to the well-developed algebraic decomposition theory of which the central result is the Krohn–Rhodes decomposition theorem [Gi68], [Ha66]. The fundamental difference between our results and the classical decomposition theory is that in the latter theory the FA is required to be decomposed into structurally simple automata (such as reset automata and permutation automata), whereas we want the constituents to be simple in the sense of being small in size.

We now turn to technical details. The rest of this section is divided into two subsections. In §4.1 we study the complexity of the Op Equivalence problem for Op $\in \{\cup, \cap, \cdot, *, \text{rev}\}$. In §4.2 we study Minimum Op Generation problems for the same operators.

4.1. Equivalence problems. In this subsection we will study the Minimum Op Equivalence problem for Op $\in \{\cup, \cap, \cdot, *, \text{rev}\}$. It is easy to see that this problem is decidable in polynomial time for union and intersection since we can efficiently find a small DFA accepting $L(M_1)\text{Op } L(M_2)$ for Op $\in \{\cup, \cap\}$. Thus the problem is reduced to equivalence testing of two DFA's, which has an almost linear time algorithm [Ho79]. The problem is more interesting for Op $=$ rev. It is easy to see that the minimum DFA accepting $L(M)^{\text{rev}}$ can be exponentially larger than M [Ra89], and so the conversion method is not efficient. Nevertheless, the problem can be shown to be in P. Given DFA's M_1 and M_2, we want to determine whether

$L(M_1) = L(M_2)^{\text{rev}}$. We design an NFA M_3 from M_2 by reversing its arcs and renaming the starting and accepting states. (If M_2 has more than one accepting state, we introduce a new start state and an ϵ arc from it to every accepting state of M_1, and we remove ϵ-moves by using the standard algorithm [Ho79].) It can be easily seen that M_3 accepts $L(M_2)^{\text{rev}}$ and that M_3, although nondeterministic, is unambiguous. Now the polynomial-time algorithm of Stearns and Hunt [St85] for equivalence testing of UFA's decides the answer to the original problem with inputs M_1 and M_3. This leaves us with two basic operations: concatenation and Kleene star. It is known that both operations can blow up the number of states in minimum DFA exponentially [Ra89], and so the conversion method leads to an exponential-time algorithm. We show in Theorems 4.1 and 4.2 below that both the problems are PSPACE-complete. These results are surprising and differ from the known results in that all known hard decision problems involve NFA's or an unbounded number of DFA's (e.g., Kozen's result on the Finite Automata Intersection Problem [Ko77]). We have the following theorem:

THEOREM 4.1. *The Concatenation Equivalence* (CE) *problem is* PSPACE-*complete.*

Proof. CE is easily seen to be in PSPACE. In what follows we will show its PSPACE-hardness. The proof is by reduction from the Linear-Space Acceptance problem [Ga78]. Let M be a fixed (one-tape) deterministic linear-bounded automaton (Turing machine), and let w be an input to M. The question is, "Does M accept w?" We show how to transform (in polynomial time) this instance to three DFA's M_1, M_2, and M_3 such that M does not accept w if and only if $L(M_1) \cdot L(M_2) = L(M_3)$. Let $|w| = n - 1$. Let Γ be the work-tape alphabet, let Q be the set of states, and let F be the set of accepting states of M. Let ¢ and \$ be the left and right end markers on the tape. Let # be a special symbol in Γ. We assume without loss of generality that in any accepting computation M writes a # on every tape square (including the two end squares) and enters an accepting state. Assume further that M makes at least two moves on every input. Let $S = \Gamma \cup Q$.

Informally, the idea behind the proof is as follows. Let INV_w^M be the set of invalid computations of M on input w. Thus INV_w^M is either S^* (if $w \notin L(M)$) or S^* with the exclusion of *exactly* one string (namely, the valid computation of M on w, if $w \in L(M)$). We show that R defined as $R = (\text{INV}_w^M)^{\text{rev}}$, the reverse of the set of invalid computations, can be expressed as $R = L_1 \cdot L_2$ for two regular languages L_1 and L_2, both of which can be accepted by "small" DFA's. Let us describe L_1 and L_2 informally. Note that the reverse of the valid computation of M on input w is of the form $w' = \text{ID}_m^{\text{rev}}\text{ID}_{m-1}^{\text{rev}} \cdots \text{ID}_0^{\text{rev}}$, where $|\text{ID}_i| = n$ for each i, the start and the final ID are correct, and the successive ID's in the list yield the preceding ID. Thus for a string w' to be in R one of the following conditions should hold: (i) the start of accepting ID is bad, or (ii) it is not true that $\text{ID}_j \vdash \text{ID}_{j+1}$ for some j. Since condition (i) can be easily handled, let us concentrate on (ii) in this informal outline. Suppose that $\neg(\text{ID}_j \vdash \text{ID}_{j+1})$ and that the position of mismatch is t. Let $w' = x_1 \cdot x_2$, where $|x_1| = n - t$. (We set this length to $n - t$ rather than to t since we consider the reverse string.) Now the position of mismatch occurs in x_2 at a position that is a multiple of n. Thus a small DFA that counts modulo n can locate the mismatch by using a small buffer (of constant number of bits). The formal details are slightly more complicated in that the compatibility checking may require matching three symbols of ID_i with three symbols of ID_{i+1} (when a state symbol is involved in the mismatch). We should further handle the starting and accepting ID's. Now we present the formal details.

For strings $a_1a_2a_3$ and $b_1b_2b_3$ (where a_i, $b_i \in S$) we say that $a_1a_2a_3 \Rightarrow b_1b_2b_3$ if (i) $a_1, a_2, a_3 \in S - Q$ and $a_2 = b_2$ or (ii) $a_2 \in Q$ and $a_3a_2a_1 \vdash b_3b_2b_1$. Thus $\Rightarrow$ is simply an extension of $\vdash$ for strings of length three to include the identity relation. We now define language L_1 as

$$L_1 = \{x | x \in S^*, |x| \leq n - 3\}.$$

L_2 is defined as the set of strings x over S satisfying one of the following properties:

(i) The leading three symbols are not all in $\{\#\} \cup F$.

(ii) ID_0 is not a suffix of x.

(iii) Let $x = x_0 x_1 \cdots x_m$, where each $x_i \in S$. There exists a t such that $0 \leq tn \leq m - n - 2$ and it is not true that $x_{(t+1)n} x_{(t+1)n+1} x_{(t+1)n+2} \Rightarrow x_{tn} x_{tn+1} x_{tn+2}$.

Note that there are DFA's with $O(n)$ and $O(n^2)$ states recognizing L_1 and L_2, respectively, and that these DFA's can be designed in polynomial time. The claim follows from the fact that $L(M_1) \cdot L(M_2) = S^*$ if and only if M does not accept w. $\square$

Observe that what we have shown is stronger than the claim of Theorem 4.1 in that the proof holds when M_3 accepts Σ^*. It is easy to modify the above proof to show that the following problem is PSPACE-complete: Inputs are DFA's M_1 and M_2 over an alphabet Σ such that $\epsilon \notin L(M_1)$ and $\epsilon \notin L(M_2)$. The question is, "Is $L(M_1) \cdot L(M_2) = L_0 = \{x | x \in \Sigma^*, |x| \geq 2\}$?" This stronger result is needed to prove the next result. We use the name SCE to denote this problem.

THEOREM 4.2. *The Kleene Star Equivalence* (KSE) *problem is* PSPACE-*complete.*

Proof. This problem is easily seen to be in PSPACE. To prove its PSPACE-hardness we reduce SCE to KSE. SCE's instance is specified by two DFA's M_1 and M_2 such that $\epsilon \notin L(M_1)$ and $\epsilon \notin L(M_2)$. The question to be answered is whether $L(M_1) \cdot L(M_2) = L_0 = \{x | x \in \Sigma^*, |x| \geq 2\}$. We will transform this (in polynomial time) to an instance of KSE. Let the input size of the instance of SCE be $n = |M_1| + |M_2|$. Let # be a new symbol not in Σ, and let $\Gamma = \Sigma \cup \{\#\}$.

Define the following languages:

$$\begin{aligned}
L_1 &= \#L(M_1), \\
L_2 &= L(M_2)\#, \\
L_3 &= (\Sigma^+\#\Sigma^*\#\Sigma^*) \cup (\Sigma^*\#\Sigma^*\#\Sigma^+), \\
L_4 &= \Gamma^*\#\Gamma^*\#\Gamma^*\#\Gamma^*, \\
L_5 &= L_1 \cup L_2 \cup L_3 \cup L_4, \\
L_6 &= \{\epsilon\} \cup L_1 \cup L_2 \cup L_3 \cup L_4 \cup (\#L_0\#).
\end{aligned}$$

We now make the following claims:

(i) L_5 and L_6 can be accepted by DFA's M_3 and M_4, both of size $O(n)$, and these DFA's can be constructed from M_1 and M_2 in time linear in n.

(ii) $L_5^* \subseteq L_6$ and $L_6 - L_5^* = (\#L_0\#) - (L_1 L_2)$.

The proof of (i) is obvious. For the proof of (ii) let $w \in L_5^*$. We will show that $w \in L_6$. We will consider mutually exclusive and collectively exhaustive possibilities for w.

Case 1: w does not contain any #. Clearly, $w = \epsilon \in L_6$.

Case 2: w has exactly one occurrence of #. In this case $w \in L_1 \cup L_2$, and so $w \in L_6$.

Case 3: w has at least three occurrences of #. Then $w \in L_4$, and so $w \in L_6$.

Case 4: w has exactly two occurrences of #, and w either begins or ends with a symbol in Σ. In this case $w \in L_3$, and so $w \in L_6$.

Case 5: w has exactly two occurrences of #, and w starts and ends with #. Note that this case occurs only when $w \in L_1 \cdot L_2$. (This is why we require that neither M_1 nor M_2 accept ϵ.) Thus $w \in L_1 \cdot L_2 \subseteq \#L_0\# \subseteq L_6$.

This argument further implies that $L_6 - L_5^*$ consists of exactly the set of strings of the form $\#z\#$, where $z \notin L_1 \cdot L_2$. This proves (ii).

The reduction is complete with M_3 and M_4 forming the instance of KSE. We conclude the proof by observing that $L(M_1) \cdot L(M_2) = L_0$ if and only if $L_5^* = L_6$. □

We briefly remark on the complexity of containment problems. In our setting the containment questions are of two types: Is $L(M_1)\mathrm{Op}\, L(M_2) \subseteq L(M_3)$? and Is $L(M_3) \subseteq L(M_1)\mathrm{Op}\, L(M_2)$? for input DFA's M_1, M_2, and M_3. Both problems are easily seen to be solvable in polynomial time for Op = union, intersection, or reversal. The last result follows from the result of Stearns and Hunt [St85] that containment is decidable in polynomial time for UFA's. Recall that containment problems involving (general) NFA's are hard [Hu76] since the universe problem is essentially the containment question "Is $\Sigma^* \subseteq L(M)$?", given M as input. The same argument carries over to concatenation since M_3 actually accepts Σ^* in Theorem 4.1. Thus the following problem is PSPACE-complete: Decide whether $L(M_3) \subseteq L(M_1) \cdot L(M_2)$ for given DFA's M_1, M_2, and M_3. However, the inclusion in the other direction is easily seen to be decidable in P:

RESULT 4.1. *There is a polynomial-time algorithm for deciding whether $L(M_1) \cdot L(M_2) \subseteq L(M_3)$, given* DFA*'s M_1, M_2, M_3 as inputs.*

Proof. Note that the above containment is equivalent to $L(M_1) \cdot L(M_2) \cap \overline{L(M_3)} = \emptyset$. We can efficiently design NFA's accepting $L(M_1) \cdot L(M_2)$ and $\overline{L(M_3)}$. We can design an NFA (whose size is the product of the sizes of the two NFA's) accepting the intersection of these languages. Finally, we can test the emptiness of the language accepted by an NFA by using the reachability algorithm. □

Similarly we can show the following:

RESULT 4.2. *There is a polynomial-time algorithm for deciding whether $L(M_1)^* \subseteq L(M_2)$, given* DFA*'s M_1 and M_2 as inputs.*

Result 4.3 below follows from Result 4.2.

RESULT 4.3. *The following problem is* PSPACE*-complete*:

INSTANCE: Two DFA's M_1, M_2.

QUESTION: Is $L(M_2) \subseteq L(M_1)^*$?

Proof. If there is a polynomial-time algorithm for this problem, then, in conjunction with Result 4.2, it will yield a polynomial-time algorithm for KSE that will imply that PSPACE = P. □

4.2. Minimization problems. In this section we present results on the complexity of Minimum Op Generation problems for all of the operations considered in §4.1, namely, union, intersection, concatenation, Kleene star, and reversal. We show that the problems are NP-complete for union and intersection and that they are PSPACE-complete for concatenation and Kleene star. We also present a pseudo-polynomial-time algorithm for reversal. The latter result implies that the diversity-based representation of a regular language can be obtained from a given DFA M in time polynomial in $n = |M|$ and D, the diversity [Ri87]. (Here D is assumed to be in unary.)

THEOREM 4.3. *The Minimum Union Generation* (MUG) *and Minimum Intersection Generation* (MIG) *problems are* NP*-complete.*

Proof. We will prove the result only for MUG since the NP-completeness of MIG readily follows from that of MUG by complementation. MUG is clearly in NP. We prove its NP-hardness by reduction from the Minimum Inferred DFA problem [Go78]. Let Σ be a finite alphabet, let $S, T \in \Sigma^*$ be two finite sets of strings (called the positive and negative samples, respectively), and k an integer form an instance of the Minimum Inferred DFA problem. The question is, "Is there a DFA with at most k states such that M accepts *every* string in S and none in T?" (We do not care about the behavior of M on other strings.) We say that such an

M is *consistent* with S and T. We assume without loss of generality that $S \cap T = \emptyset$. Let # be a new symbol not in Σ. For a regular language L let *size*(L) denote the size of the minimum DFA accepting L. Let $m = k +$ size$(\overline{T} \cap \overline{S})$.

Define the following languages:

$$\begin{aligned} L_1 &= \overline{T}, \\ L_2 &= \overline{T} \cap \overline{S}, \\ L_3 &= (L_2\#^m)^+ \quad \text{(recall the definition of } m \text{ above)}, \\ L_4 &= L_1\#^m, \\ L_5 &= L_3 \cup L_4. \end{aligned}$$

We prove the following claims:

CLAIM 4.1. *Let L be a language. Then*

$$\text{size}(L\#^m) = \text{size}((L\#^m)^+) = m + \text{size}(L).$$

Proof. Let M be the minimum DFA accepting L. A DFA M' accepting $L\#^m$ can be designed by adding a #-tail of length m to M and by letting $\delta(f, \#) = p_0$ for every final state f of M, where p_0 is the first state in the #-tail. Clearly, $L(M') = (L\#^m)$, so size$(L\#^m) \leq$ size$(L) + m$. To prove the reverse inequality let M' be the minimum DFA accepting $L\#^m$. We can obtain a DFA M from M' by removing all states having an incoming #-arc. We would have removed exactly m states in this process, and so $|M| = |M'| - m$ and the claim follows. We can also show that size$((L\#^m)^+) =$ size$(L\#^m)$. This concludes the proof of Claim 4.1. □

CLAIM 4.2. $L_1\#^m = (L_2 \cup L(M'))\#^m$ *for any* DFA M' *consistent with S and T.*

Proof. Since $S \cap T = \emptyset$, $S \subseteq \overline{T}$. Thus

$$\begin{aligned} L_1 &= \overline{T} \\ &= (\overline{T} \cap \overline{S}) \cup (\overline{T} \cap S) \\ &= \overline{T} \cap \overline{S} \cup S \\ &= L_2 \cup S. \end{aligned}$$

Observe that $S \subseteq L(M') \subseteq \overline{T}$ and that $L_2 \cup L(M') \subseteq L_1$. We have

$$\begin{aligned} L_1\#^m &= (L_2 \cup S)\#^m \\ &\subseteq (L_2 \cup L(M'))\#^m \\ &\subseteq L_1\#^m \qquad \text{(since } L_2 \cup L(M') \subseteq L_1). \end{aligned}$$

This concludes the proof of Claim 4.2. □

Now we present the reduction. Let M be the minimum DFA accepting L_5. The instance of MUG to which S, T, k has been transformed is M, $3m$. Let $n = |S| + |T| + k$ be the size of the instance of the Minimum Inferred DFA problem. It is easy to see that M can be constructed from S, T, and k in time bounded by a polynomial in n. The correctness of the reduction is stated as the next claim.

CLAIM 4.3. *There is a k-state* DFA *consistent with S and T if and only if $L(M) = L_5$ is the union of $L(M_1)$ and $L(M_2)$ for two* DFA's M_1 *and* M_2 *such that* $|M_1| + |M_2| \leq 3m$.

Proof. (only if) Suppose that there is a k-state DFA M' consistent with S and T. Let M_1 be the minimum DFA accepting L_3, and let M_2 be the minimum DFA accepting $L(M')\#^m$. We show that M_1 and M_2 satisfy our requirements. First we show that $|M_1| + |M_2| \leq 3m$. By Claim 4.1, $|M_2| = k + m$ and $|M_1| =$ size$((L_2\#^m)^+) =$ size$(L_2) + m$. Thus $|M_1| + |M_2| = k + m +$ size$(L_2) + m = 3m$. Next we show that $L(M_1) \cup L(M_2) = L_5$.

$$\begin{aligned}
L(M_1) \cup L(M_2) &= (L_2\#^m)^+ \cup (L(M')\#^m) \\
&= (L_2\#^m)^+ \cup (L_2\#^m) \cup (L(M')\#^m) && \text{(since } L_2\#^m \subseteq (L_2\#^m)^+\text{)} \\
&= (L_2\#^m)^+ \cup (L_2 \cup L(M'))\#^m \\
&= (L_2\#^m)^+ \cup (L_1\#^m) && \text{(by Claim 4.2)} \\
&= L_3 \cup L_4 \\
&= L_5.
\end{aligned}$$

(if) Suppose L_5 can be expressed as the union of $L(M_1)$ and $L(M_2)$ for some DFA's M_1 and M_2 such that $|M_1| + |M_2| \leq 3m$. We can assume without loss of generality that both $L(M_1)$ and $L(M_2)$ are nonempty. (If not, let $L(M_1) = \emptyset$. Then $L(M_2) = L_5$, and we can show that $|M_2|$ must be at least m^2.) We further assume that M_1 and M_2 are minimized. We then make the following claim.

CLAIM 4.4. *Either* (i) $L(M_1) \supseteq S\#^m$ *and* $L(M_2) = L_3$ *or* (ii) $L(M_2) \supseteq S\#^m$ *and* $L(M_1) = L_3$.

Proof. Informally, the idea behind the proof can be outlined as follows. Since $L(M_1) \cup L(M_2) = L_5$, the two DFA's M_1 and M_2 should collectively have a #-tail (a sequence of m #-arcs ending at some state with no outgoing arc) and a #-waist (a sequence of m #-arcs ending at some state with at least one outgoing arc). Since $L(M_1)$ and $L(M_2)$ are nonempty, both must contain either a #-tail or a #-waist. But if one of them contains both a #-tail and a #-waist, then $|M_1| + |M_2|$ will exceed $3m$. Thus exactly one of them contains a #-tail and the other contains a #-waist. We can then show that the DFA containing the #-tail must accept every string in $S\#^m$ and that the DFA containing the #-waist must accept the language L_3.

Formally, a #-tail is defined as a sequence of states $q_1, q_2, \ldots, q_m$ such that (i) $\delta(q_i, \#) = q_{i+1}$ for $i = 1, 2, \ldots, m-1$ and (ii) q_m has no outgoing arc. A #-waist is defined as a sequence of states $q_1, q_2, \ldots, q_m$ such that the following conditions hold: (i) for all $i = 1, 2, \ldots, m-1$, $\delta(q_i, \#) = q_{i+1}$ and (ii) q_m has at least one outgoing arc. Claim 4.4 is an easy consequence of the following observations, which we state without proof:

Observation 4.1. M_i must contain a #-tail or a #-waist (or both).

Observation 4.2. If either M_1 or M_2 contains both a #-tail and a #-waist, then $|M_1|+|M_2| > 3m$.

Observation 4.3. If M_i contains the #-tail, then $L(M_i) \supseteq S\#^m$. (To prove this observation, note that $L_2 \cap S = \emptyset$. Thus the DFA containing the #-waist does not accept any string in $S\#^m$.)

Observation 4.4. If M_i contains the #-waist, then $L(M_i) = L_3$.

This concludes the proof of Claim 4.4. □

Suppose M_2 contains the #-tail (the other case is identical). Then $S\#^m \subseteq L(M_2) \subseteq L_4$, i.e., $S \subseteq L(M_2)/\#^m \subseteq L_1 = \overline{T}$. Thus L_6 defined as $L_6 = L(M_2)/\#^m$ is a regular language consistent with S and T.

The rest of the proof of Theorem 4.3 is aimed at showing that a k-state DFA accepting L_6 can be designed from M_2 by removing the #-tail from it. We do this by showing that the size of M_2 is at most $m + k$.

CLAIM 4.5. $|M_2| \leq m + k$.

Proof. Observe that $|M_2| \leq 3m + 1 - |M_1|$. We also note that $|M_1| \geq$ size(L_3). (This is true because $L(M_1) = L_3$ and M_1 is minimized.) Hence

$$\begin{aligned}
|M_1| &= \text{size}(L_3) \\
&= \text{size}((L_2\#^m)^+) \\
&= m + \text{size}(L_2) && \text{(by Claim 4.1)} \\
&= 2m - k && \text{(by definition of } m).
\end{aligned}$$

Thus $|M_2| \leq 3m - (2m - k) = m + k$. □

Thus the DFA M accepting $L_6 = L(M_2)/\#^m$ has at most k states since all states in the #-tail of M_2 have been removed. This concludes the proof of Claim 4.3 and Theorem 4.3. □

We next consider concatenation and Kleene star. We show that the problems corresponding to both of these operations are PSPACE-complete. We need the following lemma in the proofs of Theorems 4.4 and 4.5.

LEMMA 4.1. *For every sufficiently large positive integer n there is a regular language L_3 over* $\{0, 1\}$ *such that* (i) *the minimum* DFA *accepting L_3 is of size at least n^2 but at most $O(n^2)$ and* (ii) *L_3 can be expressed as $L_4 \cdot L_5$ such that L_4 and L_5 can be accepted by* DFA's *of size at most n.*

Proof. Let $L_3 = \{x1y | x \in (0+1)^*, |y| = 2\log_2 n\}$. The claim follows since $L_3 = L_4 \cdot L_5$, where $L_4 = (0+1)^*1$ and L_5 is the set of all strings of length $2\log_2 n$. We omit the easy proof that conditions (i) and (ii) are satisfied. □

THEOREM 4.4. *The Minimum Concatenation Generation problem* (MCG) *is* PSPACE-*complete.*

Proof. MCG is easily seen to be in PSPACE. In what follows we will prove its PSPACE-hardness by reducing CE to MCG in polynomial time. (Recall that CE was shown to be PSPACE-complete in Theorem 4.1.) Let M_1, M_2, M_3 be three DFA's forming an instance of the CE problem. We may assume without loss of generality that $L(M_3) = \Sigma^*$ and that M_1 and M_2 are minimal. Let $n = |M_1| + |M_2|$, let $L_1 = L(M_1)$, and let $L_2 = L(M_2)$. Let 0 and 1 be two new symbols not in Σ. Let L_3, L_4, and L_5 be as in Lemma 4.1, and let # be another new symbol.

Now let

$$L = (L_1\#L_3\#L_2) \cup (L_1\#L_4 \cdot L_2) \cup (L_1 \cdot L_5\#L_2) \cup \Sigma^*.$$

Let M be the minimum DFA accepting L. It can be shown that $|M| = \Theta(n^2)$ and that M can be constructed from M_1 and M_2 in polynomial time. The reduction is complete with M and $k = 3n$ forming an instance of the MCG problem. We complete the proof by establishing the correctness of the reduction:

CLAIM 4.6. *$L_1 \cdot L_2 = \Sigma^*$ if and only if there exist* DFA's *N_1, N_2 such that $L(N_1) \cdot L(N_2) = L$ and $|N_1| + |N_2| \leq k$.*

Proof. (only if) Let $L(N_1) = L_1(\#L_4 \cup \{\epsilon\})$, let $L(N_2) = (L_5\# \cup \{\epsilon\})L_2$, and choose N_1 and N_2 to be minimal. It is easy to show that $|N_1| \leq n + 2$ and that $|N_2| \leq n + 2\log_2 n + 1$, so that $|N_1| + |N_2| \leq 2n + 2\log_2 n + 3 \leq 3n$ for large enough n.

(if) It is easy to see that if $L(N_1)$ contains any string in $\Sigma^*\#(0+1)^+\#$, then $L(N_2)$ can only have strings over Σ. Thus one of the following must be true: (i) $L(N_i) \subseteq \Sigma^*$ for either $i = 1$ or $i = 2$ or (ii) $L(N_1) \subseteq (\Sigma^*\#(0+1)^*) \cup \Sigma^*$ and $L(N_2) \subseteq ((0+1)^*\#\Sigma^*) \cup \Sigma^*$.

Suppose that (i) is true and that $L(N_1) \subseteq \Sigma^*$ (the proof for $L(N_2) = \Sigma^*$ is identical). We can show in this case that $|N_2| \geq \Omega(n^2)$: The idea is to note that we can get a DFA for L_3 from N_2 by removing some (but not adding any) states. We omit the details.

Thus (ii) must be true. Let

$$L_6 = L(N_1) \cap \Sigma^*,$$
$$L_7 = L(N_2) \cap \Sigma^*.$$

Then clearly $L_6 \cdot L_7 = \Sigma^*$. Also,

$$L_6 \cdot (L(N_2) \cap ((0+1)^*\#\Sigma^*)) = L_1 \cdot L_5\#L_2,$$

$$(L(N_1) \cap (\Sigma^*\#(0+1)^*)) \cdot L_7 = L_1\#L_4 \cdot L_2.$$

Since neither L_4 nor L_5 is empty, it follows that $L_6 = L_1$ and $L_7 = L_2$. Thus $L_1 \cdot L_2 = \Sigma^*$. This concludes the proofs of Claim 4.6 and Theorem 4.4 $\square$

We next consider Kleene star. The Minimum Kleene Star Generation (MKSG) problem is also PSPACE-complete. The proof is similar to that of Theorem 4.2, but it is more complicated.

THEOREM 4.5. *The Minimum Kleene Star Generation problem is* PSPACE-*complete.*

Proof. The reduction is from SCE. Let the instance of SCE be M_1 and M_2 (over alphabet Σ) such that $L(M_1)$ and $L(M_2)$ do *not* contain ϵ. Recall that we want to decide whether $L(M_1) \cdot L(M_2) = L_0 = \{x \mid x \in \Sigma^*, |x| \geq 2\}$. Let $n = |M_1| + |M_2|$.

Let 0, 1, and # be new symbols not in Σ, and let $\Gamma = \{0, 1, \#\} \cup \Sigma$. Let L_3, L_4, and L_5 be as in Lemma 4.1. Note that L_5 is finite.

We now define the following new languages:

$$\begin{aligned}
L_6 &= L(M_2) \cdot L_4,\\
L_7 &= \#L(M_1),\\
L_8 &= L_5\#,\\
L_9 &= \Gamma^*(0+1)\Sigma\Gamma^*,\\
L_{10} &= \Gamma^+\#\Gamma^+,\\
L_{11} &= (\#L_0 \cdot L_4) \cup (L(M_2) \cdot L_3\#) \cup (\#L_0 \cdot L_3\#),\\
L_{12} &= L_6 \cup L_7 \cup L_8 \cup L_9 \cup L_{10},\\
L_{13} &= L_6^* \cup L_7 \cup L_8 \cup L_9 \cup L_{10} \cup L_{11}.
\end{aligned}$$

We state and prove the following claims about these languages:

CLAIM 4.7. *If* $L(M_1) \cdot L(M_2) = L_0$, *then* $L_{12}^* = L_{13}$.

CLAIM 4.8. *The minimum* DFA M *accepting* L_{13} *has at least* $\Omega(n^2)$ *states, and it can be designed in polynomial time.*

CLAIM 4.9. *There is a* DFA M *with* $25n$ *states that accepts* L_{12}.

Proof of Claim 4.7. Suppose $L(M_1) \cdot L(M_2) = L_0$. Then it can be observed that $L_{13} = L_7 \cdot L_6 \cup L_6 \cdot L_7 \cup L_7 \cdot L_6 \cdot L_8$, and so it follows that $L_{12}^* \subseteq L_{13}$. The reverse inclusion is more complicated. Let $w \in L_{12}^*$. We will show that $w \in L_{13}$. We have to consider many cases. (Assume that $w \neq \epsilon$.)

Case 1: w contains three or more occurrences of #. Then $w \in L_{10}$, and so $w \in L_{13}$.

Case 2: $w \in L_6^*$, i.e., w was obtained by repeated use of L_6 alone. Then obviously $w \in L_{13}$.

Case 3: w has a substring in L_9 or L_{10}. In this case it is easily seen that $w \in L_9$ or $w \in L_{10}$ and hence in L_{13}.

If none of these cases applies, then w should satisfy the following conditions: (a) w begins or ends with a # symbol, (b) w has at most two occurrences of #, and (c) $w \in (L_6 \cup L_7 \cup L_8)^* - L_6^*$. Now it is easy to see that w should be in $L_7L_6^* \cup L_6^*L_8 \cup L_7L_6^*L_8$. Since $L_7L_6^* \cup L_6^*L_8 \cup L_7L_6^*L_8 \subseteq L_{11} \subseteq L_{13}$, the claim follows. $\square$

Proof of Claim 4.8. The lower bound on the size of M follows from the observation that a DFA for L_3 can be obtained from a DFA for L_{13} by removing (but not adding) states. We leave out the easy proof that M can be designed in polynomial time. $\square$

We omit the proof of Claim 4.9.

The reduction is complete with the DFA M for L_{13} and $k = 25n$ forming the instance of the MKSG problem. Obviously the reduction can be carried out in polynomial time, as noted in Claim 4.8. We prove the correctness of the reduction:

CLAIM 4.10. *$L(M_1) \cdot L(M_2) = L_0$ if and only if there is a* DFA *M with k states such that $L_{13} = L(M)^*$.*

Proof. (only if) This readily follows from the observations made above. One choice of M is the DFA of Claim 4.9.

(if) This is more involved. We outline a sketch of the proof. Since neither $L(M_2)$ nor L_4 contains ϵ, it is easy to see that

$$\begin{gathered} L(M) \cap (\Sigma^* \cup (0+1)^*) = \emptyset, \\ L(M) \cap (\Sigma^+ \cdot (0+1)^+) = L_6, \\ L(M) \cap (\#\Sigma^*) = L_7, \\ L(M) \cap (\#\Sigma^* \cdot (0+1)^*) = L_7 \cdot L_6, \\ L(M) \cap ((0+1)^*\#) = L_8, \\ L(M) \cap (\Sigma^*(0+1)^*\#) = L_6 \cdot L_8. \end{gathered}$$

Let $L_{14} = L(M) \cap (\#\Sigma^*(0+1)^*\#)$, and let L_{15} be obtained by homeomorphically erasing the symbols of $\Sigma \cup \{\#\}$ from L_{14}. More formally, let $h : \Gamma \to \Gamma^*$ be defined as $h(a) = a$ for $a \notin \Sigma \cup \{\#\}$, $h(a) = \epsilon$ if $a \in \Sigma \cup \{\#\}$. Then let $L_{15} = h(L_{14})$.

Clearly, $L_{15} \subseteq L_3$. In fact, we can show that $L_{13} \subset L_3$. (Otherwise, it follows that there is a k-state DFA accepting L_3 since we can obtain a k-state DFA for L_3 by suitably removing some (and not adding any) states from M.) Moreover, one can show that $L_3 - L_{13}$ is infinite. (Otherwise, one can still construct a k-state DFA for L_3 by removing states from M.)

Let x be a string in $L_3 - L_{13}$ such that x is longer than all strings in L_5. Note that $x \in (0+1)^+$. Clearly,

$$(\#L_0 \cdot x\#) \subseteq L_9 \subseteq L_{11} = L(M)^*.$$

It is easy to see that (again note that $L_6 \subseteq \Sigma^+ \cdot (0+1)^+$)

$$\begin{aligned} (\#L_0 \cdot x\#) &= L(M)^* \cap (\#L_0 \cdot x\#) \\ &= ((L_7 \cdot L_6 \cdot L_8) \cap (\#L_0 \cdot x\#)) \cup (L_7 \cdot L_8 \cap (\#L_0 \cdot x\#)) \cup (L_{14} \cap (\#L_0 \cdot x\#)) \\ &= ((L_7 \cdot L_6 \cdot L_8) \cap (\#L_0 \cdot x\#)) \cup (L_7 \cdot L_8 \cap (\#L_0 \cdot x\#)) \\ &\qquad\qquad\qquad\qquad\qquad\qquad (\text{since } L_{14} \cap (\#L_0 \cdot x\#) = \emptyset) \\ &= ((\#L(M_1) \cdot L(M_2) \cdot L_4 \cdot L_5\#) \cap (\#L_0 \cdot x\#)) \cup ((\#L(M_1) \cdot L_5\#) \cap (\#L_0 \cdot x\#)). \end{aligned}$$

Since x is longer than all strings in L_5,

$$(\#L(M_1) \cdot L_5\#) \cap (\#L_0 \cdot x\#) = \emptyset.$$

Thus $(\#L(M_1) \cdot L(M_2) \cdot L_4 \cdot L_5\#) \cap (\#L_0 \cdot x\#) = (\#L_0 \cdot x\#)$. That is, $L(M_1) \cdot L(M_2) = L_0$. This completes the proofs of Claim 4.10 and Theorem 4.5. $\square$

We conclude §4.2 with the Minimum Reversal Generation (MRG) problem. The problem is to find, given a DFA M and an integer k, whether there is a DFA M' with at most k states such that $L(M) = L(M')^{\text{rev}}$ or, equivalently, $L(M)^{\text{rev}} = L(M')$. Let M^{rev} denote the reverse of M obtained by using the standard algorithm (see the beginning of §4.2). It is easy to see that if M_1 is deterministic, then M_1^{rev} is unambiguous. Thus the above problem is a special case of the problem of converting a UFA to a minimum DFA left open in §3. We show that the MRG problem has a pseudo-polynomial-time algorithm, i.e., it has a polynomial-time algorithm if the input k is bounded by a polynomial in n the size of M or, equivalently, if k is presented in unary. Note, as remarked in §3, that the more general problem of finding whether there is a k-stage DFA equivalent to a given NFA is NP-hard even if k is fixed.

The MRG problem can be stated in terms of diversity of a finite automaton as defined in [Ri87]. Let M be a DFA. Two strings x and y are defined as equivalent (with respect to M) if for all states q, $\delta(q, x)$ is an accepting state if and only if $\delta(q, y)$ is. This equivalence relation induces a partition of Σ^*, and the number of equivalence classes induced is called the *diversity* of M. If $L(M_1) = L(M_2)$, then the diversity of M_1 is equal to the diversity of $L(M_2)$, so that we can define the diversity of a regular language L as the diversity of DFA accepting L. It has been observed by Young and Angluin [Sc88] that the diversity of a regular language L is the size of the minimum DFA accepting L^{rev}. Thus the MRG problem can be restated as the problem of finding, given k and a DFA M, whether the diversity of $L(M)$ is bounded by k.

THEOREM 4.6. *There is a pseudo-polynomial-time algorithm for the* MRG *problem.*

Proof. Let M, a DFA, and integer k be given as inputs. Let $M_1 = M^{\text{rev}}$. We apply the standard algorithm for converting an NFA to a DFA by subset construction [Ho79] to M' with the condition that it produce only live subsets. We apply this algorithm on M' until the number of states generated exceeds k. In this case the algorithm outputs "no." If the conversion is complete with k or fewer subsets generated, the output is "yes." Clearly, the algorithm runs in time bounded by a polynomial in $k + |M|$. It is also clear that the "yes" answer of the algorithm is correct. We show that the "no" answers are also correct by showing that the resulting DFA produced by the subset construction method is, in fact, minimal when applied to a reverse deterministic NFA.

Let M be a DFA with states $Q = \{q_1, q_2, \ldots, q_n\}$. Define $L(q_i)$ as the set of strings accepted by M is q_i is considered as the start state and if there is no other change in M. In the same way define $P(q_i)$ as the set of strings accepted by M is q_i is considered the only accepting state with no other change in M. Let M_1 be the resulting DFA when the subset construction method is applied to M^{rev}. The states of M_1 are named by the subsets of Q. Let $T_1, T_2 \subseteq Q$ be two states of M_1. We observe that $L(T_i)$ can be expressed as

$$L(T_i) = \bigcup_{q \in T_i} P(q)^{\text{rev}}, \qquad i = 1, 2.$$

Since M is a DFA, $P(q_1), P(q_2), \ldots, P(q_n)$ are pairwise disjoint. It follows that $T_1 \neq T_2$ implies $L(T_1) \neq L(T_2)$. This concludes the proof. ☐

5. Conclusions. The complexity of decision problems for various types of language-generating mechanisms has been a favorite area of research since the pioneering work of Cook [Co71] and Karp [Ka72]. Such problems have many applications, including text editing, data storage and retrieval, and parsing. The problem of converting one type of finite-state automaton to an optimal finite automaton of another type is a fundamental one. Our main contribution is to complement the classical work of [Ra59], [Me71], [Me72], [St73], [Hu76] by showing that DFA $\rightarrow$ NFA and other related problems are hard.

Our results in §4 reveal that even the weakest forms of nondeterminism can render a decision problem intractable. These results substantially strengthen the known hardness results in which the input machines are generally assumed to have unrestricted nondeterminism. Although our proofs use the classical technique of reduction, the reductions and the connections we have established are new; they may find wider applications in the study of other related problems.

In a companion paper [Ji90] we study the DFA $\to$ NFA problem over a unary alphabet. It turns out that this problem is hard even in this special case. However, the evidence of hardness we can provide is weaker than that for the standard ones, such as NP-hardness and PSPACE-hardness. The result can be stated more precisely as follows: A *unary cyclic* DFA is a DFA over a unary alphabet, in which the states form a simple cycle. A unary cyclic language is a regular language accepted by a unary cyclic DFA. The main result of [Ji90] is, "[The] DFA $\to$ NFA problem, when the input is a unary cyclic DFA, is in NP but not in P unless every language in NP can be accepted by a deterministic Turing machine operating in $n^{O(\log n)}$ time." Whether this problem is NP-complete remains an interesting open question.

Our work has settled most of the fundamental questions in the proposed area of study. Yet there are some questions that are not resolved. Some of them are stated below:

(i) Can the study in §4 be generalized to an arbitrary regularity-preserving operation? Such a study should be able to state qualitatively what makes, for example, the minimum Op equivalence problem easy for Op = rev but hard for Op = $\cdot$.

(ii) Is the Minimum Reverse Generation problem NP-complete when k is presented in binary?

(iii) What is the complexity of the *decision* version of UFA $\to$ DFA? Is it NP-complete? PSPACE-complete? (Clearly, there is an exponential increase in the size when a UFA is converted to a DFA, even when the input is restricted to a unary alphabet [Ra89].)

(iv) What is the complexity of converting DFA to an *approximately* optimal NFA? More specifically, let $n\text{size}(L)$ be the number of states in the minimal NFA accepting L, and let k be a fixed integer. The problem is as follows: Given a DFA M accepting a language L and given an integer k, design an NFA accepting L with at most $(n\text{size}(L)^k)$ states.

Appendix. We present below a list of significant decision problems relevant to this paper. This includes our new results as well as the known problems that we use in our reductions as candidates. We present them in the format used in the appendix of the book by Garey and Johnson [Ga78], which has become quite standard at present for stating decision problems. For each problem we also cite the section and the result where it appears. When presenting the results from other published work, we also cite the original reference, along with its use in our work.

Normal set basis

INSTANCE: Collection C of subsets of a finite set S and positive integer $k \leq |C|$.

QUESTION: Is there a collection B of subsets of S with $|B| \leq k$ such that for each $c \in C$ there is a pairwise disjoint subcollection of B whose union is exactly c?

Remark. The problem is NP-complete, as shown in Lemma 3.3. Stockmeyer [St76] earlier showed that the Set Basis problem is NP-complete.

Universe problem for multiple DFA

INSTANCE: A collection of DFA's $M_1, \ldots, M_n$ over a finite alphabet Σ.

QUESTION: Is $\bigcup_i L(M_i) = \Sigma^*$?

Remark. The problem is PSPACE-complete. The proof readily follows from the next result. This result is used in the proof of Theorem 3.2

Finite automata intersection (Theorem 3.2)

INSTANCE: A collection of DFA's $M_1, \ldots, M_n$ over a finite alphabet Σ.

QUESTION: Is $\bigcap_i L(M_i) = \emptyset$?

Remark. The problem is PSPACE-complete [Ko77]. This result is used in the above problem.

Vertex cover

INSTANCE: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.

QUESTION: Is there a *vertex cover* of size k or less for G, that is, a subset $V' \subseteq V$ such that $|V| \leq k$ and, for each edge $\langle u, v\rangle \in E$ either u or v (or both) belongs to V'?

Remark. The problem is NP-complete [Ka72]. This result is used in the proof of Lemma 3.3.

DFA $\rightarrow$ NFA (UFA)

INSTANCE: A DFA M and an integer k.

QUESTION: Is there an NFA (UFA) with k (or fewer) states accepting $L(M)$?

Remark. The problem is PSPACE-complete, as shown in Theorem 4.2. The problem remains PSPACE-complete when the alphabet is binary. The unary case is partially solved in [Ji90]. In Theorem 4.2 the problem DFA $\rightarrow$ UFA is shown to be NP-complete.

Linear space acceptance

INSTANCE: A linear space-bounded deterministic Turing machine M and a string w.

QUESTION: Is $w \in L(M)$?

Remark. The problem is PSPACE-complete [Ka72]. This result is used to prove Theorem 4.1.

Concatenation equivalence (CE)

INSTANCE: DFA's M_1, M_2, and M_3.

QUESTION: Is $L(M_1) \cdot L(M_2) = L(M_3)$?

Remark. The problem is PSPACE-complete, as shown in Theorem 4.1. This result has been used to derive a PSPACE-completeness result in Theorem 4.4.

Strong concatenation equivalence (SCE)

INSTANCE: DFA's M_1, M_2.

QUESTION: (Is $L(M_1) \cdot L(M_2) = \{x | x \in \Sigma^*, |x| \geq 2\}$?) Is $L(M_1) \cdot L(M_2) = L(M_3)$?

Remark. The problem is PSPACE-complete, which follows by a minor modification of the proof of Theorem 4.1. This result is used to prove a PSPACE-hardness result in Theorem 4.5.

Kleene star equivalence (KSE)

INSTANCE: DFA's M_1 and M_2.

QUESTION: Is $L(M_1)^* = L(M_2)$?

Remark. The problem is PSPACE-complete, as shown in Theorem 4.2. The reduction is from SCE.

Minimum inferred DFA [Go78]

INSTANCE: Finite alphabet Σ, two finite subsets $S, T \subseteq \Sigma^*$, and a positive integer k.

QUESTION: Is there a k-state DFA that accepts a language L such that $S \subseteq L$ and $T \subseteq \Sigma^* - L$?

Remark. The problem is NP-complete [Go78]. This result is used to prove Theorem 4.3.

Minimum Op generation (Op = union, concatenation, Kleene star, or reversal)

INSTANCE: DFA M and an integer k.

QUESTION: Are there DFA's M_1 and M_2 (DFA M_1 if Op is unary) with $|M_1| + |M_2| \leq k$ ($|M_1| \leq k$) such that $L(M_1)\mathrm{Op}L(M_2) = L(M)$? ($\mathrm{Op}(L(M_1)) = L(M)$?)

Remark. The problems are NP-complete for union and intersection, PSPACE-complete for concatenation and Kleene star, and solvable in pseudo-polynomial time for reverse.

Acknowledgments. We are very grateful to Wing Ning Li for once again bringing the DFA $\rightarrow$ NFA problem to our attention. We thank Ed McDowell for carefully reading the manuscript and suggesting improvements and for pointing out an omission in Theorem 4.1. We thank Larry Stockmeyer for sending us the technical report [St76]. We thank the three anonymous referees, whose valuable suggestions improved the presentation of this paper.

REFERENCES

[Ch86] M. CHROBAK, *Finite automata and unary languages*, Theoret. Comput. Sci., 47 (1986), pp. 149–158.

[Co71] S. COOK, *Complexity of theorem proving procedures*, in Proc. 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1971, pp. 151–158.

[Ga78] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to* NP-*Completeness*, Freeman, San Francisco, 1978.

[Gi68] A. GINZBURG, *Algebraic Theory of Automata*, Academic Press, New York, 1968.

[Go78] E. GOLD, *Complexity of automaton identification from given data*, Inform. and Control, 37 (1978), pp. 302–320.

[Ha66] J. HARTMANIS AND R. STEARNS, *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Ho71] J. HOPCROFT, *An n* log *n algorithm for minimizing the states in a finite automation*, in The Theory of Machines and Computations, Z. Kohavi, ed., Academic Press, New York, 1971, pp. 189–196.

[Ho79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[Hu73] H. HUNT, *On the time and tape complexity of languages*, in Proc. 5th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1973, pp. 10–19.

[Hu74] H. HUNT AND D. ROSENKRANTZ, *Computational parallels between regular and context-free languages*, in Proc. 6th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1974, pp. 64–74.

[Hu76] H. HUNT, D. ROSENKRANTZ, AND T. SZYMANSKI, *On the equivalence, containment and covering problems for the regular and context-free languages*, J. Comput. System Sci., 12 (1976), pp. 222–268.

[Ji90] T. JIANG, E. MCDOWELL, AND B. RAVIKUMAR, *The structure and complexity of minimal* NFA*'s over unary alphabet*, Internat. J. Found. Comput. Sci., 2 (1991), pp. 163–182.

[Ka72] R. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, 1972, pp. 85–103.

[Ko77] D. KOZEN, *Lower bounds for natural proof systems*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1977, pp. 254–266.

[Ma78] A. MANDEL AND I. SIMON, *On finite semi-groups of matrices*, Theoret. Comput. Sci., 5 (1978), pp. 183–204.

[Me71] A. MEYER AND M. FISCHER, *Economy of description by automata, grammars and formal systems*, in Proc. 12th Annual IEEE Symposium on Switching and Automata Theory, IEEE Computer Society, Washington, DC, 1971, pp. 188–191.

[Me72] A. MEYER AND L. STOCKMEYER, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proc. 13th Annual IEEE Symposium on Switching and Automata Theory, IEEE Computer Society, Washington, DC, 1972, pp. 125–129.

[My57] J. MYHILL, *Finite automata and the representation of events*, Tech. Report WADD TR-57-624, Wright Patterson AFB, OH, 1957, pp. 112–137.

[Na91] B. NATARAJAN, *Machine Learning, A Theoretical Approach*, Morgan-Kaufmann, San Mateo, CA, 1991.

[Ne58] A. NERODE, *Linear automaton transformations*, Proc. Amer. Math. Soc., 9 (1958), pp. 541–544.

[Pi89] L. PITT AND M. WARMUTH, *The minimum consistent* DFA *problem cannot be approximated within any polynomial*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1989, pp. 421–432.

[Ra59] M. RABIN AND D. SCOTT, *Finite automata and their decision problems*, IBM J. Res. Develop., 3 (1959), pp. 114–125.

[Ra89] B. RAVIKUMAR AND O. IBARRA, *Relating the type of ambiguity to the succinctness of their representations*, SIAM J. Comput., 18 (1989), pp. 1263–1282.

[Re77] C. REUTENAUER, *Propriétés arithmétiques et topologiques de séries rationelles en variables noncommutatives*, Thèse troisième cycle, Universite de Paris VI, Paris, 1977.

[Ri87] R. RIVEST AND R. SCHAPIRE, *Diversity-based inference of finite automata*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1987, pp. 78–87.

[Sa81] A. SALOMAA, *Jewels of Formal Language Theory*, Computer Science Press, New York, 1981.

[Sc88] R. SCHAPIRE, *Diversity Based Inference on Finite Automata*, master's thesis, MIT Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1988; Tech. Report MIT/LCS/TR-413, 1988.

[St85] R. STEARNS AND H. HUNT, *On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata*, SIAM J. Comput., 14 (1985), pp. 598–611.

[St73] L. STOCKMEYER AND A. MEYER, *Word problems requiring exponential time (preliminary report)*, in Proc. 5th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1973, pp. 1–9.

[St76] L. STOCKMEYER, *Set Basis Problem is* NP-*Complete*, Report RC-5431, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1976.

[Th68] K. THOMPSON, *Regular expression searching algorithm*, Comm. ACM, 11 (1968), pp. 419–422.

[Tz89] W. TZENG, *The equivalence and learning of probabilistic automata*, in Proc. 30th Annual IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1989, pp. 268–273.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1142–1162, December 1993

AN $O(m \log n)$-TIME ALGORITHM FOR THE MAXIMAL PLANAR SUBGRAPH PROBLEM*

JIAZHEN CAI[†], XIAOFENG HAN[‡], AND ROBERT E. TARJAN[‡§]

Abstract. Based on a new version of the Hopcroft and Tarjan planarity testing algorithm, this paper develops an $O(m \log n)$-time algorithm to find a maximal planar subgraph.

Key words. algorithm, complexity, depth-first-search, embedding, planar graph, selection tree

AMS subject classifications. 68R10, 68Q35, 94C15

1. Introduction. In [15], Wu defined the problem of planar graphs in terms of the following four subproblems:

P1. *Decide whether a connected graph G is planar.*

P2. *Find a minimal set of edges the removal of which will render the remaining part of G planar.*

P3. *Give a method of embedding G in the plane in case G is planar.*

P4. *Give a description of the totality of possible planar embeddings of G in the plane in case G is planar.*

Linear-time algorithms for P1, P3, and P4 have been known for a long time. The first linear-time solution (which we call the H-T algorithm) for problem P1(the planarity-testing problem) was given by Hopcroft and Tarjan [7] in 1974 using depth-first-search (DFS) trees. In 1976 a P-Q tree solution for P1 based on an earlier algorithm given by Lempel, Even, and Cederbaun [11] was proved to have a linear-time implementation partly by Even and Tarjan [4] and partly by Booth and Lueker [1]. The P-Q tree approach is conceptually simpler, but its implementation is more complicated than that of the H-T algorithm. Linear-time solutions for P3 and P4, also based on P-Q trees, were given by Chiba et al. [2] in 1985.

Wu [15] gave an algebraic solution for all four problems. He proved that a graph is planar if and only if a certain system of linear equations is solvable. In case the graph is planar, an actual embedding can be obtained by considering another system of quadratic equations. His solution is elegant, but his algorithm takes $O(m^2)$ time on an m-edge graph.

Recently, Jayakumar, Thulasiraman, and Swamy [9] studied problem P2 (the maximal planar subgraph problem). For the special case in which a biconnected spanning planar subgraph is given, their algorithm runs in $O(n^2)$ time and $O(mn)$ space on a graph with n vertices and m edges. For more general situations, their algorithm runs in $O(mn)$ time. Their algorithm is also based on P-Q trees. Note that not every biconnected graph has a biconnected spanning planar subgraph (see Fig. 1).

In this paper we give an $O(m \log n)$-time and $O(m)$-space solution to P2. For sparse graphs (i.e., graphs with $m = O(n^{1+\epsilon})$, where $\epsilon < 1$), it beats the algorithm of Jayakumar, Thulasiraman, and Swamy even in the special case when a biconnected spanning planar subgraph is given. Independent of our work, Di Battista and Tamassia [3] claimed an $O(\log n)$-time-per-operation solution to the problem of maintaining a planar graph under edge additions.

*Received by the editors June 5, 1989; accepted for publication (in revised form) April 1, 1992.

[†]Courant Institute, New York University, New York, New York 10012. This work of this author was partly supported by Thomson-CSF/DSE and by National Science Foundation grant CCR90-02428.

[‡]Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

[§]NEC Research Institute, 4 Independence Way, Princeton, New Jersey, 08540. This author's research at Princeton University was partially supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center, grant NSF-STC88-09648, and Office of Naval Research contract N00014-87-K-0467.

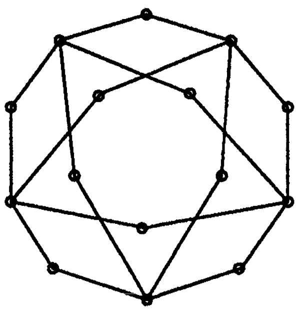


FIG. 1. *A biconnected graph that does not have a biconnected spanning planar subgraph.*

Their algorithm also solves the minimal planar subgraph problem in $O(m \log n)$ time. Their method is much more complicated than ours, however, as it is designed to solve a more general problem. Recently, Kobayashi, Masuda, Kashiwabara [10] showed that if a Hamiltonian tour of the graph is given, then P2 can be solved in linear time. We show that this result can be easily derived from our algorithm as a special case.

The maximal planar subgraph problem is closely related to the planarity-testing problem. In fact, a graph is planar if and only if it is the maximal planar subgraph of itself. Our solution to the maximal planar subgraph problem is based on the H-T algorithm. But for our purpose, we need to modify the algorithm. The main difference is that our version of the algorithm admits a more general ordering than the original H-T algorithm does in processing the successors of each tree edge. Also, the H-T algorithm processes one path at a time, while our algorithm processes one edge at a time. In this sense, our algorithm is a more recursive version of the H-T algorithm.

For the above reason, many of our lemmas and theorems are similar, but not identical, to those in [7]. Instead of referring the readers to [7] for the proofs, we find it more convenient and accurate to supply all main proofs in this paper.

The rest of this paper is organized as follows. Section 2 gives preliminary definitions. Section 3 is a new version of the H-T planarity testing algorithm, which leads to our maximal planar subgraph algorithm in §4. Section 5 is a summary.

2. Preliminaries. Consider an undirected graph $G_0 = (V_0, E_0)$ with edge set E_0 and vertex set V_0. Let $n = |V_0|$ and $m = |E_0|$. We can draw a picture G'_0 of G_0 in the plane as follows: For each vertex $v \in V_0$, we draw a distinct point v'; for each edge $(u, v) \in E_0$, we draw a simple arc connecting the two points u' and v'. We call this arc an *embedding* of the edge (u, v). For brevity, we will sometimes identify graphs with their pictures thus drawn on the plane. If no arcs of G'_0 cross each other, then we call G'_0 a *planar embedding*, or simply an *embedding* of G_0. If G_0 has an embedding, then we say that G_0 is *planar*.

The following facts are important to our discussion.

OBSERVATION 1. *Let C be a simple closed curve in the plane as in Fig.* 2; *let a be a point inside C and b be a point outside C. Then any curve that joins a and b crosses C.*

OBSERVATION 2. *Let G_1 be the undirected graph represented by Fig.* 3, *in which P is a path joining the two vertices a and b on cycle C. Then in any embedding of G_1, all the edges of path P are on the same side of the cycle C* (*either inside or outside*).

OBSERVATION 3. *Let G_2 be the undirected graph represented by Fig.* 4, *in which a_1, a_2, b_1, and b_2 are four distinct vertices that appear in order on C. Then in any embedding of G_2, the two paths P_1 and P_2 are on opposite sides of the cycle C.*

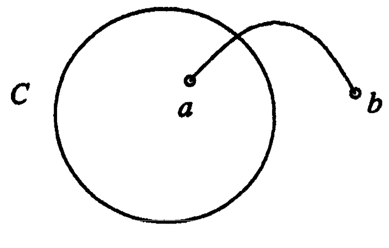


FIG. 2

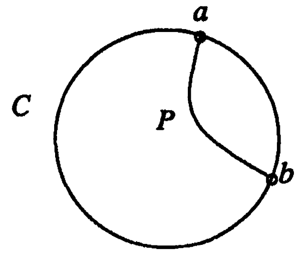


FIG. 3

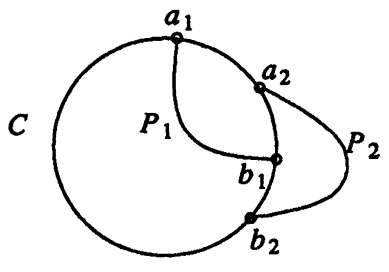


FIG. 4

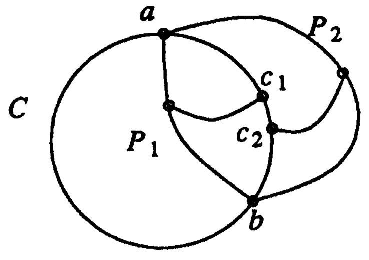


FIG. 5

OBSERVATION 4. *Let G_3 be the undirected graph represented by Fig. 5, in which a, c_1, c_2, and b are vertices that appear in order on C, and c_1 and c_2 may be the same. Then in any embedding of G_3, the two subgraphs P_1 and P_2 are on opposite sides of the cycle C.*

All four observations above are intuitively obvious and follow from the Jordan Curve Theorem [6], [14].

A depth-first-search (DFS) [7] will convert the undirected graph $G_0 = (V_0, E_0)$ into a directed graph $G = (V, T, B)$, where V is the set of DFS numbers of vertices in V_0, T is the set of tree edges, and B is the set of back edges. Each edge of G_0 is converted into either a tree edge or a back edge. All the tree edges form a DFS forest. If $[a, b]$ is a tree edge, then $a < b$. If $[a, b]$ is a back edge, then $b < a$, and there is a tree path in T from b to a. In either case, a is called the *tail* of $[a, b]$, and b is called the *head* of $[a, b]$. The union of T and B will be denoted by E.

For notational convenience, we will frequently identify undirected graphs with their DFS representations. Since we are interested only in graphs with no isolated vertices, we will represent graphs with their edge sets.

We define *successors* for both vertices and edges. If $[a, b]$ is a tree edge, then b is a *successor* of a. If $[a, b]$ is a tree edge and $[b, c]$ is any edge, then $[b, c]$ is a *successor* of $[a, b]$. Back edges have no successors. We also define descendants and ancestors for both vertices and edges. A *descendant* of vertex (respectively, edge) x is defined recursively as either x itself or a successor of a descendant of x. If y is a descendant of x, then x is an *ancestor* of y.

Let $e = [a, b] \in E$. Let Y be the set of vertices y such that for some x, $[x, y]$ is a back edge and also a descendant of e. If Y is not empty, we define $low_1(e)$ to be the smallest integer in Y, and $low_2(e)$ to be the second smallest integer in $Y \cup \{n + 1\}$. Otherwise, we define $low_1(e) = low_2(e) = n + 1$. The two mappings low_1 and low_2 can be computed in $O(m)$ time during the depth-first-search on G_0 [7]. If a is not the root of a DFS tree, and $low_1(e) \geq a$, then a is an articulation point of G [12].

We define the function ϕ on E as follows. If $e = [a, b]$ is any edge in E, then,

$$\phi(e) = \begin{cases} 2\, low_1(e) & \text{if } low_2(e) \geq a \\ 2\, low_1(e) + 1 & \text{otherwise.} \end{cases}$$

We arrange the successors of each tree edge in increasing order on their ϕ values. This ordering can be computed in $O(m)$ time using a bucket sort [7]. If $e_1, \ldots, e_k$ are the successors of e ordered this way, we will call e_i the ith successor of e for $i = 1 \ldots k$.

As in [7], for $e = [a, b]$, we define $S(e)$, the *segment* of e, to be the subgraph of G that consists of all the descendants of e. We use $ATT(e)$ to denote the set of back edges $[c, d]$ in $S(e)$ such that d is an ancestor of a, including a itself. Each back edge in $ATT(e)$ is called an *attachment* of e.

For any edge $e = [a, b]$, we define $cycle(e)$ as follows: If e is a back edge, then $cycle(e) = \{e\} \cup \{e' : e'$ belongs to the tree path from b to $a\}$; if e is a tree edge and $low_1(e) > a$, then $cycle(e) = \{\}$; otherwise, $cycle(e) = cycle(e_1)$, where e_1 is the first successor of e. We use $sub(e)$ to denote the subgraph $S(e) \cup cycle(e)$. It is easy to see that if $cycle(e)$ is not empty, then the vertex $low_1(e)$ is always on $cycle(e)$. Also, if $low_1(e) \geq a$, then $sub(e) = S(e)$; if $low_1(e) < a$, then $sub(e) - S(e) = \{e' : e'$ belongs to the tree path from $low_1(e)$ to $a\}$.

Figure 6 illustrates some of these definitions, where $low_1(e) = 1$; $low_2(e) = 2$; $cycle(e) = \{[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 1]\}$; $S(e)$ contains all the edges in the graph except $[1, 2], [2, 3], [3, 4]$; $sub(e)$ is the whole graph; $ATT(e) = \{[8, 1], [9, 3], [12, 1], [14, 2], [13, 4]\}$.

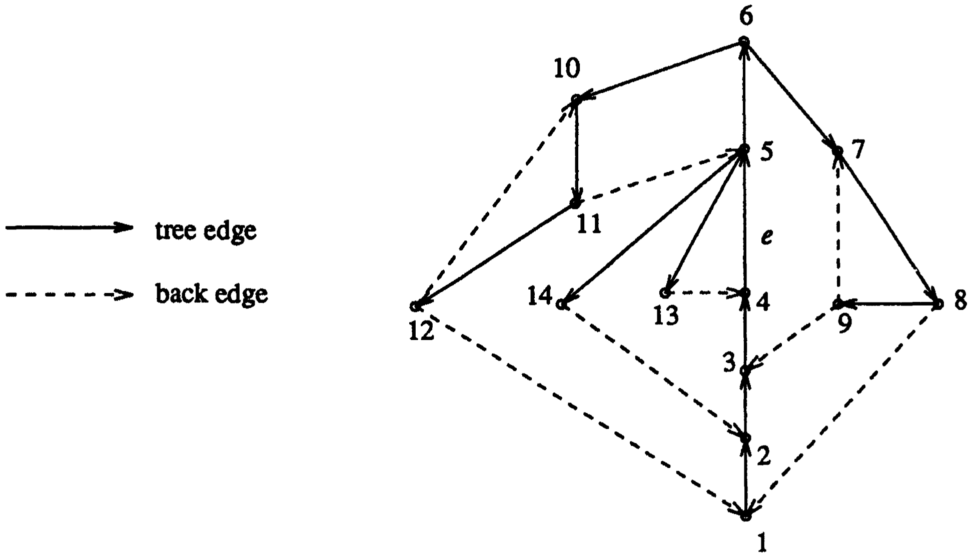


FIG. 6

3. Planarity testing. As explained in [7], a graph is planar if and only if each of its biconnected components is planar. Also, a graph of one edge is always planar. Thus, we need only consider how to test the planarity of biconnected graphs with more than one edge. Let $G = (V, T, B)$ be a DFS representation of such a graph. Then T forms a single tree with only one tree edge leaving the root. Call this tree edge e_0. Since $sub(e_0)$ is the whole graph, we can determine the planarity of G with a procedure that can determine the planarity of $sub(e)$ for all e in G.

We say that an edge e is planar if $sub(e)$ is planar. To determine the planarity of an edge e, we consider two cases. If e is a back edge, then $sub(e) = cycle(e)$, which is always planar. Otherwise e is a tree edge having at least one successor. In this case we first determine the planarity of each of its successors. If all these successors are planar, then we determine the planarity of e based on the structure of its attachments. The details follow.

3.1. Structure of attachments. The planarity of an edge $e = [a, b]$ directly depends on the structure of its attachments. Since we assume that G is a biconnected graph with more than one edge, then $low_1(e) \leq a$, and both $ATT(e)$ and $cycle(e)$ are not empty. If e is planar, then we can partition the edges of $ATT(e)$ into *blocks* as follows. We put two back edges of $ATT(e)$ in the same block if they are on the same side of $cycle(e)$ in every embedding of $sub(e)$. Two blocks B_1 and B_2 of $ATT(e)$ *interlace* if they are on opposite sides of $cycle(e)$ in every embedding of $sub(e)$. Each block B_i of $ATT(e)$ can interlace at most one other block, since two attachments of e that cannot be embedded on the same side of $cycle(e)$ as B_i must be in the same block.

The back edge on $cycle(e)$ is the only attachment of e that will not be embedded on either side of $cycle(e)$. By convention, this back edge forms a block by itself, called the *singular block* of e, which does not interlace other blocks of $ATT(e)$.

In Fig. 6, $ATT(e)$ consists of four blocks: $B_1 = \{[8, 1]\}$, $B_2 = \{[12, 1], [14, 2]\}$, $B_3 = \{[9, 3]\}$, and $B_4 = \{[13, 4]\}$. B_1 is singular. B_2 and B_3 are interlacing.

If $e' = [u, v]$ is an attachment of e, then $low_1(e) \leq v \leq a$. If $low_1(e) < v < a$, then we say that e' is *normal*. Otherwise we say that e' is *special*. A block of attachments of e is *normal* if it contains some normal attachment of e. Otherwise we say that it is *special*. In Fig. 6, B_1 and B_4 are special, and other blocks are all normal. We say that $sub(e)$ is *strongly planar* with respect to e if e is planar and if all the normal blocks of $ATT(e)$ can be embedded on the same side of $cycle(e)$. If $sub(e)$ is strongly planar (with respect to e), then we say that e is strongly planar. We have the following lemma.

LEMMA 1. *Let e_i be the ith successor of e, where $i > 1$. Then e_i is strongly planar if and only if the subgraph $S(e_i) \cup cycle(e)$ is planar.*

Proof. $\Rightarrow$ If e_i is strongly planar, then there is an embedding of $sub(e_i)$ such that all its normal blocks are on the same side of $cycle(e_i)$. Thus we can add $cycle(e)$ to the other side of $cycle(e_i)$ to get an embedding of $S(e_i) \cup cycle(e)$.

$\Leftarrow$ If $S(e_i) \cup cycle(e)$ is planar, then in any embedding of $S(e_i) \cup cycle(e)$, all the normal blocks of $ATT(e_i)$ must be on the same side of $cycle(e_i)$. □

Note that in an embedding of $S(e_i) \cup cycle(e)$, the special blocks of e_i do not have to be on the same side of $cycle(e_i)$. (See Fig. 7.)

We will represent a block of back edges $H = \{[b_1, a_1], [b_2, a_2], \ldots, [b_t, a_t]\}$ by a list $K = [a_1, a_2, \ldots, a_t]$, where $a_1 \leq a_2 \leq \cdots \leq a_t$. Frequently, we will identify blocks with their list representations. Define $first(H) = first(K) = a_1$, and $last(H) = last(K) = a_t$. If K is empty, we define $first(H) = first(K) = n + 1$, and $last(H) = last(K) = 0$. We can further organize the blocks of $ATT(e)$ as follows: If two blocks X and Y interlace, we put them into a pair $[X, Y]$, assuming $last(X) \geq last(Y)$; if a nonempty block X does not interlace any other block, we form a pair $[X, [\ ]]$. Let $[X_1, Y_1]$ and $[X_2, Y_2]$ be two pairs of interlacing blocks. We say $[X_1, Y_1] \leq [X_2, Y_2]$ if and only if $last(X_1) \leq \min(first(X_2), first(Y_2))$.

We say a list of interlacing pairs $[q_1, \ldots, q_s]$ is *well ordered* if $q_1 \leq \ldots \leq q_s$. Empty lists or lists of one pair are well ordered by convention. We will see that all the interlacing pairs of $ATT(e)$ can be organized into a well-ordered list $[p_1, \ldots, p_t]$. We call this list $att(e)$.

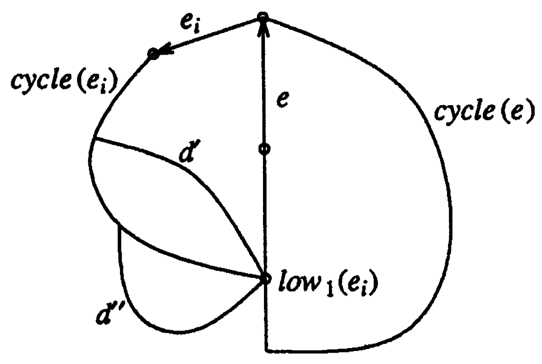


FIG. 7. *The two special attachments d' and d'' of e_i can be on different sides of* cycle(e_i), *although they are on the same side of* cycle(e).

In Fig. 6, $att(e) = [p_1, p_2, p_3]$, where $p_1 = [[1], [\ ]]$, $p_2 = [[3], [1, 2]]$, and $p_3 = [[4], [\ ]]$.

3.2. Computing *att(e)*. Now we are ready to compute $att(e)$. The planarity of e will be decided at the same time.

Consider any edge $e = [a, b]$. If e is a back edge, then its only attachment is e itself. Therefore $att(e) = [[[b], [\ ]]]$. Otherwise, let $e_1, \ldots, e_k$ be the successors of e in increasing order by their ϕ values. We first recursively compute $att(e_i)$ for each successor e_i of e. Then we compute $att(e)$ in the following four steps.

ALGORITHM 1.

Step 1. For $i = 1 \ldots k$, delete all occurrences of b appearing in blocks within $att(e_i)$. Because these occurrences appear together at the end of the blocks that are contained in the last pairs of $att(e_i)$ only, a simple list traversal suffices to delete all these occurrences in time $O(k+$ *number of deletions*$)$. After this, initialize $att(e)$ to be $att(e_1)$.

Step 2. For $i = 2, \ldots k$, merge all the blocks of $att(e_i)$ into one intermediate block B_i. See Fig. 8.

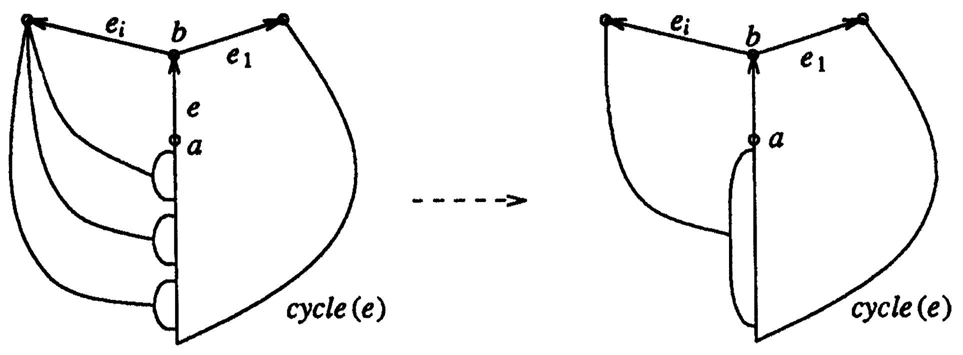


FIG. 8

According to Lemma 1, this step can be done for a given value of i only if the normal blocks of $att(e_i)$ do not interlace. (If a pair of normal blocks of $att(e_i)$ interlace, the graph is not planar, and the computation fails.) To merge the blocks for a given value of i, we traverse the list of pairs $att(e_i)$, concatenating blocks to form B_i. Initially, B_i is empty. To process pair $[X, Y]$, if X and Y are both normal, the computation fails, since the graph is not planar. Otherwise, we concatenate Y and X onto the end of B_i in order and continue. In case Y is a special block, we know that Y has only attachments ending in $low_1(e_i)$, since b was deleted from all blocks of $att(e_i)$ in Step 1. Thus the correct ordering of attachments is maintained by this process. This step takes $O(1 + \textit{number of blocks in } att(e_i))$, resulting in one block for each i.

Step 3. Merge blocks in $att(e)$. (See Fig. 9.)

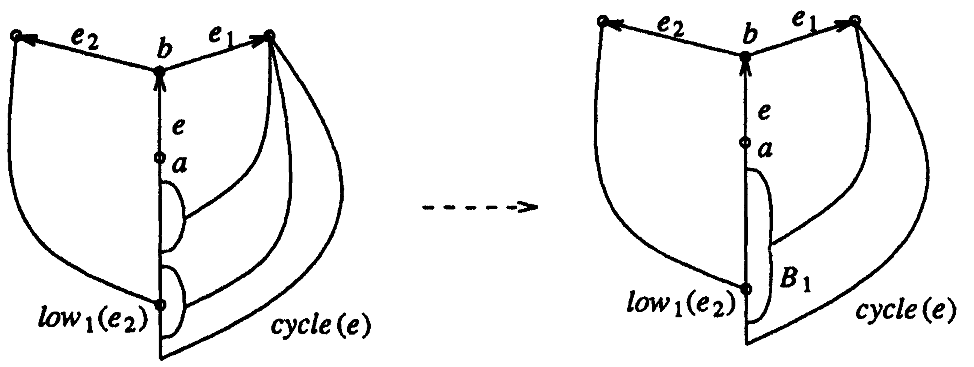


FIG. 9

By Observation 3, all blocks D in $att(e)$ with $last(D) > low_1(e_2)$ must be merged into one block B_1. (If any two of these blocks interlace, the graph is not planar, and the computation fails.) This is achieved by merging from the high end of $att(e)$. The time is $O(1+$ *reduction in number of blocks*). This step turns $att(e)$ into a list of pairs $p_1 \leq \cdots \leq p_h$ with only p_h possibly having a block D with $last(D) > low_1(e_2)$. Note that $low_1(e_2)$ is the lowest among the vertices $low_1(e_2), \dots, low_1(e_k)$.

Step 4. For $i = 2, \dots, k$, add blocks B_i into $att(e)$.

To process B_i, consider the highest pair $P : [X, Y]$ of $att(e)$. Consider three subcases:

i. If B_i cannot be embedded on either side of $cycle(e)$, then the computation of $att(e)$ fails.

ii. If B_i interlaces X only, then merge B_i into Y by concatenating their ordered list representations. Next, switch X and Y if $last(X) < last(Y)$.

iii. If B_i interlaces neither X nor Y, then add $[B_i, [\ ]]$ to the high end of $att(e)$; $P := [B_i, [\ ]]$.

By Lemma 2, testing whether B_i interlaces X or Y takes $O(1)$ time. Also by this lemma, it is not possible that B_i interlaces Y only, since $last(X) \geq last(Y)$. (See Fig. 10.) □

LEMMA 2. *In Step* 4, B_i *and* D *can be embedded on the same side of* $cycle(e)$ *if and only if* $low_1(e_i) \geq last(D)$, *where* $D = X$ *or* $D = Y$.

Proof. $\Rightarrow$ Assume $low_1(e_i) < last(D)$. Then there must be a path P_1 in $S(e_i)$ from b to $low_1(e_i)$ containing some back edge in B_i, and another path P_2 in $S(e_1) \cup \dots \cup S(e_{i-1})$ from

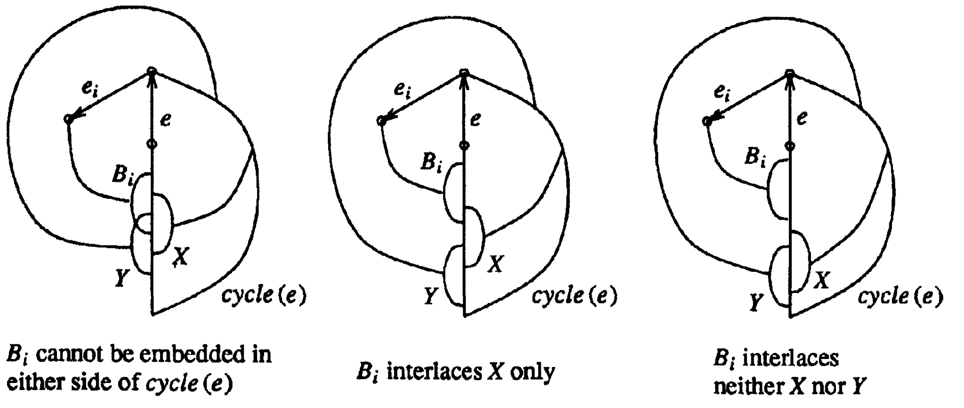

e_i
e
B_i
X
Y
$cycle(e)$
B_i cannot be embedded in either side of $cycle(e)$
B_i interlaces X only
B_i interlaces neither X nor Y

FIG. 10. (a) B_i *cannot be embedded in either side of cycle* (e); (b) B_i *interlaces* X *only*; (c) B_i *interlaces neither* X *nor* Y.

a vertex w on $cycle(e)$ to $last(D)$ containing some back edge in D but no edge on $cycle(e)$. We consider two cases (see Fig. 11). If $w > b$ then, by Observation 3, P_1 and P_2 cannot be embedded on the same side of $cycle(e)$. If $w = b$, then the first edge on P_2 is e_j for some $1 < j < i$, which implies $i > 2$ and $\phi(e_j) \leq \phi(e_i)$. Consequently, $low_1(e_j)low_2(e_j) < b \leq low_1(e_i) < last(D) < b$, which implies that $low_2(e_j) < b$. If $low_1(e_j) < low_1(e_i)$, then there must be an undirected simple path P_3 between $last(D)$ and $low_1(e_j)$ containing back edges in D but no edges on $cycle(e)$. By Observation 3 again, P_1 and P_3 cannot be embedded on the same side of $cycle(e)$. If $low_1(e_j) = low_1(e_i)$, then $low_2(e_i) < b$ (recall $low_2(e_j) < b$). By Observation 4, $S(e_i)$ and $S(e_j)$ cannot be embedded on the same side of $cycle(e)$. All of the above cases imply that B_i and D cannot be embedded on the same side of $cycle(e)$.

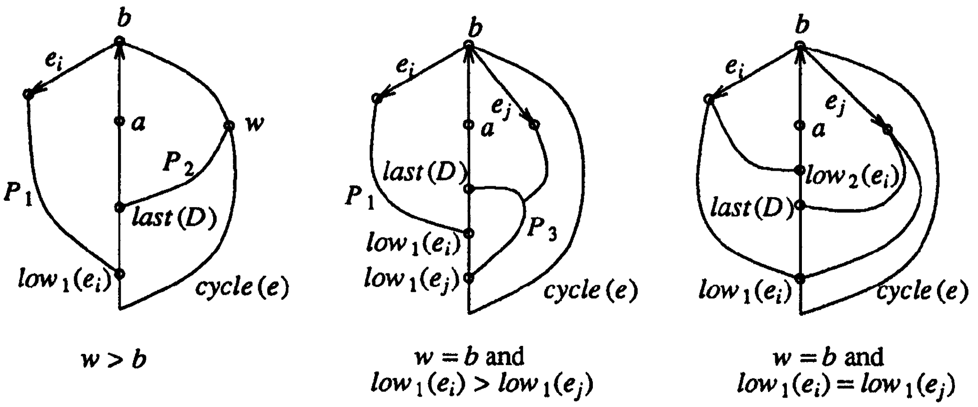


FIG. 11. (a) $w > b$; (b) $w = b$ *and* $low_1(e_i) > low_1(e_j)$; (c) $w = b$ *and* $low_1(e_i) = low_1(e_j)$.

$\Leftarrow$ See the proof of Lemma 4 in the next section. □

THEOREM 1. (i) *Algorithm* 1 *computes* $att(e)$ *successfully if and only if* e *is planar.* (ii) *If* e *is planar, then Algorithm* 1 *computes* $att(e)$ *correctly.*

Proof. For the proof see the next section. □

3.3. Correctness. In the following proofs, unless stated otherwise, we will use $att(e)$ to mean the list $att(e)$ computed by Algorithm 1. But we will prove that this $att(e)$ correctly implements the $att(e)$ defined in §3.1.

During the presentation of Algorithm 1, we explained that two nonempty blocks form a pair within $att(e)$ only if they cannot be embedded on the same side of $cycle(e)$, and the computation of $att(e)$ fails only when e is not planar. Also, we can see that the singular block of e is not merged with any other block. To prove Theorem 1, we still have to show that the following assertions are true:

(1) If computation of $att(e)$ succeeds, then e is planar.

(2) If any two nonempty, nonsingular blocks of $att(e)$ do not form a pair, then these blocks can be embedded on the same side as well as on different sides of $cycle(e)$.

(3) $att(e)$ is well ordered.

We first prove (3), using the following lemma.

LEMMA 3. *The list of pairs* $att(e)$ *computed by Algorithm* 1 *is well ordered.*

Proof. We prove this lemma by induction on the number of descendants of e. If e has no successor, then e is a back edge, and the lemma is trivially true. Now assume that e is a tree edge with successors $e_1, \dots, e_k$ in increasing order by ϕ value, and that $att(e_1), \dots, att(e_k)$ are all well ordered. After Steps 1 and 2 are executed, $att(e_1), \dots, att(e_k)$ are still well ordered. Thus, $att(e)$ is well ordered when it is initialized to $att(e_1)$. In Step 3, only blocks in the highest pairs of $att(e)$ are merged, and therefore $att(e)$ is still well ordered after the merge. Then consider the moment in Step 4 just before B_i is added to $att(e)$. Assume $att(e)$ is well ordered at this moment. Let $P : [X, Y]$ be the last pair of $att(e)$. We need only consider the two cases in which the computation does not fail.

1. $last(Y) \le low_1(B_i) < last(X)$. Then B_i is merged with Y. If P is the only pair in $att(e)$, then $att(e)$ is well ordered by definition. Otherwise, let $Q : [X_1, Y_1]$ be the pair next to P. Then we have $Q \le P$ before merge. We need only to show that this is still true after the merge. If $i = 2$, then Step 3 guarantees that $first(B_2) = low_1(e_2) \ge \max(last(X_1), last(Y_1))$. If $i > 2$, then B_{i-1} is contained in either X or Y. Since $first(B_i) = low_1(e_i) \ge low_1(e_{i-1}) = first(B_{i-1}) \ge \min(first(X), first(Y))$, then merging B_i into Y does not change the value of $\min(first(X), first(Y))$. Thus, after merging Y and B_i, we still have $Q \le P$.

2. $low_1(B_i) \ge last(X)$. Then $[B_i, [\,]]$ becomes the last pair of $att(e)$. Since $last(X) \le low_1(e_i) = first(B_i)$ in this case, we have $P \le [B_i, [\,]]$.

Thus, $att(e)$ is still well ordered after each B_i is added, $i = 2 \dots k$. Therefore $att(e)$ is well ordered after Step 4. $\square$

Next we prove the *if* part of Lemma 2: In Step 4 of Algorithm 1, if $low_1(e_i) \ge last(D)$, then B_i and D can be embedded on the same side of $cycle(e)$, where $D = X$ or $D = Y$.

Proof. Consider an embedding of $sub(e)$ before B_i is added. Let att_i be the current value of $att(e)$. Assume, without loss of generality, that X is embedded on the left-hand side of $cycle(e)$ in this embedding. Let $h = last(X)$. Then $h = \max\{last(Z) : Z \text{ is a block in } att_i\}$, and there is a face F on the left $cycle(e)$ in the current embedding of $sub(e)$ such that the tree path from h to b is on the boundary of F. See Fig. 12. Thus, if $low_1(B_i) \ge last(X)$, then B_i can be embedded in F. Similarly, let $h' = \max\{last(U) : U \text{ is a block in } att_i \text{ embedded on the right-hand side of } cycle(e)\}$. Then there is a face F' on the right of $cycle(e)$ in the current embedding of $sub(e)$ such that the tree path from h' to b is on the boundary of F'. According to the proof of Lemma 3, if $low_1(B_i) \ge last(Y)$, then $low_1(B_i) \ge h'$ also. Therefore B_i can be embedded in F' in this case. $\square$

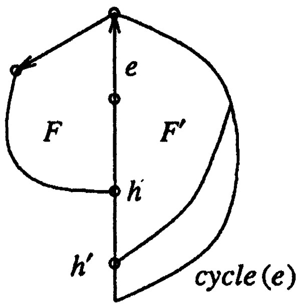


FIG. 12

Then we prove the following lemma that implies the assertions (1) and (2). We say that a set W of blocks of $att(e)$ is *consistent* with respect to e if for all $X, Y \in W$, neither $[X, Y]$ nor $[Y, X]$ is in $att(e)$.

LEMMA 4. *If Algorithm* 1 *does not fail, and* D_1 *and* D_2 *are two disjoint consistent sets of nonsingular blocks from* $att(e)$, *then there is an embedding of* $sub(e)$ *such that blocks of* D_1 *are on one side of* $cycle(e)$ *and blocks of* D_2 *are on the other side of* $cycle(e)$.

Proof. The lemma is trivially true if e has no successors. If e has successors, let $e_1, \ldots, e_k$ be the list of successors of e in increasing order by their ϕ values. Assume that the lemma holds for each of these successors. We want to construct an embedding of $sub(e)$ such that D_1 and D_2 are embedded on different sides of $cycle(e)$.

If W is a set of blocks, then a *W-attachment* is an attachment contained in some block of W. For $j = 1, 2$, let $D_j' = \{X : [X, Y]$ or $[Y, X]$ is a pair in $att(e)$ and $Y \in D_j\}$. Let $H_1 = \{X : ([X, Y]$ is a pair in $att(e))$ and (X and Y are not in $D_1 \cup D_2)\}$, and let $H_2 = \{Y : [X, Y]$ is a pair in $att(e)$ and (X and Y are not in $D_1 \cup D_2)\}$. Let $C_1 = D_1 \cup D_2' \cup H_1$, and $C_2 = D_2 \cup D_1' \cup H_2$. For $j = 1, 2$, let $K_j = \{X : X$ is a block in $sub(e_1)$ containing some C_j-attachment$\}$. Then K_1 and K_2 are two disjoint consistent subsets of blocks of $sub(e_1)$.

Initially, we construct an embedding of $sub(e_1)$ such that K_1 and K_2 are on different sides of $cycle(e_1)$. As a result, those C_1-attachments and C_2-attachments contained in $sub(e_1)$ are on different sides of $cycle(e)$ (which is $cycle(e_1)$). This embedding exists by the induction hypothesis. Take this embedding to be the initial embedding of $att(e)$. Then for $i = 2, \ldots, k$, we add $sub(e_i)$ to this embedding one by one as follows.

Since the normal blocks of $att(e_i)$ do not interlace, we can, by induction, find an embedding of $sub(e_i)$ such that all of its normal blocks are embedded on the same side of $cycle(e_i)$. We call this embedding E_i, and its mirror image E_i'. Let B_i, P, and $[X, Y]$ be the same as in Step 4 of Algorithm 1. Assume, without loss of generality, that X is embedded on the left-hand side of $cycle(e)$. Consider the following two cases.

Case 1. $last(Y) \leq low_1(e_i) < last(X)$. Then B_i is merged with Y. According to Lemma 2, one of E_i or E_i' can be embedded on the right of $cycle(e)$. If B_i contains any C_1-attachment, then X contains some C_2-attachment; if B_i contains any C_2-attachment, then X contains some C_1-attachment. In any case, C_1-attachments and C_2-attachments are still on different sides of $cycle(e)$ after B_i is embedded.

Case 2. $low_1(e_i) \geq last(X)$. Again by Lemma 2, one of E_i or E_i', say, E_i, can be embedded on the left of $cycle(e)$, and the other, E_i', can be embedded on the right of $cycle(e)$. One of these two choices will result in an embedding such that C_1-attachments are on one side of $cycle(e)$, and C_2-attachments are on the other side.

When all the B_i's are added, we get an embedding of $sub(e)$ such that D_1 is on one side of $cycle(e)$, and D_2 is on the other side of $cycle(e)$. This is true because every D_1-attachment is a C_1-attachment, and every D_2-attachment is a C_2-attachment. □

This completes the proof of Theorem 1, and establishes that the list $att(e)$ computed by Algorithm 1 has the properties discussed at the end of §3.1.

Let e, e_i, B_i, att_i, X, Y, and h' be the same as in the above proofs. Let $L = \{Z : [Z, U]$ is a pair in $att_i\}$, $R = \{U : [Z, U]$ is a pair in $att_i\}$, and $h_2 = \{last(U) : U$ is a block in $R\}$. Then it is easy to see that $h' \geq h_2$. According to Lemma 4, there exists an embedding of $sub(e)$ (before adding B_i) such that L is embedded in one side of $cycle(e)$ and R is embedded in the other side. According to the proof of Lemma 2, $low_1(e_i) \geq last(Y)$ if and only if $low_1(e_i) \geq h_2$. Therefore we also have the following corollary.

COROLLARY 1.1. *B_i cannot be embedded in either side of $cycle(e)$ if and only if $low_1(e_i) < h_2$.*

Corollary 1.1 gives a test of whether B_i can be added to att_i without referring to the top pair of att_i. This is useful in our maximal planar subgraph algorithm, where we need to test whether B_2 can be added to $att(e)$ before Step 3 is performed.

3.4. Data structure and running time. As suggested in [7], we can implement blocks as linked lists. An interlacing pair of blocks can be represented as a record containing two pointers to the two linked lists representing these two blocks. Then $att(e)$ can be represented as a linked list of such records. In this way, the time cost for Step 1 is $O(k+$ *number of deletions*$)$. The cost for Steps 2, 3, and 4 is $O(k+$ *reduction in number of blocks*$)$. The cumulate expense of executing these steps over the whole graph is $O(m)$. The initial DFS in which low_1 values are computed takes time $O(m)$. Arranging the successors in increasing order by ϕ value for all tree edges takes $O(m)$ time using a bucket sort. Thus the whole algorithm runs in $O(m)$ time. It is well known that any $O(m)$-time algorithm for planarity testing can be implemented in $O(n)$ time, since $m = O(n)$ for a planar graph [7].

3.5. A modification to Algorithm 1. Consider Step 4 of Algorithm 1. Lemma 2 requires that the successors of each tree edge be ordered by ϕ values. Maintaining this ordering causes difficulties in solving the maximal planar subgraph problem. Fortunately, we can modify Algorithm 1 so that it requires only the low_1 ordering of the successors of each tree edge.

Let $e = [a, b]$ be a tree edge, and $e_1, \ldots, e_k$ be the list of its successors in increasing order by low_1 values. Still define $cycle(e) = cycle(e_1)$. Then Steps 1, 2, and 3 can be performed with respect to this ordering without any modification.

Next we want to merge $B_2, \ldots, B_k$ into $att(e)$ in that order. In general, successors ordered by low_1 values may not be ordered by ϕ values. Consequently, there may be some $1 < i \leq k$ such that $\phi(e_{i-1}) > \phi(e_i)$. But if this happens, we know that $low_1(e_{i-1}) = low_1(e_i)$ and $low_2(e_i) \geq b$. If $i = 2$, Lemma 2 still applies, and we can merge B_2 into $att(e)$ as before. Otherwise, the following lemma says that we do not have to merge B_i into $att(e)$ at all.

LEMMA 5. *If for some $2 < i \leq k$, $low_1(e_{i-1}) = low_1(e_i)$, $low_2(e_i) \geq b$, and e_i is planar, then G is planar if and only if $G - S(e_i)$ is planar.*

Proof. The *only if* part is trivial, so we just prove the *if* part. Consider an embedding E_i of $G_i = G - S(e_i)$. Under the condition of the lemma, e_i has no normal attachments. Since e_i is planar, then e_i is strongly planar. Also, b and $low_1(e_i)$ are the only two vertices shared by $S(e_i)$ and G_i. Therefore $S(e_i)$ can be embedded in any face of E_i whose boundary contains the two vertices b and $low_1(e_i)$.

Let P be the tree path $cycle(e_{i-1}) \cap cycle(e)$ and let C be the closed curve $cycle(e_{i-1}) \cup cycle(e) - P$. Then C contains edges from both $S(e_{i-1})$ and $G_i - S(e_{i-1})$. By Observation 2, P is on one side of C. Call this side of C S_1, and the other side S_2. Let U be the set of faces in S_2 whose boundaries contain edges from $S(e_{i-1})$ only, and let W be the set of faces in S_2 whose boundaries contain edges from $G_i - S(e_{i-1})$ only. Then faces in U and faces in W do not share common boundaries. Thus, within S_2 there must be some face F whose boundary contains edges from both $S(e_{i-1})$ and $G_i - S(e_{i-1})$, and therefore contains at least two vertices common to $S(e_{i-1})$ and $G_i - S(e_{i-1})$. But all the vertices common to $S(e_{i-1})$ and $G_i - S(e_{i-1})$ are on P, and among them only b and $low_1(e_{i-1})$ are on the boundary of S_2. Therefore these two vertices must be on the boundary of F. Thus we can embed $S(e_i)$ in F to get an embedding of G. □

Therefore, under the conditions of Lemma 5, in deciding the planarity of G, we can ignore its subgraph $S(e_i)$. Since the condition $low_1(e_{i-1}) = low_1(e_i)$ and $low_2(e_i) \geq b$ is implied by $low_1(e_{i-1}) \leq low_1(e_i)$ and $\phi(e_{i-1}) > \phi(e_i)$, we can modify Step 4 as follows.

Step 4$'$. Add blocks $B_2, \ldots, B_k$ into $att(e)$ in that order, assuming $low_1(e_1) \leq low_1(e_2) \leq \cdots \leq low_1(e_k)$. Initially, let $j = 1$ and $i = 2$. To process B_i, we consider two cases. If $j = 1$ or $\phi(e_j) \leq \phi(e_i)$, we do the same thing as in Step 4, and then let $j = i$; otherwise, we do nothing.

The list $att(e)$ computed by the modified algorithm may not contain all the attachments of e. Some attachments may be omitted by Step 4$'$, because their existence does not affect the planarity of the whole graph G.

4. The maximal planar subgraph problem.

Now we consider the maximal planar subgraph problem: Find a minimal set of edges whose deletion results in a planar graph. The resulting graph is called a *maximal planar subgraph* of G. We can always find a maximal planar subgraph of G by deleting back edges only, since all the tree edges form a forest, which is planar.

We will not assume that the input graph is biconnected, since deletion of back edges may turn a biconnected graph into a graph with articulation points. But, without loss of generality, we can assume that the input graph is connected. Thus the tree edges of G form a single tree with root r. Let $t_1, \ldots, t_s$ be the tree edges leaving the root. If $s = 1$, then $sub(t_1)$ is the whole graph G. If $s > 1$, then r is the only vertex common to $sub(t_1), \ldots, sub(t_s)$. Thus, to find a maximal planar subgraph of G, we can just find a maximal planar subgraph for each of the subgraphs $sub(t_1), \ldots, sub(t_s)$, and then simply put these subgraphs together. Therefore, what we need is a procedure that can find a maximal planar subgraph of $sub(e)$ for any given edge e of G.

4.1. Maximal *l-planar* subgraphs.

We cannot build a maximal planar subgraph of $sub(e)$ by simply putting together the maximal planar subgraphs of $sub(e_1), \ldots, sub(e_k)$, and deleting those back edges causing failure in Algorithm 1. The reason is that, after these edges are deleted, it may turn out that some other edges, which we deleted for making $sub(e_1), \ldots, sub(e_k)$ planar, would not have had to be deleted at all. We avoid this difficult situation by constructing such maximal subgraphs $S_1, \ldots, S_k$ of $sub(e_1), \ldots, sub(e_k)$ so that they can be used to construct a planar subgraph S of $sub(e)$ without further deletion of edges. Two measures are taken for this purpose. First, those back edges in $sub(e_i)$ that can cause failure in Steps 3 or 4 of Algorithm 1 are deleted before a maximal subgraph of S_i is recursively computed. Second, the information where blocks of $sub(e_i)$ are allowed to interlace is passed to the recursive call that computes S_i, so that when the returned S_i is merged to $sub(e)$. Step 2 of Algorithm 1 can also be performed successfully without deletion. Since the planar subgraph S of $sub(e)$ computed by our algorithm may be used to build a larger planar subgraph of G in the same way as we use $S_1, S_2, \ldots, S_k$ to build S, we also need to know where in S blocks are allowed to interlace. This approach leads naturally to the concept of *l-planar subgraphs,* which is a generalization of the concept of strongly planar subgraphs.

Consider an edge $e = [a, b]$ and a vertex l on the tree path from $low_1(e)$ to a. An attachment $[u, v]$ of e is *l-normal* if $low_1(e) < v < l$. A block of attachments is *l-normal* if it contains some l-normal attachment. Let D be the list representation of a nonempty block of attachments. Define $second(D)$ to be the second smallest element in the set $\{x : x \in D\} \cup \{n + 1\}$, and define $second([\ ]) = n + 1$. Then D is l-normal if and only if $low_1(e) < first(D) < l$ or $second(D) < l$. The two mappings *first* and *second* can be maintained during the computation of $att(e)$ in $O(1)$ time for each modification to $att(e)$.

We say that the subgraph $sub(e)$ is *l-planar* if e is planar and the l-normal blocks of $att(e)$ do not interlace (cf. Fig. 13, where (a) is l-planar, but (b) is not). Edge e is *l-planar* if $sub(e)$ is l-planar.

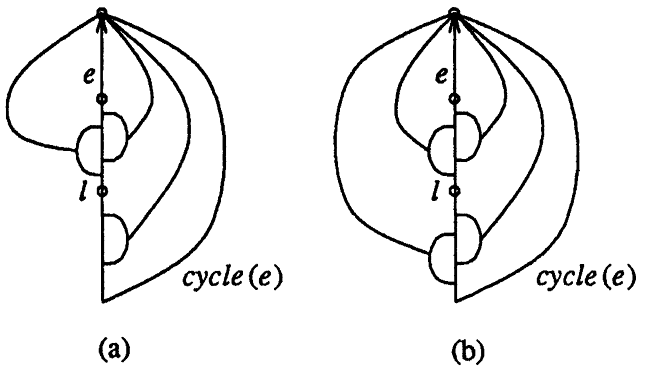


FIG. 13

If H is a subgraph obtained from $sub(e)$ by deleting back edges only, then we can define the l-planarity for H (with respect to e) in the same way as we did for $sub(e)$. We will talk about l-planar subgraphs of $sub(e)$ in this sense. An l-planar subgraph of $sub(e)$ is *maximal* if it can be obtained from $sub(e)$ by deleting a minimal set of back edges.

Consider edge $e = [a, b]$. According to our definition, e is planar if and only if e is $low_1(e)$-planar, and e is strongly planar if and only if e is a-planar. Therefore, if we can find a maximal l-planar subgraph of $sub(e)$ for any l with $low_1(e) \leq l \leq a$, then we can compute a maximal planar subgraph of $sub(e)$.

The following is an outline of our maximal l-planar subgraph algorithm, where l is a given integer with $low_1(e) \leq l \leq a$ and remains fixed during the processing of an edge. Let $e = [a, b]$, and consider three cases:

Case 1. e is a back edge. Assign $[[[b], [\ ]]]$ to $att(e)$, and return.

Case 2. e is a tree edge with no successors. Assign $[\ ]$ to $att(e)$, and return.

Case 3. e is a tree edge with successors $e_1, \ldots, e_k$, among which e_1 has the smallest low_1 value. We construct a sequence $G_1, \ldots, G_k$ of l-planar subgraphs of $sub(e)$ such that G_1 is a maximal l-planar subgraph of $sub(e_1)$ and $low_1(e_1)$ remains unchanged; G_k is a maximal l-planar subgraph of $sub(e)$; and each g_i, $1 < i \leq k$, is obtained from G_{i-1} by adding to it a strongly planar subgraph S_i of $sub(e_i)$, where e_i is some successor of e not contained in G_{i-1}. During the construction, we compute $att(e)$ using the modified version of Algorithm 1. We describe below in rough terms how we compute S_i.

```
1   select an edge $e_i$ with the smallest $low_1$ value from successors of $e$ not contained
            in $G_{i-1}$;
2   **while** there exists a maximal strongly planar subgraph of $sub(e_i)$ whose addition
            to $G_{i-1}$ destroys its $l$-planarity **do**
```

```
3           delete some attachments from $sub(e_i)$;
4           if the deletion changes the $low_1$ value of $e_i$ then
5                select a possibly new edge $e_i$ with the smallest $low_1$ value from
                      successors of $e$ not contained in $G_{i-1}$;
6           end if;
7      od;
8      recursively construct a maximal strongly planar subgraph of $sub(e_i)$ without
       changing $low_1(e_i)$ further. We take this subgraph as $S_i$.
```

In the procedure sketched above, lines 1, 4, 5, 6, and 8 guarantee that subgraphs S_i are generated in increasing order by new low_1 values of the corresponding successors. For each $1 < i \leq k$, once S_i is computed, no edges will be deleted further from it. There are still two questions remaining to be answered: how the testing in line 2 can be done without constructing a maximal strongly planar subgraph of $sub(e_i)$, and how the attachments are chosen so that the deletion in line 3 makes the set of deleted edges minimal. These two questions are closely related and will be explained together in the next section.

Remark. In Algorithm 1, we do not need the concept of l-planarity, since our purpose is to check the planarity of G. If some interlacing blocks of $sub(e)$ are found not to fit in the whole graph after returning from several levels of recursive calls, we simply declare that the graph is not planar. But if we want to construct a maximal planar subgraph of G, then it is too late to delete edges efficiently by that time. Therefore we use the parameter l to pass the information where blocks of $sub(e)$ are allowed to interlace, to the recursive calls, so that the correct edges are already deleted during the processing of $sub(e)$.

The need to generalize to l-planarity arises in the following way in the algorithm sketched above. To compute a maximal planar subgraph of the input graph, the recursive calls that construct S_i for $i = 2, \ldots, k$ must construct maximal strongly planar graphs. Within one of these recursive calls, the initial second-level recursive call (to construct a maximal b-planar subgraph of $sub(e_{i1})$, where e_{i1} is the first successor of edge e_i) and more deeply nested recursive calls of the same kind construct maximal l-planar subgraphs for general values of l.

4.2. Algorithm for deleting back edges. Let $e = [a, b]$, and consider the **while** loop in the procedure sketched above. If $low_1(e_i) \geq b$, then b is the only vertex common to $sub(e_i)$ and $G - sub(e_i)$. In this case, we can apply the maximal planar subgraph to $sub(e_i)$ separately, and do not have to consider the effect on the whole graph. Next we consider the case when $low_1(e_i) < b$. Assume that $sub(e_i)$ is made strongly planar by deleting some back edges. Suppose that the low_1 value and the low_2 value of e_i are not changed by these deletions. We want to see whether the union of $sub(e_i)$ and G_{i-1} is l-planar.

As in planarity testing, let B_i be the block of attachments obtained by merging $att(e_i)$; let att_i be the current value of $att(e)$; let B_j be the last block merged into att_i by Step 4′; let $h_1 = \max\{last(Z) : [Z, U] \text{ is a pair in } att_i\}$ and $h_2 = \max\{last(U) : [Z, U] \text{ is a pair in } att_i\}$. (Initially, we set $j = 1$ and $att(e) = att(e_1)$ after removing all the occurrences of b from $att(e_1)$.) Finally, let $h_3 = \max\{last(Z) : Z \text{ is an } l\text{-normal block of } att_i\}$.

The two variables h_1 and h_2 can be maintained in $O(1)$ time per modification to $att(e)$ by maintaining two lists L and R (as suggested in [7]), where L is the ordered list of nonempty blocks X such that $[X, Y]$ is a pair in $att(e)$, and R is the ordered list of nonempty blocks Y such that $[X, Y]$ is a pair in $att(e)$. If B_L and B_R are the highest blocks of L and R, respectively, then $h_1 = last(B_L)$ and $h_2 = last(B_R)$. Lists L and R also let h_3 be maintained easily. If e is a back edge, $h_3 = 0$ from its definition. If e is a tree edge, we get the initial value of h_3 from the computation of $att(e_1)$, and modify it in $O(1)$ time for each modification of $att(e)$. The details will not be discussed here.

By Lemma 5, in case that e_i is strongly planar, $sub(e_i)$ can affect the planarity of G only if any of the following conditions holds:

(a) $j = 1$, i.e., no block B_j has been merged into $att(e)$ yet,

(b) $low_1(e_j) < low_1(e_i)$, or

(c) $low_2(e_i) < b$.

If any of these conditions is true, we consider two additional cases:

1. The union of $sub(e_i)$ and G_{i-1} is not planar. By Corollary 1.1, this happens if and only if $low_1(e_i) < h_2$.

2. The union of $sub(e_i)$ and G_{i-1} is planar, but not l-planar. Then B_i is l-normal, and it interlaces an l-normal block of att_i. We know that B_i is l-normal if and only if $low_1(e_1) < low_1(e_i) < l$ or $low_2(e_i) < l$. Also, it is easy to see that B_i interlaces an l-normal block of att_i if and only if $low_1(e_i) < h_3$.

This means, under the conditions (a), (b), or (c), that the union of $sub(e_i)$ and G_{i-1} is not l-planar if and only if any of the following conditions holds.

(i) $low_1(e_i) < h_2$.

(ii) $low_1(e_1) < low_1(e_i) < \min\{h_3, l\}$.

(iii) $low_2(e_i) < l$ and $low_1(e_i) < h_3$.

Therefore, if any of the conditions (i), (ii), and (iii) holds, some back edge has to be deleted to change either the low_1 value or the low_2 value of e_i. Such testing and deletion can be done even before making $sub(e_i)$ strongly planar. For this purpose, we combine the above conditions ((a) or (b) or (c)) and ((i) or (ii) or (iii)) into two groups according to whether they involve $low_2(e_i)$:

Condition AA.

($j = 1$ and $low_1(e_i) < h_2$), or

($low_1(e_j) < low_1(e_i) < h_2$), or

($low_1(e_j) < low_1(e_i) < \min\{h_3, l\}$).

Condition BB.

($low_2(e_i) < b$ and $low_1(e_i) < h_2$), or

($low_2(e_i) < b$ and $low_1(e_1) < low_1(e_i) < \min\{h_3, l\}$), or

($low_2(e_i) < l$ and $low_1(e_i) < h_3$).

It can be checked that condition ((a) or (b) or (c) and (i) or (ii) or (iii)) is equivalent to condition (AA or BB).

If Condition AA is true, we can make it false only by changing the value $low_1(e_i)$. In this case, we delete all the back edges of $sub(e_i)$ entering the vertex $low_1(e_i)$. After the deletion, we choose a possibly new e_i with the smallest low_1 value.

If Condition AA is false, then we test Condition BB. If the result is true, we know that $low_1(e_i) = low_1(e_j)$. This is because $low_1(e_i) < low_2(e_i)$, which means that BB implies that $low_1(e_i) < h_2$ or $low_1(e_i) < \min\{h_3, l\}$, from which it follows that $low_1(e_j) = low_1(e_i)$; otherwise AA would be true. (We have $low_1(e_j) \leq low_1(e_i)$ by the ordering of the successors of e.) To make Condition BB false, we can change the value of either $low_1(e_i)$ or $low_2(e_i)$. If we choose to change $low_2(e_i)$ consistently, then at least one of the back edges $[u, v]$ of $sub(e_i)$ with $v = low_1(e_i)$ will survive. But if we choose to change $low_1(e_i)$, it may happen that all the attachments in $ATT(e_i) \cap ATT(e)$ are eventually deleted and that the resulting graph is not maximal. Therefore, in this case we choose to delete all the back edges $[u, v]$ of $sub(e_i)$ with $v = low_2(e_i)$ (see Fig. 14).

We test and delete repeatedly as described above until we find an edge e_i that does not satisfy AA or BB. Then we can construct S_i recursively from $sub(e_i)$ and merge it into G_{i-1}.

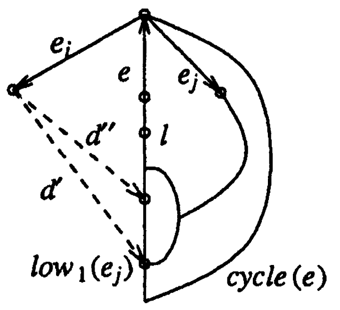


FIG. 14. *The edge e_i satisfies condition* BB. *If we choose to delete d', then d'' will also be deleted later because of Condition* AA, *and the resulting graph will not be maximal.*

Since no edge is added to $sub(e_i)$ during the construction of S_i, conditions AA and BB remain false after the construction. Thus, the resulting graph G_i will be planar, and no l-normal blocks will interlace.

To see that the deleted set of back edges is minimal, let $[u, v]$ be an edge deleted by the above algorithm, and add it back to G_i. If $[u, v]$ was deleted because of Condition AA, then $low_1(e_i) = v$ now, and Condition AA is true again. If $[u, v]$ was deleted because of Condition BB, then $low_2(e_i) = v$ now, and Condition BB is true again. Notice that, in the latter case, the low_1 value of e_i has remained unchanged since the deletion of $[u, v]$. In either case, G_i will not be l-planar.

4.3. Data structures and running time. In the algorithm described above, we need to select repeatedly an unprocessed successor of e with the smallest low_1 value, and the low_1 values of tree edges are constantly changing. Therefore we maintain a heap [13] based on low_1 values of the unprocessed successors of the tree edge e currently being processed. Since the algorithm is recursive, we actually maintain simultaneously a heap of unprocessed successors for each tree edge along the path to the currently active tree edge. The total size of all such heaps is $O(m)$. The initialization of all these heaps takes a total of $O(m)$ time. When the low_1 value of some element in a heap increases, we modify the heap accordingly. It is important to note that any two edges in active heaps are unrelated; thus deletion of a single attachment can modify the low_1 value of only a single such edge. It follows that the total number of modifications to and deletions from heaps is $O(m)$. The time for the heap operations is $O(\log n)$ time per operation, for a total of $O(m \log n)$ time. (Since $m < n^2$, $\log m = O(\log n)$.)

We also need a data structure for the back edges of $sub(e)$ so that the following operations can be done efficiently:

1. delete an attachment $[u, v]$ of e with $v = low_1(e)$ or $v = low_2(e)$;
2. maintain the low_1 and low_2 values of e;
3. split the data structure into several pieces, one for each successor of e.

One easy solution that meets these requirements is the *selection tree* [8]. To represent a set of edges E_0 as a selection tree T_0, we store edges of E_0 inside the leaves of T_0 from left to right in increasing order (by DFS number) of their tails. Edges with the same tail are ordered arbitrarily. Each internal node w of T_0 has two children $w.lchild$ and $w.rchild$. Let S_w be the set of edges stored in the leaves of the subtree rooted at w; let $lb = \min\{x : [x, y] \in S_w\}$ and $rb = \max\{x : [x, y] \in S_w\}$; let $low_1 = \min\{y : [x, y] \in S_w\}$, and $low_2 = \min(\{y : [x, y] \in S_w | y \neq low_1\} \cup \{n + 1\})$. Then the four values lb, rb, low_1, and low_2 are stored in the four fields $w.lb$, $w.rb$, $w.low_1$, and $w.low_2$ of w, respectively. If w is a leaf storing the edge $[x, y]$, then $w.lb = x$, $w.rb = x$, $w.low_1 = y$, and $w.low_2 = n + 1$. The values in each internal node can be computed from the values in the children (in constant time).

In the following discussion, we will refer to a tree by its root. Let r_1 and r_2 be two selection trees representing the two disjoint sets of edges E_1 and E_2. If $u_1 \leq u_2$ for all $[u_1, v_1] \in E_1$ and $[u_2, v_2] \in E_2$, then we can merge r_1 and r_2 to get the selection tree for $E_1 \cup E_2$ in $O(1)$ time:

```
procedure $merge(r_1, r_2)$;
begin if $r_1 = null$ then
            return $r_2$;
      end if;
      if $r_2 = null$ then
            return $r_1$;
      end if;
      $r := newnode()$;
      $r.lchild := r_1$;
      $r.rchild := r_2$;
      $r.lb := r_1.lb$;
      $r.rb := r_2.rb$;
      $r.low_1 := \min\{r_1.low_1, r_2.low_1\}$;
      $r.low_2 := \min(\{r_1.low_1, r_2.low_1, r_1.low_2, r_2.low_2\} - \{r.low_1\})$;
      return $r$;
end;
```

Let r be a selection tree representing a set of edges E_0. To split E_0 into two sets $E_1 = \{[u, v] \in E_0 | u \leq u_x\}$ and $E_2 = \{[u, v] \in E_0 | u > u_x\}$, we split r with respect to u_x as follows:

```
procedure $split(r, u_x)$;
begin if $u_x < r.lb$ then
            return $[null, r]$;
      elseif $u_x \geq r.rb$ then
            return $[r, null]$;
      else  $[r_l, r_r] := [r.lchild, r.rchild]$;
            if $u_x < r_l.rb$ then
                  $[r_{l1}, r_{l2}] := split(r_l, u_x)$;
                  return $[r_{l1}, merge(r_{l2}, r_r)]$;
            else  $[r_{r1}, r_{r2}] := split(r_r, u_x)$;
                  return $[merge(r_l, r_{r1}), r_{r2}]$;
            end if;
      end if;
end;
```

The height of any tree that results from splitting a tree r can be no greater than the height of r. To select and delete an edge $[x, v]$ from a tree r, where $v \in \{r.low_1, r.low_2\}$, we do the following:

```
procedure $delete(r, v)$;
begin if $r$ is a leaf then
            mark the back edge stored in $r$ as 'deleted';
            return $null$;
      else  $[r_l, r_r] := [r.lchild, r.rchild]$;
            if $v = r_l.low_1$ or $v = r_l.low_2$ then
```

```
                return $merge(delete(r_l, v), r_r)$;
            else    return $merge(r_l$, delete$(r_r, v))$;
            end if;
        end if;
    end;
```

Assuming the input graph G is connected, we know that all the tree edges form a tree. Let the root be one. For technical reasons, we add a dummy edge $e_0 = [0, 1]$ to the tree edges. To get a maximal planar subgraph of G, we just construct a 0-planar subgraph of $sub(e_0)$, and then delete e_0 from it. Initially, we construct a balanced selection tree $tree(e_0)$ to store all the back edges of G. The height of this tree is $O(\log n)$. The time and space needed to initialize $tree(e_0)$ are both $O(m)$.

When we begin to construct a maximal l-planar subgraph for a tree edge e, we first split $tree(e)$ into several pieces $tree(e_1), \ldots, tree(e_k)$, where $e_1, \ldots, e_k$ are the successors of e not marked as 'deleted.' For each such successor e_i, $tree(e_i)$ is a selection tree representing the set of back edges in $sub(e_i)$, and can be obtained as follows. If e_i is a back edge, then $tree(e_i)$ consists of a single edge, and can be constructed in $O(1)$ time. If e_i is a tree edge, let $e_i = [b, c_i]$, and let n_i be the number of descendants of c_i. It is well known that a back edge $[u, v]$ is a descendant of e_i if and only if $c_i \leq u < c_i + n_i$. Then we can use the procedure *split* to get $tree(e_i)$ from $tree(e)$ in $O(\log n)$ time. Since there are at most n tree edges in G, the splitting takes $O(n \log n)$ time for the whole algorithm. After each split, the total size of the trees is still $O(m)$.

To select and delete an attachment $[x, v]$ of e_i, where $v \in \{low_1(e_i), low_2(e_i)\}$, we execute $delete(tree(e_i), v)$, which takes $O(\log(n))$ time. There can be at most $O(m)$ such invocations of $delete$, so the total cost for executing $delete$ is $O(m \log n)$. Given the selection tree $tree(e_i)$, the values $\phi(e_i)$, $low_1(e_i)$, and $low_2(e_i)$ can be computed from $tree(e_i)$ in $O(1)$ time: If $tree(e_i)$ is null, we just set these values to $n + 1$; otherwise, they can be computed from $tree(e_i).low_1$ and $tree(e_i).low_2$. Thus, the total cost of selection tree operations is $O(m \log n)$.

We have mentioned that the total cost of heap operations is also $O(m \log n)$. The other costs of the algorithm are the same as in planarity testing. Thus the total cost of our maximal planar subgraph algorithm is $O(m \log n)$.

4.4. The complete algorithm. Now we summarize our maximal planar subgraph algorithm. We take a connected undirected graph as input, and convert it into a DFS representation $G = (V, T, B)$. At the same time, we compute the two mappings $succ$ and N, where, for each $e \in T$, $succ(e)$ gives the successor edges of e in increasing order of their heads, and for each $v \in V$, $N(v)$ gives the number of descendants of v. We assume that there is a dummy edge $e_0 = [0, 1]$ such that $succ(e_0)$ gives the list of tree edges leaving the root. The whole preprocessing takes $O(m)$ time.

We summarize the maximal l-planar subgraph algorithm below.

```
procedure $lplanar(e, l)$;
begin let $e = [a, b]$;
    if $e \in B$ then
            return [[[$b$], []]];
    end if;
    if $e$ has no successors then
            return [];
    end if;
    let $e_1, \ldots, e_k$ be the successors of $e$ not marked as 'deleted';
```

```
      split $tree(e)$ into $tree(e_1), \ldots, tree(e_k)$;
      organize $e_1, \ldots, e_k$ into a heap based on their $low_1$ values, with the smallest one on the top;
      let $e_1$ be the edge on the top of the heap;
      delete $e_1$ from the heap;
1     $att(e) := lplanar(e_1, l)$;
      delete all the occurrences of $b$ from the top blocks of $att(e)$;
      $j := 1$;
      $i := 2$;
      while heap is not empty do
          let $e_i$ be the edge on the top of the heap;
          if $low_1(e_i) \geq b$ then
              delete $e_i$ from the heap;
2             $dummy := lplanar(e_i, b)$;
          elseif Condition AA is true then
              $v := tree(e_i).low_1$;
              while $v = tree(e_i).low_1$ do
3                 $tree(e_i) := delete(tree(e_i), v)$;
              end while;
              if $e_i$ is a back edge then
                  delete $e_i$ from heap;
              else modify heap;
              end if;
          elseif Condition BB is true then
              $v := tree(e_i).low_2$;
              while $v = tree(e_i).low_2$ do
4                 $tree(e_i) := delete(\text{tree}(e_i), v)$;
              end while;
          else  delete $e_i$ from the heap;
5             $att(e_i) := lplanar(e_i, b)$;
6             merge blocks of $att(e_i)$ into one block $B_i$;
              delete all the occurrences of $b$ from the top blocks of $att(e_i)$;
              if $i = 2$ then
7                 perform Step 3 of Algorithm 1;
              end if;
8             merge $B_i$ into $att(e)$ as described in Step 4′;
              $i := i + 1$;
          end if
      end while;
      return $att(e)$;
  end;
```

The procedure $lplanar(e, l)$ implicitly constructs a maximal l-planar subgraph of $sub(e)$ by deleting a minimal set of back edges. The parameter l specifies where the blocks of $att(e)$ are not allowed to interlace in the resulting subgraph, so that this subgraph can be used to build a larger planar subgraph without further deletion when we process the predecessor of e. For the initial call where $e = e_0$, we have $l = 0$, meaning that we need to construct a maximal planar subgraph of $sub(e_0)$. In the recursive calls for the successors of e, the l values are determined as follows. Since no l-normal blocks are allowed to interlace in $sub(e)$, then no l-normal blocks are allowed to interlace in $sub(e_1)$ either. Thus the recursive call of $lplanar$ for e_1 (line 1) has the same parameter l as for the edge e. The remaining calls for

$e_2, \ldots, e_k$ (line 5) just construct maximal strongly planar subgraphs, therefore they have b as their l values. Thus when we merge blocks at line 6, no normal blocks of $att(e_i)$ interlace. At line 7, we merge all normal blocks of $att(e)$ above $low_1(e_2)$ into one block; at line 8, we merge B_i into $sub(e)$. Because of the deletions at lines 3 and 4, these steps can be performed successfully (without any further deletion). At line 2, $sub(e_i)$ is detected to be a biconnected component and is processed separately.

To compute a maximal planar subgraph, we simply do the following:

1. Organize B into a selection tree $tree(e_0)$;
2. Execute $lplanar(e_0, 0)$.

Then $T \cup B - B'$ gives a maximal planar subgraph of G, where B' is the set of back edges deleted by the procedure *delete* in the preceding algorithm.

Remark. The procedure *lplanar* can be greatly simplified if we know that all the tree edges of G are on a same cycle. In this case, the low_1 values need not be dynamically maintained: if $e = [a, b]$ is a back edge, then $low_1(e) = b$; otherwise $low_1(e) = 1$. The low_2 values, which are used only for testing condition BB when $i > 1$, need not be maintained either, since for $i > 1$, e_i is a back edge. As a result, the selection trees are no longer useful, and the heaps storing the successors of tree edges can be replaced by lists precomputed as in Algorithm 1. With these simplifications, our algorithm gives the following result, which is first reported in [10] by Kobayashi, Masuda, and Kashiwabara: A maximal planar subgraph can be constructed in linear time provided that a Hamiltonian tour of the graph is given. □

5. Summary. The problem of drawing graphs in the plane arises naturally in circuit layout. Since finding a maximum planar subgraph is NP-complete [5], a maximal planar subgraph seems to be a reasonable approximation. Because planarity testing can be done in linear time, it is easy to solve the maximal planar subgraph problem in $O(mn)$ time: Start with a graph H with no edge; for each edge of the input graph G, add it to H if the resulting graph is planar, and reject it otherwise. The resulting graph H will be a maximal planar subgraph of G. However, a better solution seemed to be hard to find for a long time. Jayakumar, Thulasiraman, and Swamy [9] even made the conjecture that "no maximal planarization algorithm of complexity better than $O(mn)$ will be possible." Our $O(m \log n)$ solution disproves this conjecture, as does the method of Di Battista and Tamassia [3].

We have assumed that the input graph to our algorithm is connected. For a more general graph, we can find a maximal planar subgraph by applying our algorithm to each of its connected components.

Acknowledgment. We thank the referees for their careful and invaluable comments on the earlier versions of this paper.

REFERENCES

[1] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using* PQ-*tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[2] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa, *A linear algorithm for embedding planar graphs using* PQ-*trees*, J. Comput. System Sci., 30 (1985), pp. 54–76.

[3] G. Di Battista and R. Tamassia, *Incremental planarity testing (extended abstract)*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 436–441.

[4] S. Even and R. E. Tarjan, *Computing an st-numbering*, Theoret. Comput. Sci., 2 (1976), pp. 339–344.

[5] M. Garey and D. Johnson, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[6] D. Hall and G. Spencer, *Elementary Topology*, John Wiley, New York, 1955.

[7] J. Hopcroft and R. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.

[8] E. Horowitz and S. Sahni, in Fundamentals of Data Structure, Computer Science Press, Rockville, MD, 1983.

[9] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy, *$O(n^2)$ algorithms for graph planarization*, IEEE Trans. CAD, 8 (1989), pp. 257–267.

[10] N. Kobayashi, S. Masuda, and T. Kashiwabara, *Algorithms to obtain a maximal planar Hamilton subgraph*, IEICE Transactions E74, 4 (1991), pp. 657–664.

[11] A. Lempel, S. Even, and I. Cederbaun, *An algorithm for planarity testing of graphs*, in Theory of Graphs, International Symposium, Rome, Italy, July, 1966, pp. 215–232.

[12] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[13] ———, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1984.

[14] W. T. Thron, *Introduction to the Theory of Functions of a Complex Variable*, John Wiley, New York, 1953.

[15] W. Wu, *On the planar imbedding of linear graphs*, J. Systems Sci. Math. Sci., 5 (1985), pp. 290–302.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1163–1175, December 1993

ON THE EXISTENCE OF PSEUDORANDOM GENERATORS*

ODED GOLDREICH[†], HUGO KRAWCZYK[‡], AND MICHAEL LUBY[§]

Abstract. Pseudorandom generators (suggested and developed by Blum and Micali and Yao) are efficient deterministic programs that expand a randomly selected k-bit seed into a much longer pseudorandom bit sequence that is indistinguishable in polynomial time from an (equally long) sequence of unbiased coin tosses. A fundamental question is to find simple conditions, as the existence of one-way functions, which suffice for constructing pseudorandom generators. This paper considers *regular* functions, in which every image of a k-bit string has the same number of preimages of length k. This paper shows how to construct pseudorandom generators from any regular one-way function.

Key words. pseudorandom generators, one-way functions, cryptography, randomness, complexity theory

AMS subject classifications. 11K45, 11T71, 68Q99, 94A60

1. Introduction.

In recent years, randomness has become a central notion in the theory of computation. It is used heavily in the design of sequential, parallel, and distributed algorithms, and is, of course, crucial to cryptography. Once so frequently used, randomness itself has become a resource and economizing on the amount of randomness required for an application has become a natural concern. It is in this light that the notion of pseudorandom generators was first suggested and the following fundamental result was derived: the number of coin tosses used in any practical application (modeled by a polynomial time computation) can be decreased to an arbitrarily small power of the input length.

The key to the above informal statement is the notion of a pseudorandom generator suggested and developed by Blum and Micali [BM] and Yao [Y]. A *pseudorandom generator* is a deterministic polynomial time algorithm that expands short seeds into longer bit sequences, such that the output ensemble is polynomially indistinguishable from the uniform probability distribution. More specifically, the generator (denoted G) expands a k-bit seed into a longer, say $2k$-bit, sequence so that for every polynomial time algorithm (distinguishing test) T, any constant $c > 0$, and sufficiently large k

$$|Prob[T(G(X_k)) = 1] - Prob[T(X_{2k}) = 1]| \leq k^{-c},$$

where X_m is a random variable assuming as values strings of length m, with uniform probability distribution. It follows that the strings output by a pseudorandom generator G can substitute the unbiased coin tosses used by any polynomial time algorithm A, without changing the behavior of algorithm A in any noticeable fashion. This yields an equivalent polynomial time algorithm, A', which randomly selects a seed, uses G to expand it to the desired amount, and then runs A using the output of the generator as the random source required by A. The theory

*Received by the editors April 20, 1989; accepted for publication (in revised form) July 1, 1992. A preliminary version of this paper was presented at the IEEE 29th Annual Symposium on Foundations of Computer Science, 1988.

[†]Department of Computer Science, Technion, Haifa 32000, Israel. This author was supported by grant 86-00301 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

[‡]Department of Computer Science, Technion, Haifa 32000, Israel. Present address, IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, New York 10598.

[§]International Computer Science Institute, 1947 Center Street, Berkeley, California 94704-1105, and Department of Mathematics, University of California, Berkeley, California 94720. A large portion of the author's research contribution to this work was done while visiting the Computer Science Department of the Technion. At the time, the author was a faculty member at the University of Toronto. This author's research was partially supported by Natural Sciences and Engineering Research Council of Canada operating grant A8092, by a University of Toronto grant, by National Science Foundation grant CCR-9016468, and by grant 89-00312 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

of pseudorandomness was further developed to deal with function generators, and permutation generators, and additional important applications to cryptography have emerged [GGM], [LR]. The existence of such seemingly stronger generators was reduced to the existence of pseudorandom (string) generators.

In light of their practical and theoretical value, constructing pseudorandom generators and investigating the possibility of such constructions is of major importance. A necessary condition for the existence of pseudorandom generators is the existence of one-way functions (since the generator itself constitutes a one-way function). On the other hand, stronger versions of the one-wayness condition were shown to be sufficient. Before reviewing these results, let us recall the definition of a one-way function.

DEFINITION 1. A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is called *one-way* if it is polynomial time computable, but not "polynomial time invertible." Namely, there exists a constant $c > 0$ such that for *any* probabilistic polynomial time algorithm A, and sufficiently large k

$$Prob\left[A(f(x), 1^k) \notin f^{-1}(f(x))\right] > k^{-c}, \ (*),$$

where the probability is taken over all x's of length k and the internal coin tosses of A, with uniform probability distribution.

(*Remark.* The role of 1^k in the above definition is to allow Algorithm A to run for time polynomial in the length of the preimage it is supposed to find. Otherwise, any function that shrinks the input by more than a polynomial amount would be considered one-way.)

1.1. Previous results. The first pseudorandom generator was constructed and proved valid by Blum and Micali, under the assumption that the discrete logarithm problem is intractable on a nonnegligible fraction of the instances [BM]. In other words, it was assumed that exponentiation modulo a prime (i.e., the 1-1 mapping of the triple (p, g, x) to the triple $(p, g, g^x \bmod p)$, where p is prime and g is a primitive element in Z_p^*), is one-way. Assuming the intractability of factoring integers of the form $N = p \cdot q$, where p and q are primes and $p \equiv q \equiv 3 \bmod 4$, a simple pseudorandom generator exists [BBS], [ACGS].[1] Under this assumption the permutation, defined over the quadratic residues by modular squaring, is one-way.

Yao has presented a much more general condition, which suffices for the existence of pseudorandom generators; namely, the existence of one-way permutations [Y].[2]

Levin has weakened Yao's condition, presenting a necessary and sufficient condition for the existence of pseudorandom generators [L]. Levin's condition, hereafter referred to as *one-way on iterates*, can be derived from Definition 1 by substituting the following line instead of line(*)

$$(\forall i, 1 \le i < k^{c+2}) Prob\left[A(f^{(i)}(x), 1^k) \notin f^{-1}(f^{(i)}(x))\right] > k^{-c},$$

where $f^{(i)}(x)$ denotes f iteratively applied i times on x (as before the probability is taken uniformly over all x's of length k). Clearly, any one-way permutation is one-way on its iterates. It is also easy to use any pseudorandom generator in order to construct a function that satisfies Levin's condition.

Levin's condition for the construction of pseudorandom generators is somewhat cumbersome. In particular, it seems hard to test the plausibility of the assumption that a particular

[1]A slightly more general result, concerning integers with all prime divisors congruent to 3 mod 4, also holds [CGG].

[2]In fact, Yao's condition is slightly more general. He requires that f is 1-1 and that there exists a probability ensemble Π, which is invariant under the application of f and that inverting f is "hard on the average" when the input is chosen according to Π.

function is one-way on its iterates. Furthermore, it has been an open question whether or not Levin's condition is equivalent to the mere existence of one-way functions.

1.2. Our results. In this paper, we consider "regular" functions, in which every element in the range has the same number of preimages. We show how to construct pseudorandom generators from any regular one-way function.

DEFINITION 2. A function f is called *regular* if there is a function $m(\cdot)$ such that for every n and for every $x \in \{0, 1\}^n$ the cardinality of $f^{-1}(f(x)) \cap \{0, 1\}^n$ is $m(n)$.

Clearly, *every* 1-1 function is regular (with $m(n) = 1, \forall n$). Our main result is the following theorem.

MAIN THEOREM. *If there exists a regular one-way function, then there exists a pseudorandom generator.*

A special case of interest is of 1-1 one-way functions. The sufficiency of these functions for constructing pseudorandom generators does not follow from previous works. In particular, Yao's result concerning one-way permutations does not extend to 1-1 one-way functions.

Regularity appears to be a simpler condition than the intractability of inverting on the function's iterates. Furthermore, many natural functions (e.g., squaring modulo an integer) are regular and thus, using our result, a pseudorandom generator can be constructed assuming that any of these functions is one-way. In particular, if factoring is weakly intractable (i.e., every polynomial time factoring algorithm fails on a nonnegligible fraction of the integers) then pseudorandom generators do exist. This result was not known before. (It was only known that the intractability of factoring a special subset of the integers implies the existence of a pseudorandom generator.) Using our results, we can construct pseudorandom generators based on the (widely believed) conjecture that decoding random linear codes is intractable, and on the assumed average case difficulty of combinatorial problems as subset-sum.

The main theorem is proved essentially by transforming any given regular one-way function into a function that is one-way on its iterates (and then applying Levin's result [L]).

It is interesting to note that not every (regular) one-way function is "one-way on its iterates." To emphasize this point, we show (in Appendix A) that from a (regular) one-way function we can construct a (regular) one-way function, which is easy to invert on the distribution obtained by applying the function *twice*. The novelty of this work is in presenting *a direct way to construct a function that is one-way on its iterates from any regular one-way function* (*which is not necessarily one-way on its iterates*).

1.3. Subsequent results. Recent results of Impagliazzo, Levin, and Luby [ILL] and Hastad [H], inspired by the current work, has resolved the problem of equivalence of existence of one-way functions and pseudorandom generators, in the affirmative. However, in light of the inefficiency of their construction, some of the ideas presented in the current work may be useful in future attempts to construct more efficient pseudorandom generators from one-way functions.

2. Main result.

2.0. Preliminaries. In the sequel, we make use of the following definition of *strongly* one-way function. (When referring to Definition 1, we shall call the function *weak* one-way or simply one-way.)

DEFINITION 3. A polynomial time computable function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is called *strongly one-way* if for any probabilistic polynomial time algorithm A, any positive constant c, and sufficiently large k,

$$Prob\left[A(f(x), 1^k) \in f^{-1}(f(x))\right] < k^{-c},$$

where the probability is taken over all x's of length k and the internal coin tosses of A, with uniform probability distribution.

THEOREM (Yao [Y]). *There exists a strong one-way function if and only if there exists a (weak) one-way function. Furthermore, given a one-way function, a strong one can be constructed.*

It is important to note that Yao's construction preserves the regularity of the function. Thus, we may assume without loss of generality, that we are given a function f, which is strongly one-way and regular.

For the sake of simplicity, we assume f is *length preserving* (i.e., for all x, $|f(x)| = |x|$). Our results hold also without this assumption (see §2.6).

Notation. For a finite set S, the notation $s \in_R S$ means that the element s is randomly selected from the set S with uniform probability distribution.

2.1. Levin's criterion: A modified version. The proof of the Main Theorem relies on the transformation of a function, which is one-way and regular into a function, which satisfies a variant of Levin's one-way on iterates condition. The modified condition relates to functions, which leave the first part of their argument unchanged. It requires that the function is one-way on a number of iterates, which exceeds the length of the second part of its argument. (Levin requires that the function is one-way on a number of iterations exceeding the length of the *entire* argument.)

More precisely, we consider functions $F(\cdot, \cdot)$ defined as

$$F(h, x) = (h, F_0(h, x)).$$

That is, F applies a function F_0 on its arguments and concatenates the first argument h to this result. We prove the following condition.

LEMMA 1. *A sufficient condition for the existence of a pseudorandom generator is the existence of a function F of the form*

$$F(h, x) = (h, F_0(h, x)),$$

such that F is strongly one-way for $|x| + 1$ iterations.

Before proving Lemma 1, let us recall the Blum–Micali scheme for the construction of pseudorandom generators [BM]. This scheme uses two basic elements: the first, a (strongly) one-way function f, and the second, a Boolean predicate $b(\cdot)$ called a "hard-core" of the function f. (Roughly speaking, a Boolean function $b(\cdot)$ is a *hard-core predicate of f* if it is polynomial time computable, but no polynomial time probabilistic algorithm given $f(x)$, for randomly selected x, can compute the value of $b(x)$ with a probablility significantly better than $\frac{1}{2}$.) A pseudorandom generator G is constructed in the following way. On input x (the seed), the generator G applies iteratively the one-way function $f(\cdot)$ on x for $t (= poly(|x|))$ times (i.e., $f(x), f^{(2)}(x), \ldots, f^{(t)}(x)$). In each application of f, the predicate $b(f^{(i)}(x))$ is computed and the resultant bit is output by the generator; that is, G outputs a string of length t. Blum and Micali show that the above sequence of bits is unpredictable when presented in reverse order (i.e., $b(f^{(t)}(x))$ first and $b(f^{(1)}(x))$ last), provided that the Boolean function $b(\cdot)$ is a hard-core predicate on the distribution induced by the iterates $f^{(i)}, 0 \leq i \leq t$. The unpredictability of the sequence is proved by showing that an algorithm, which succeeds to predict the next bit of the sequence with probability better than one-half can be transformed into an algorithm for "breaking" the hard-core of the function f. Finally applying Yao's Theorem [Y] that unpredictable sequences are pseudorandom, we get that the above G is indeed a pseudorandom generator.

The crucial ingredient in the proof of Levin's condition, as well as of our modified version, is the existence of a hard-core predicate for any (slightly modified) one-way function. A recent result of Goldreich and Levin [GL] greatly simplifies the original proof in [L]. This result states that any function $f'(x, r) = (f(x), r)$, where $|x| = |r|$ has a hard-core predicate for the uniform distribution on r and any distribution on x for which f is strongly one-way. This hard-core predicate is the inner product modulo 2 of r and x (viewed as vectors over Z_2).

Finally, we recall the following notable property of pseudorandom generators: in order to have a generator that expands strings to any polynomial length, it suffices to construct a generator that expands strings of length k into strings of length $k + 1$. This generator can be iteratively applied for polynomially many times without harming the pseudorandomness of its output [GrM]. We now prove Lemma 1.

Proof of Lemma 1. Note that $F^{(i)}(h, x) = (h, F_0^{(i)}(h, x))$. Thus, the condition in the lemma implies that $F_0(h, x)$ is hard to invert for $|x| + 1$ iterations even when h is given to the inverter. We construct the following generator, G, which expands its input by one bit. Let s be the seed for G, so that $s = (\bar{r}, h, x)$, where $|x| = n, \bar{r} = r_0, \dots, r_n)$, and for all i, $|r_i| = n$. Then, we define

$$G(s) = G(\bar{r}, h, x) = (\bar{r}, h, b_0, \dots, b_n),$$

where $i = 0, \dots, n, b_i$ is the inner product modulo 2 of r_i and $F_0^{(i)}(h, x)$. (We denote $F_0^{(0)}(h, x) = x$.)

We claim that this generator is pseudorandom. This is proved by noting that the output string is unpredictable. This is true for the $\bar{r}$ and h part as they were chosen as truly random strings. For the other bits, this is guaranteed by the Goldreich–Levin result and the fact that F_0 is hard to invert for $n + 1$ iterations (even when h is given to the inverter). □

2.2. Main ideas. We prove the Main Theorem by transforming any regular and (strongly) one-way function f into a new strongly one-way function F for which the conditions of Lemma 1 hold.

The following are the main ideas behind this construction. Since the function f is strongly one-way, any algorithm trying to invert f can succeed only with negligible probability. Here the probability distribution on the range f is induced by choosing a random element from the domain and applying f. However, this condition says nothing about the capability of an algorithm to invert f when the distribution on the range is substantially different. For example, there may be an algorithm that is able to invert f if we consider the distribution on the range elements induced by choosing a random element from the domain and applying f twice or more (see Appendix A). To prevent this possibility, we "randomly" redistribute, after each application of f, the elements in the range to locations in the domain. We prove the validity of our construction by showing that the probability distribution induced on the range of f by our "random" transformations (and the application of f) is close to the distribution induced by a single application of f.

The function F we construct must be deterministic; and therefore, the "random" redistribution must be deterministic (i.e., uniquely defined by the input to F). To achieve this, we use high-quality hash functions. More specifically, we use hash functions that map n-bit strings to n-bit strings, such that the locations assigned to the strings by a randomly selected hash function are uniformly distributed and n-wise independent. For properties and implementations of such functions, see [CW], [J], [CG], [Lu]. We denote this set of hash functions by $H(n)$. Elements of $H(n)$ can be described by bit strings of length n^2. In the sequel, $h(\in H(n))$ refers to both the hash function and to its representation.

2.3. The construction of *F*. We view the input string to F as containing two types of information. The first part of the input is the description of hash functions that implement the "random" redistributions, and the other part is interpreted as the input for the original function f.

The following is the definition of the function F:

$$F(h_0, \ldots, h_{t(n)-1}, i, x) = (h_0, \ldots, h_{t(n)-1}, i^+, h_i(f(x))),$$

where $x \in \{0,1\}^n$, $h_j \in H(n)$, $0 \leq i \leq t(n) - 1$. The function $t(n)$ is a polynomial in n, and i^+ is defined as $(i+1) \bmod t(n)$.

The rest of this section is devoted to the proof of the following theorem.

THEOREM 2. *Let f be a regular and strongly one-way function. Then the function F defined above is strongly one-way for $t(n)$ iterations on strings x of length n.*

Our Main Theorem follows from Theorem 2 and Lemma 1 by choosing $t(n) > n$.

Let $h_0, h_1, \ldots, h_{t(n)-1}$ be $t(n)$ functions from the set $H(n)$. For $r = 1, \ldots, t(n)$, let g_r be the function $g_r = fh_{r-1}fh_{r-2}f \ldots h_0 f$ acting on strings of length n, let $G_r(n)$ be the set of all functions g_r, let g be $g_{t(n)}$, and let $G(n)$ be the set of such functions g. From the above description of the function F it is apparent that the inversion of an iterate of F boils down to the problem of inverting f when the probability distribution on the range of f is $g_r(x)$, where $x \in_R \{0,1\}^n$. We show that, for most $g \in G(n)$, the number of preimages under g for each element in its range is close (up to a polynomial factor) to the number of preimages for the same range element under f. This implies that the same statement is true for most $g_r \in G_r(n)$ for all $r = 1, \ldots, t(n)$. The proof of this result reduces to the analysis of the combinatorial game that we present in the next subsection.

2.4. The game. Consider the following game played with M balls and M cells, where $t(n) \ll M \leq 2^n$. Initially, each cell contains a single ball. The game has $t(n)$ iterations. In each iteration, cells are mapped randomly to cells by means of an independently and randomly selected hash function $h \in_R H(n)$. This mapping induces a transfer of balls so that the balls residing (before an iteration) in cell σ are transferred to cell $h(\sigma)$. We are interested in bounding the probability that some cells contain "too many" balls when the process is finished. We show that after $t(n)$ iterations, for $t(n)$ a polynomial, the probability that there is any cell containing more than some polynomial in n balls is negligibly small (i.e., less than any polynomial in n fraction).

We first proceed to determine a bound on the probability that a specific set of n balls is mapped after $t(n)$ iterations to a single cell.

LEMMA 3. *The probability, over $h_0, h_1, \ldots, h_{t(n)-1} \in_R H(n)$, that a specific set of n balls is mapped after $t(n)$ iterations to the same cell is bounded above by $p(n) = [(n \cdot t(n))/M]^{n-1}$.*

Proof. Let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of n balls. Notice that each execution of the game defines for every ball b_i a path through $t(n)$ cells. In particular, fixing $t(n)$ hash functions $h_0, h_1, \ldots, h_{t(n)-1}$, a path corresponding to each b_i is determined. Clearly, if two such paths intersect at some point, then they coincide beyond this point. We modify these paths in the following way. The initial portion of the path for b_i that does not intersect the path of any smaller indexed ball is left unchanged. If the path for b_i intersects the path for b_j for some $j < i$, then the remainder of the path for b_i is chosen randomly and independently of the other paths from the point of the first such intersection.

Because the functions h_i are chosen totally independently of each other and because each of them has the property of mapping cells in an n-independent manner, it follows that the modified process just described is equivalent to a process in which a totally random path is selected for each ball in B. Consider the modified paths. We say that two balls b_i and b_j *join* if and only if their corresponding paths intersect. Define *merge* to be the reflexive and transitive closure of the relation join (over B). The main observation is that if $h_0, h_1, \ldots, h_{t(n)-1}$ map the balls of B to the same cell, then $b_1, b_2, \ldots, b_n$ are all in the same equivalence class with respect to the relation merge. In other words, the probability that the balls in B end up in the same cell in the original game is bounded above by the probability that the merge relation has a single equivalence class (containing all of B). Let us now consider the probability of the latter event.

If the merge relation has a single equivalence class, then the join relation defines a connected graph, which we call the *join graph*, with the n balls as vertices and the join relation as the set of edges. The join graph is connected if and only if it contains a spanning tree. Thus an upper bound on the probability that the join graph is connected is obtained by the sum of the probabilities of each of the possible spanning trees, which can be embedded in the graph. Each particular tree has probability at most $(t(n)/M)^{n-1}$ to be embedded in the graph since $(t(n)/M$ is an upper bound on the probability of each edge to appear in the graph). Multiplying this probability by the (Cayley) number of different spanning trees (n^{n-2} cf. [E, §2.3]), the lemma follows. □

A straightforward upper bound on the probability that there is some set of n balls, which are merged, is the probability that n specific balls are merged multiplied by the number of possible distinct subsets of n balls. Unfortunately, this bound is worthless as $\binom{M}{n} \cdot p(n) > 1$. (This phenomenon is independent of the choice of the parameter n.) Instead, we use the following technical lemma.

LEMMA 4. *Let S be a finite set, and let Π denote a partition of S. Assume we have a probability distribution on partitions of S. For every $A \subseteq S$, we define $\chi_A(\Pi) = 1$ if A is contained in a single class of the partition Π and $\chi_A(\Pi) = 0$ otherwise. Let n and n' be integers such that $n < n'$. Let $p(n)$ be an upper bound on the probability that $\chi_A = 1$, for any subset $A \subseteq S$ of size n. Let $q(n')$ be the probability that there exists some $B \subseteq S$ such that $|B| \geq n'$ and $\chi_B = 1$. Then*

$$q(n') \leq \frac{\binom{|S|}{n} \cdot p(n)}{\binom{n'}{n}}.$$

Proof. For $B \subseteq S$ we define $\xi_B(\Pi) = 1$ if B is exactly a single class of the partition Π and $\xi_B(\Pi) = 0$ otherwise. Fix a partition Π. Observe that every B, $|B| \geq n'$, for which $\xi_B(\Pi) = 1$, contributes at least $\binom{n'}{n}$ *different* subsets A of size n for which $\chi_A = 1$. Thus, we get that

$$\binom{n'}{n} \cdot \sum_{B\subseteq S, |B|\geq n'} \xi_B(\Pi) \leq \sum_{A\subseteq S, |A|=n'} \chi_A(\Pi).$$

Dividing both sides of this inequality by $\binom{n'}{n}$, and averaging according to the probability distribution on the partitions Π, the left-hand side is an upper bound for $q(n')$, while the right-hand side is bounded above by $\binom{|S|}{n} \cdot p(n)/\binom{n'}{n}$. □

Remark. Lemma 4 is useful in situations when the ratio $p(n)/p(n')$ is smaller than $(|S| - n \quad n' - n)$. Assuming that $n' \ll |S|$, this happens when $p(n)$ is greater than $|S|^{-n}$. Lemma 3 is such a case; and thus, the application of Lemma 4 is useful.

Combining Lemmas 3 and 4, we get the following theorem.

THEOREM 5. *Consider the game played for $t(n)$ iterations. Then, the probability that there are $4t(n) \cdot n^2 + n$ balls, which end up in the same cell, is bounded above by 2^{-n}.*

Proof. Let S be the set of M balls in the above game. Each game defines a partition of the balls according to their position after $t(n)$ iterations. The probability distribution on these partitions is induced by the uniform choice of the mappings h. Theorem 5 follows by using Lemma 4 with $n' = 4t(n) \cdot n^2 + n$ and the bound $p(n)$ of Lemma 3. (We also use the fact that $M \leq 2^n$ and the binomial bound $(n' \quad n) \geq (n'/n - 1)^n$.) □

2.5. Proof of Theorem 2. We now apply Theorem 5 to the analysis of the function F. As before, let $G(n)$ be the set of functions of the form $g = fh_{t(n)-1}f \ldots h_0 f$. The functions $h = h_j$ are hash functions used to map the range of f to the domain of f. We let $h_0, \ldots, h_{t(n)-1}$ be randomly chosen uniformly and independently from $H(n)$, and this induces a probability distribution on $G(n)$. Denote the range of f (on strings of length n) by $R(n) = \{z_1, z_2, \ldots, z_M\}$. Let each z_i represent a cell. Consider the function h as mapping cells to cells. We say that h maps the cell z_i to the cell z_j if $h(z_i) \in f^{-1}(z_j)$, or in other words $f(h(z_i)) = z_j$. By the regularity of the function f, we have that the size of $f^{-1}(z_i)$ (which we have denoted by $m(n)$) is equal for all $z_i \in R(n)$; and therefore, the mapping induced on the cells is uniform. It is now apparent that $g \in_R G(n)$ behaves exactly as the random mappings in the game described in §2.4; and thus, Theorem 5 can be applied to obtain the next lemma. (Notice that $g \in_R G(n)$ means choosing $t(n)$ functions $h_0, \ldots, h_{t(n)-1} \in_R H(n)$ and putting $g = fh_{t(n)-1}f \ldots h_0 f$.)

LEMMA 6. *There is a constant c_0, such that for any constant $c > 0$ and sufficiently large n*

$$Prob\left[\exists z \text{ with } |g^{-1}(z)| \geq n^{c_0} \cdot m(n)\right] \leq \frac{1}{n^c},$$

where $g \in_R G(n)$.

Note. The constant c_0 depends on the degree of $t(n)$. More precisely, we need $n^{c_0} \geq 4t(n) \cdot n^2 + n$ (see Thm. 5).

Let us denote by $G'(n)$ the set of functions $g \in G(n)$ such that for all z in the range of f, $|g^{-1}(z)| < n^{c_0} \cdot m(n)$. By Lemma 6, $G'(n)$ contains almost all of $G(n)$. It is clear that if $g \in G'(n)$, then for all z in the range of f and for all $r = 1, \ldots, t(n)$ the function g_r defined by the first r iterations of g satisfies $|g_r^{-1}(z)| < n^{c_0} \cdot m(n)$.

LEMMA 7. *For any probabilistic polynomial time algorithm A, for any positive constant c and sufficiently large n and for all $r = 1, \ldots, t(n)$,*

$$Prob(A(g_r, z) \in f^{-1}(z)) < n^{-c},$$

where $g_r \in_R G_r(n)$ and $z = g_r(x), x \in_R \{0, 1\}^n$.

Proof. We prove the claim for $r = t(n)$, and the claim for $r = 1, \ldots, t(n)$ follows in an analogous way. Assume to the contrary that there is a probabilistic polynomial time algorithm A and a constant c_A such that $Prob(A(g, z) \in f^{-1}(z)) > n^{-c_A}$, where $g \in_R G(n)$ and $z = g(x), x \in_R \{0, 1\}^n$.

By using A, we can demonstrate an Algorithm A' that inverts f, contradicting the one-wayness of f. The input to A' is $z = f(x)$, where $x \in_R \{0, 1\}^n$. A' chooses $g \in_R G(n)$ and outputs $A(g, z)$. We show that A' inverts f with nonnegligible probability. By assumption there is a nonnegligible subset $G''(n)$ of $G'(n)$ such that, for each $g \in G''(n)$, A succeeds with significant probability to compute a $y \in f^{-1}(z)$, where $z = g(x)$ and $x \in_R \{0, 1\}^n$. Since $g \in G'(n)$, for all z in the range of f the probability induced by g on z differs by at most a polynomial factor in n from the probability induced by f. Thus, for $g \in G''(n)$, A succeeds with significant probability to compute a $y \in f^{-1}(z)$, where $z = f(x)$ and $x \in_R \{0, 1\}^n$. This is exactly the distribution of inputs to A', and thus A' succeeds to invert f with nonnegligible probability, contradicting the strong one-wayness of f. □

The meaning of Lemma 7 is that the function f is hard to invert on the distribution induced by the functions $g_r, r = 1, \ldots, t(n)$, thus proving the strong one-wayness of the function F for $t(n)$ iterations. Theorem 2 follows.

2.6. Extensions. In the above exposition, we assumed for simplicity that the function f is length preserving, i.e., $x \in \{0, 1\}^n$ implies that the length of $f(x)$ is n. This condition is not essential to our proof and can be dispensed with in the following way. If f is not length preserving, then it can be modified to have the following property: For every n, there is an n' such that $x \in \{0, 1\}^n$ implies that the length of $f(x)$ is n'. This modification can be carried out using a padding technique that preserves the regularity of f. We can then modify our description of F to use hash techniques mapping n'-bit strings to n-bit strings. Alternatively, we can transform the above f into a length preserving and regular function $\hat{f}$ by defining $\hat{f}(xy) = f(x)$, where $|x| = n$, $|y| = n' - n$.

For the applications in §3, and possibly for other cases, the following extension (referred to as *semiregular*) is useful. Let $\{f_x\}_{x\in\{0,1\}^*}$ be a family of regular functions, then our construction can still be applied to the function f defined as $f(x, y) = (x, f_x(y))$. The idea is to use the construction for the application of the function f_x, while keeping x unchanged.

Another extension is a relaxation of the regularity condition. A useful notion in this context is the histogram of a function.

DEFINITION 4. The *histogram* of the function $f : \{0, 1\}^* \to \{0, 1\}^*$ is a function $hist_f : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ such that $hist_f(n, k)$ is the cardinality of the set

$$\{x \in \{0, 1\}^n : \lfloor \log_2 |f^{-1}(f(x))| \rfloor = k\}.$$

Regular functions have trivial histograms: Let f be a regular function such that for all $x \in \{0, 1\}^n$, $|f^{-1}(f(x))| = m(n)$. The histogram satisfies $hist_f(n, k) = 2^n$ for $k = \lfloor \log_2(m(n)) \rfloor$ and $hist_f(n, k) = 0$ otherwise. Weakly regular functions have slightly less dramatic histograms.

DEFINITION 5. The function f is *weakly regular* if there is a polynomial $p(\cdot)$ and a function $b(\cdot)$ such that the histogram of f satisfies (for all n)

(i) $hist_f(n, b(n)) \geq \frac{2^n}{p(n)}$

(ii) $\sum_{k=b(n)+1}^{n} hist_f(n, k) < \frac{2^n}{(n \cdot p(n))^2}$

Clearly, this definition extends the original definition of regularity. Using our techniques, one can show that the existence of weakly regular strongly one-way functions implies the existence of pseudorandom generators. Details follow.

Observe that if the $b(n)$th level of the histogram contains all of the 2^n strings of length n, then we can apply a similar analysis as done for the regular case. The only difference is that we have to analyze the game of §2.4 not for cells of equal size, but for cells that differ in their size by a multiplicative factor of at most two. Similar arguments hold when considering the case where the $b(n)$th level of the histogram contains at least $1/p(n)$ of the strings and the rest of strings lie below this level (i.e., $hist_f(n, k) = 0$, for $k > b(n)$). Note that the "small" balls of low levels cannot cause the cells of the $b(n)$th level to grow significantly. On the other hand, for balls below level $b(n)$ nothing is guaranteed. Thus, we get that in this case the function F we construct is weakly one-way on its iterates. More precisely, it is hard to invert on its iterates for at least a $1/p(n)$ fraction of the input strings. In order to use this function for generating pseudorandom bits, we have to transform it into a strongly one-way function. This is achieved following Yao's construction [Y] by applying F in parallel on many copies. For the present case, the number of copies could be any function of n, which grows faster than $c \cdot p(n) \cdot \log n$, for any constant c. This increases the number of iterations for which F has to remain one-way by a factor equal to the number of copies used in the above transformation. That is, the number $t(n)$ of necessary iterates increases from the original requirement of $n + 1$ (see §2.1) to a quantity that is greater than $c \cdot p(n) \cdot n \cdot \log n$, for any constant c. Choosing this way the function $t(n)$ in the definition of F in §2.3, we get F, which is one-way for the right number of iterations.

Finally, consider the case in which there exist strings above the $b(n)$th level. When considering the game of §2.4 we want to show that, also in this case, most of the cells of the $b(n)$th level do not grow considerably. This is guaranteed by condition (ii) in Definition 5. Consider the worst case possibility in which in every iteration the total weight of the "big" balls (those above level $b(n)$) is transferred to cells of the $b(n)$th level. After $t(n)$ iterations, this causes a concentration of big balls in the $b(n)$th level having a total weight of at most $t(n) \cdot 2^n/(n \cdot p(n))^2$. Choosing $t(n) = \frac{1}{2}p(n)n^2$, this weight will be at most $2^n/(2p(n))$. But then one-half of the weight in the $b(n)$th level remains concentrated in balls that were not affected by the big balls. In other words, we get that the function F so constructed is one-way for $t(n)$ iterations on $1/(2p(n))$ of the input strings. Applying Yao's construction, as explained above, we get a function F, which satisfies the criterion of Lemma 1 and is then suitable for the construction of pseudorandom generators.

Further Remarks.

1. The denominator in condition (ii) of Definition 5 can be substituted by any function growing faster than $c \cdot p^2(n) \cdot n$, for any constant c. This follows from the above analysis and the fact that the construction of a hard-core predicate in [GL] allows extracting $\log n$ secure bits with each application of the one-way function.

2. The entire analysis holds when defining histograms with polynomial base (instead of base 2). Namely, $hist_f(n, k)$ is the cardinality of the set

$$\{x \in \{0,1\}^n : \lfloor \log_{Q(n)} |f^{-1}(f(x))| \rfloor = k\},$$

where $Q(n)$ is a polynomial.

3. Applications: Pseudorandom generators based on particular intractability assumptions. In this section, we apply our results in order to construct pseudorandom generators (PRGs) based on the assumption that one of the following computational problems is "hard on a nonnegligible fraction of the instances."

3.1. PRG based on the intractability of the general factoring problem. It is known that pseudorandom generators can be constructed assuming the intractability of factoring integers of a special form [Y]. More specifically, in [Y] it is assumed that any polynomial time algorithm fails to factor a nonnegligible fraction of integers that are the product of primes congruent to 3 modulo 4. With respect to such an integer N, squaring modulo N defines a permutation over the set of quadratic residues mod N; therefore, the intractability of factoring (such N's) yields the existence of a one-way permutation [R]. It was not known how to construct a one-way permutation or a pseudorandom generator assuming that factoring a nonnegligible fraction of *all* the integers is intractable. In such a case, modular squaring is a one-way function, but this function does not necessarily induce a permutation. Fortunately, modular squaring is a semiregular function (see §2.6), so we can apply our results.

Assumption IGF (*Intractability of the General Factoring Problem*): There exists a constant $c > 0$ such that for *any* probabilistic polynomial time algorithm A and sufficiently large k

$$Prob\,[A(N) \text{ does not factorize } N] > k^{-c},$$

where $N \in_R \{0,1\}^k$.

COROLLARY 8. *The* IGF *assumption implies the existence of pseudorandom generators.*

Proof. Define the following function $f(N, x) = (N, x^2 \bmod N)$. Clearly, this function is semiregular. The one-wayness of the function follows from IGF (using Rabin's argument [R]). Using an extension of Theorem 2 (see §2.6) the corollary follows. □

Subsequently, J. (Cohen) Benaloh has found a way to construct a one-way permutation based on the IGF assumption. This yields an alternative proof of Corollary 8.

3.2. PRG based on the intractability of decoding random linear codes. One of the most outstanding open problems in coding theory is that of decoding random linear codes. Of particular interest are random linear codes with constant information rate, which can correct a constant fraction of errors. An *(n, k, d)-linear code* is an k-by-n binary matrix in which the bit-by-bit XOR of any subset of the rows has at least d ones. The Gilbert–Varshamov bound for linear codes guarantees the existence of such a code provided that $k/n < 1 - H_2(d/n)$, where H_2 is the binary entropy function [McS, Chap. 1, p. 34]. The same argument can be used to show (for every $\epsilon > 0$) that if $k/n < 1 - H_2((1+\epsilon) \cdot d/n)$, then almost all k-by-n binary matrices constitute (n, k, d)-linear codes.

We suggest the following function $f : \{0,1\}^* \rightarrow \{0,1\}^*$. Let C be an k-by-n binary matrix, $x \in \{0,1\}^k$, and let $e \in E_t^n \subseteq \{0,1\}^n$ be a binary string with at most $t = \lfloor (d-1)/2 \rfloor$ ones, where d satisfies the condition of the Gilbert–Varshamov bound (see above). Clearly, E_t^n can be uniformly sampled by an algorithm S running in time polynomial in n (i.e., $S : \{0,1\}^{poly(n)} \rightarrow E_t^n$). Let $r \in \{0,1\}^{poly(n)}$ be a string such that $S(r) \in E_t^n$. Then,

$$f(C, x, r) = (C, C(x) + S(r)),$$

where $C(x)$ is the code word of x (i.e., $C(x)$ is the vector resulting by the matrix product xC). One can easily verify that f just defined is semiregular (i.e., $f_C(x, r) = C(x) + S(r)$ is regular for all but a negligible fraction of the C's). The vector $xC + e(e = S(r))$ represents a code word perturbed by the error vector e.

Assumption IDLC (*intractability of decoding random linear codes*). There exists a constant $c > 0$ such that for *any* probabilistic polynomial time Algorithm A and sufficiently large k

$$Prob\,[A(C, C(x) + e) \neq x] > k^{-c},$$

where C is a randomly selected k-by-n matrix, $x \in_R \{0,1\}^k$ and $e \in_R E_t^n$.

Now, either Assumption IDLC is false, which would be an earth-shaking result in coding theory, or pseudorandom generators do exist.

COROLLARY 9. *The* IDLC *assumption implies the existence of pseudorandom generators.*

Proof. The one-wayness of the function f follows from IDLC. Using an extension of Theorem 2 (see §2.6) the corollary follows. □

3.3. PRG based on the average difficulty of combinatorial problems. Some combinatorial problems, which are believed to be hard on the average, can be used to construct a regular one-way function and hence be a basis for a pseudorandom generator. Consider, for example, the *Subset-Sum Problem.*

Input. Modulo M, $|M| = n$, and $n+1$ integers $a_0, a_1, \ldots, a_n$ of length n-bit each.

Question. Is there a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i \equiv a_0 (\text{mod } M)$?

Conjecture. The above problem is hard on the average, when the a_i's and M are chosen uniformly in $[2^{n-1}, 2^n - 1]$.

Under the above conjecture, the following weakly regular function is one-way

$$f_{SS}(a_1, a_2, \ldots, a_n, M, I) = \left(a_1, a_2, \ldots, a_n, M, \left(\sum_{i \in I} a_i \bmod M\right)\right).$$

Appendix A. One-way functions, which are not one-way on their iterates. Assuming that f is a (regular) one-way function, we construct a (regular) one-way function $\bar{f}$, which is easy to invert on the distribution obtained by iterating $\bar{f}$ twice. Assume for simplicity that f is length preserving (i.e., $|f(x)| = |x|$). let $|x| = |y|$ and let

$$\bar{f}(xy) = 0^{|y|} f(x).$$

Clearly, $\bar{f}$ is one-way. On the other hand, for every $xy \in \{0, 1\}^{2n}$, $\bar{f}(\bar{f}(xy)) = 0^n f(0^n)$ and $0^n f(0^n) \in \bar{f}^{-1}(0^n f(0^n))$.

Acknowledgments. We are grateful to Josh (Cohen) Benaloh, Manuel Blum, Leonid Levin, Richard Karp, Charles Rackoff, Ronny Roth, and Avi Wigderson for very helpful discussions concerning this work.

The first author wishes to express special thanks to Leonid Levin and Silvio Micali for the infinite number of discussions concerning pseudorandom generators.

REFERENCES

[ACGS] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, RSA *and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[BBS] L. Blum, M. Blum, and M. Shub, *A simple secure unpredictable pseudo-random number generator*, SIAM J. Comput., 15 (1986), pp. 364–383.

[BM] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1986), pp. 850–864.

[CW] J. Carter and M. Wegman, *Universal classes of hash functions*, JCSS, 18 (1979), pp. 143–154.

[CG] B. Chor and O. Goldreich, *On the power of two-point sampling*, J. Complexity, 5 (1989), pp. 96–106.

[CGG] B. Chor, O. Goldreich, and S. Goldwasser, *The bit security of modular squaring given partial factorization of the modulos*, Adv. Cryptology – Crypto 85 Proceedings, H. C. Williams, ed., Lecture Notes in Computer Science, 218, Springer-Verlag, 1985, pp. 448–457.

[DH] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory, IT-22, Nov. 1976, pp. 644–654.

[E] S. Even, *Graph Algorithms*, Computer Science Press, New York, 1979.

[GGM] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. Assoc. Comput. Mach., 33 (1986), pp. 792–807.

[GKL] O. Goldreich, H. Kawczyk, and M. Luby, *On the existence of pseudorandom generators*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 12–24.

[GL] O. Goldreich and L. A. Levin, *A hard-core predicate for any one-way function*, Proc. 21st Symposium on Theory of Computing, 1989, pp. 25–32.

[GrM] O. Goldreich and S. Micali, *The Weakest Pseudorandom Bit Generator Implies the Strongest One*, manuscript, 1984.

[GM] S. Goldwasser and S. Micali, *Probabilistic encryption*, JCSS, 28 (1984), pp. 270–299.

[H] J. Hastad, *Pseudo-random generators under uniform assumptions*, Proc. 22nd Symposium on Theory of Computing, 1990, pp. 395–404.

[ILL] R. Impagliazzo, L. A. Levin, and M. G. Luby, *Pseudo-random generation from one-way functions*, Proc. 21st Symposium on Theory of Computing, 1989, pp. 12–24.

[J] A. Joffe, *On a set of almost deterministic k-independent random variables*, Ann. Probab., 2 (1974), pp. 161–162.

[L] L. A. Levin, *One-way function and pseudorandom generators*, Combinatorica, 7 (1987), pp. 357–363; a preliminary version appeared in Proc. 17th Symposium on Theory of Computing, 1985, pp. 363–365.

[L2] ———, *Homogeneous measures and polynomial time invariants*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 36–41.

[Lu] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1054.

[LR] M. Luby and C. Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, SIAM J. Comput., 17 (1988), pp. 373–386.

[McS] F. J. McWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland Publishing Company, Amsterdam, 1977.

[R] M. O. Rabin, *Digitalized Signatures and Public Key Functions as Intractable as Factoring*, MIT/LCS/TR-212, 1979.

[RSA] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

[S] A. Shamir, *On the generation of cryptrographically strong pseudorandom sequences*, ACM Trans. Comput. Systems, 1 (1983), pp. 38–44.

[Y] A. C. Yao, *Theory and applications of trapdoor functions*, Proc. of 23rd IEEE Symposium on Foundation of Computer Science, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1176–1198, December 1993

A GENERALIZED SUFFIX TREE AND ITS (UN)EXPECTED ASYMPTOTIC BEHAVIORS*

WOJCIECH SZPANKOWSKI†

Abstract. Suffix trees find several applications in computer science and telecommunications, most notably in algorithms on strings, data compressions, and codes. Despite this, very little is known about their typical behaviors. In a probabilistic framework, a family of suffix trees—further called b-suffix trees—built from the first n suffixes of a random word is considered. In this family a noncompact suffix tree (i.e., such that every edge is labeled by a single symbol) is represented by $b = 1$, and a compact suffix tree (i.e., without unary nodes) is asymptotically equivalent to $b \to \infty$ as $n \to \infty$. Several parameters of b-suffix trees are studied, namely, the depth of a given suffix, the depth of insertion, the height and the shortest feasible path. Some new results concerning typical (i.e., *almost sure*) behaviors of these parameters are established. These findings are used to obtain several insights into certain algorithms on words, molecular biology, and universal data compression schemes.

Key words. generalized suffix trees, algorithms on words, data compression, height, shortest path, typical depth and depth of insertion, probabilistic models, mixing condition, Rényi's entropy

AMS subject classifications. 68Q25, 68P05

1. Introduction.

In recent years, there has been a resurgence of interest in algorithmic and combinatorial problems on words due to a number of novel applications in computer science, telecommunications, and most notably in molecular biology (cf. [40]). In computer science and molecular biology, many algorithms depend on a solution to the following problem: given a word X and a set of arbitrary $b + 1$ suffixes $S_1, \ldots, S_{b+1}$ of X, what is the longest common prefix of these suffixes (cf. [2], [3], [9], [12], [42]). In coding theory (e.g., prefix codes) one asks for the shortest prefix of a suffix S_i, which is not a prefix of any other suffixes S_j, $1 \le j \le n$ of a given sequence X (cf. [34]). In data compression schemes, the following problem is of prime interest: for a given "data base" subsequence of length n, find the longest prefix of the $(n + 1)$st suffix S_{n+1} which is not a prefix of any other suffixes S_i $(1 \le i \le n)$ of the data base sequence (cf. [25], [43], [44]). And last but not least, in molecular sequences comparison (e.g., finding homology between DNA sequences), one may search for the longest run of a given motif (cf. [16], [17], [40]), a unique sequence or the longest alignment (cf. [13], [40]). These, and several other problems on words, can be efficiently solved and analyzed by a clever manipulation of a data structure known as a *suffix tree* (cf. [2], [27], [41]). In literature, other names have been also coined for this structure, and among these we mention here position trees, subword trees, directed acyclic graphs, etc. (cf. [1]).

Suffix trees find a wide variety of applications in algorithms on words including: the longest repeated substring (cf. [41]), squares or repetitions in strings (cf. [3]), string statistics (cf. [3]), string matching (cf. [9], [42]), approximate string matching (cf. [12], [9], [42]) string comparison, compression schemes (cf. [25]), implementation of the Lempel–Ziv algorithm, genetic sequences, biologically significant motif patterns in DNA (cf. [9], [40]), sequence assembly (cf. [9]), approximate-overlaps (cf. [9], [40]), and so forth. It is fair to say that suffix trees are the most widely used data structure in algorithms on words. Despite this, very little is known about their behaviors in a probabilistic framework. Recently, Chang and Lawler (cf. [9]) used some elementary property of suffix trees to design a superfast algorithm

*Received by the editors March 26, 1992; accepted for publication (in revised form) July 2, 1992. A preliminary version of this paper was presented at the *Combinatorial Pattern Matching* conference in Tucson, Arizona, 1992. This research was supported in part by National Science Foundation grants CCR-8900305, CCR-9201078, and NCR-9206315 and INT-8912631, in part by Air Force Office of Scientific Research grant 90-0107, North Atlantic Treaty Organization grant 0057/89, and grant R01 LM05118 from the National Library of Medicine.

†Department of Computer Science, Purdue University, West Lafayette, Indiana 47907.

for the approximate string matching problem. In our opinion, any further development in this direction requires better understanding of the behavior of suffix trees in a probabilistic framework.

In general, a suffix tree is a digital tree built from suffixes of a given word X; therefore, it fits into the subject of digital search indexes (cf. [23]). A digital tree stores n strings $\{S_1, \dots, S_n\}$ built over a finite alphabet Σ. In such a tree, every edge is labeled by a symbol (or a set of symbols) from the alphabet Σ and leaves (called also external nodes) contain the strings. The access path from the root to the external node is a minimal prefix information contained in the leaf (for more details, see [14] and [23]). If the strings $\{S_1, \dots, S_n\}$ are statistically *independent* and every edge is labeled by a *single* symbol from Σ, then the resulting digital tree is called a regular (or independent) trie (cf. [1], [14], [23]). If all unary nodes of a trie are eliminated, then the tree becomes a PATRICIA trie (cf. [14], [23], [37]). Finally, if an external node in a regular trie can store up to b strings (keys), then such a tree is called a b-trie. As mentioned above, a suffix tree is a special trie in which the strings $\{S_1, \dots, S_n\}$ are suffixes of a given sequence X. Note that in this case the strings are statistically dependent!

As in the case of regular tries, there are several modifications of the standard suffix tree. In a *noncompact suffix tree*—called also spread suffix tree and position tree—each edge is labeled by a letter from the alphabet Σ. If all unary nodes are eliminated in the noncompact version of the suffix tree, then the resulting tree is called *compact suffix tree* (cf. [2]). Gonnet and Baeza-Yates [14] coined a name PAT for such a suffix tree to resemble the name PATRICIA used for compact tries. Hereafter, we adopt this notation.

In this paper, we additionally introduce a family of suffix trees parametrized by an integer $b \geq 1$ such that $b = 1$ corresponds to a noncompact suffix tree and $b \to \infty$ is asymptotically equivalent (as $n \to \infty$) to PAT. A tree in such a family is constructed from the noncompact suffix tree by eliminating all unary nodes b levels above the fringe (bottom) of the tree. To simplify analysis, however, we shall modify this definition and assume that external nodes of b-suffix trees can store up to b suffixes. Note that such a suffix tree corresponds to a b-trie. Therefore, we coin a term b-suffix trees for them. These trees are useful in several applications, but more importantly, b-suffix trees form a spectrum of trees with noncompact suffix trees ($b = 1$) at one extreme and compact suffix trees ($b \to \infty$ as $n \to \infty$) at the other extreme (cf. Fig. 1). This allows us to assess some properties of PAT trees in a unified and substantially easier manner (e.g., compare [37], where PATRICIA tries are analyzed).

We offer a characterization of b-suffix trees in probabilistic framework, namely a word X over which the suffix tree is built represents a *stationary mixing* (*ergodic*) *sequence*. This sequence is assumed to be of infinite length (for bounded words see Rem. 2(iv), §2). We shall analyze the following parameters of b-suffix trees: the typical depth $D_n^{(b)}$, the depth of a particular suffix, say mth one, $L_n^{(b)}(m)$, the depth of insertion $L_n^{(b)}$, height $H_n^{(b)}$, and the shortest feasible path $s_n^{(b)}$. The typical depth represents the length of a path from the root to a *randomly* selected external node in a suffix tree; the depth of insertion is the depth of a *newly* inserted suffix; the height and the shortest feasible path are the longest and shortest path to an available node.

These parameters are most often used in the analysis and design of algorithms on words. For example, the typical depth D_n^{PAT} for the PAT tree built from the string $P\$T$, where P and T are the pattern and the text, respectively, is used by Chang and Lawler [9] in their design of an approximate string matching algorithm. On the other hand, the depth of insertion $L_n^{(1)}$ and the depth of a given suffix $L_n^{(1)}(m)$ of a noncompact suffix tree are of prime interest to the complexity of the Lempel–Ziv universal compression schemes (cf. [15], [25], [43]–[44]), and $L_n^{(1)}$ is responsible for a *dynamic* behavior of many algorithms on words. Furthermore, the

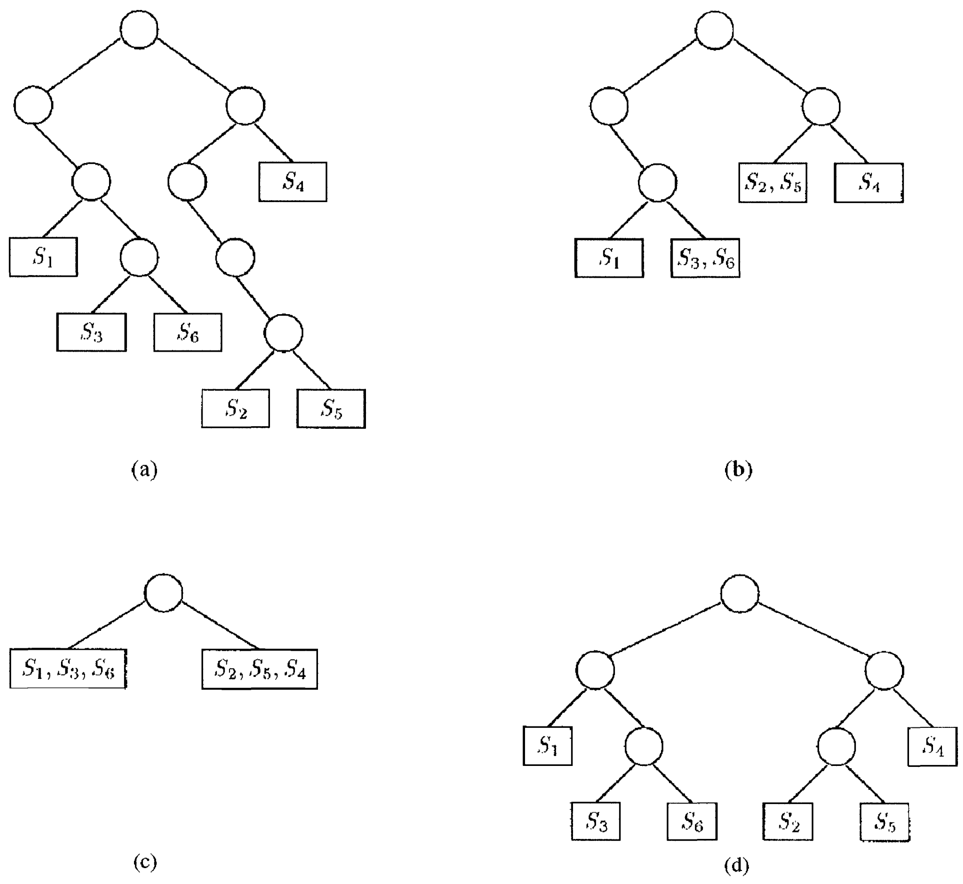


FIG. 1. *Suffix trees built from the first six suffixes of* = 0101101110...; (a) *noncompact suffix tree*; (b) 2-*suffix tree*; (c) 3-*suffix tree*; *and* (d) *compact suffix tree.*

height and the shortest path indicate how balanced a typical suffix tree is; that is, how much one has to worry about worst-case situations.

Our main results can be summarized as follows. For a b-suffix tree built over an unbounded word X, we prove that the normalized height $H_n^{(b)}/\log n$, the normalized shortest feasible path $s_n^{(b)}/\log n$, and the normalized depth of the mth suffix (m fixed) $L_n^{(b)}(m)/\log n$, almost surely (a.s.) converge to some explicit constants that depend on characteristics of the underlying probabilistic model. The most interesting behavior reveals that the normalized depth of insertion $L_n^{(b)}/\log n$ converges *in probability* (pr.) to a constant but *not* almost surely (a similar behavior shows the typical depth $D_n^{(b)}$). More interestingly, the almost sure behavior of a compact suffix tree can be deduced from the appropriate asymptotics of b-suffix trees by taking $b \to \infty$ as $n \to \infty$. More precisely, if we append superindex PAT to the appropriate parameters of a compact suffix tree, then we can prove that $\lim_{n\to\infty} H_n^{PAT}/\log n = \lim_{b\to\infty}\lim_{n\to\infty} H_n^{(b)}/\log n$, and in a similar fashion for $s_n^{(b)}$, $D_n^{(b)}$ and $L_n^{(b)}$. Note that the iterative limit above cannot be interchanged. Indeed, for example $\lim_{n\to\infty}\lim_{b\to\infty} H_n^{(b)} = 1$. It is worth mentioning that all these results are obtained in a uniform manner by a technique that encompasses the so-called string-ruler approach (cf. [19], [30]) and mixing condition technique. Our method, however, parallels, on several instances, Pittel's profound analysis of independent tries, and this research was inspired by the seminal paper of Pittel [30]. The details are discussed in §3.

Asymptotic analyses of suffix trees are very scanty in literature, and most of them deal with noncompact suffix trees, i.e., $b = 1$. To the best of our knowledge, there are no probabilistic results on b-suffix trees ($b > 1$) and compact suffix trees. This can be easily verified by checking §7.2 of the book by Gonnet and Baeza-Yates [14], which provides an up-to-date compendium of results concerning data structures and algorithms. The average case analysis of noncompact suffix trees was initialized by Grassberger [15], and Apostolico and Szpankowski [4]. For the Bernoulli model (independent sequence of letters from a finite alphabet) the asymptotic behavior of the height was recently obtained by Devroye, Szpankowski, and Rais [11], and the limiting distribution of the typical depth in a suffix tree is reported in Jacquet and Szpankowski [19]. Szpankowski [38] extended some of these results to a more general probabilistic model (still for $b = 1$). Heuristic arguments were used by Blumer, Ehrenfeucht, and Haussler [6] to show that the average number of internal nodes in a suffix tree is a linear function of n, and a rigorous proof of this can be found in [19]. Finally, Shields [34] recently established the almost sure behavior of the external path length of a noncompact suffix tree in the Bernoulli model and the Markovian model. Some related topics were discussed by Guibas and Odlyzko in [16] and [17].

This paper is organized as follows. In the next section, we formulate our main results and present several consequences of them. In particular, we intuitively explain why compact suffix trees can be considered as limiting b-suffix trees when $n \to \infty$. We also provide four applications of our results to data compression and pattern matching problems. Namely: (i) we settle two conjectures of Wyner and Ziv regarding the almost sure behavior of the repeated pattern and the size of the data base sequence in the universal data compression scheme (cf. [38]); (ii) we provide some information concerning the almost sure behavior of the block length in the Lempel–Ziv parsing algorithm (cf. [25], [44]); (iii) we present some complexity results regarding the Chang–Lawler pattern matching algorithm (cf. [9]); and finally (iv), we estimate the typical length of a unique subsequence of a given sequence (cf. [13]). Finally, §3 contains all formal proofs and presents some new results of combinatorics on words.

2. Main results and their consequences.

2. Main results and their consequences. In this section, we formally define b-suffix trees and introduce several parameters of these trees that are widely used in the complexity analysis of algorithms on words and data compression schemes. Next, we present all of our main results. We delay most of the proofs to the next section. Finally, we discuss some consequences of our findings.

2.1. Definitions and probabilistic models. A suffix tree is a trie built from suffixes of an (unbounded) sequence $\{X_k\}_{k=1}^{\infty}$ of symbols from an alphabet Σ of size V. More precisely, let $X = x_1x_2x_3\ldots$, then the ith suffix S_i of X is $S_i = x_ix_{i+1}\ldots$. By $\mathcal{S}_n$ we denote a digital tree built from the set $\{S_1, S_2, \ldots, S_n\}$ of the first n suffixes of X. In such a tree, which we further call a *noncompact suffix tree*, every edge is labeled by a single symbol from the alphabet Σ. Figure 1(a) shows a noncompact suffix tree built from the first six suffixes of $X = 0101101110\ldots$. A *compact suffix tree* called PAT tree (cf. [14]) is constructed from the noncompact version by eliminating all unary nodes (cf. Fig. 1(d)). It is characterized by the fact that an edge in such a tree is labeled by a substring of X (cf. [2], [27], [41]).

In this paper, we consider a family of suffix trees called b-suffix trees. A tree in such a family has no unary nodes in all b levels above the fringe level of the corresponding noncompact suffix tree. Note that a noncompact suffix tree coincides with 1-suffix tree, and a compact suffix tree corresponds to $b \to \infty$ as $n \to \infty$. For the purpose of our analysis, however, a modified definition of b-suffix trees is more convenient. Hereafter, *by a b-suffix tree we mean a trie built from the first n suffixes of X that can store up to b suffixes in an external node.* We denote such a suffix tree as $\mathcal{S}_n^{(b)}$. This definition is illustrated in Fig. 1(b) and 1(c). It is easy to note that if in a b-suffix tree we replace every external node by a complete binary tree with b nodes, then the latter definition of b-suffix tree corresponds to the former one.

Hereafter, we analyze several parameters of b-suffix trees that are formally defined below. The relevance of these parameters to the analysis and design of algorithms on words and data compression schemes was already discussed in the Introduction.

DEFINITION 1. TREES PARAMETERS

(i) The mth *depth* $L_n^{(b)}(m)$ is the length of a path from the root of the b-suffix tree $\mathcal{S}_n^{(b)}$ to the external node containing the mth suffix.

(ii) The *typical depth* $D_n^{(b)}$ is the depth of a randomly selected suffix, that is,

$$\Pr\{D_n^{(b)} \le k\} = \frac{1}{n}\sum_{m=1}^{n} \Pr\{L_n^{(b)}(m) \le k\}. \tag{2.1}$$

(iii) The *height* $H_n^{(b)}$ is the length of the longest depth, that is,

$$H_n^{(b)} = \max_{1\le m\le n}\{L_n^{(b)}(m)\}. \tag{2.2}$$

(iv) The *shortest feasible path* $s_n^{(b)}$ is the length of the shortest path from the root to an *available* (feasible) node, that is, a node that is *not* in the tree $\mathcal{S}_n^{(b)}$ but whose predecessor node (either an internal node or an external node) is in $\mathcal{S}_n^{(b)}$. In other words, $s_n^{(b)}$ is the shortest path to all external nodes and all internal nodes that have degree *smaller* than the size V of the underlying alphabet.

(v) The *shortest depth* $\tilde{s}_n^{(b)}$ is the length of the shortest path from the root to an *external node*. (Clearly, $s_n^{(b)} \le \tilde{s}_n^{(b)}$.)

(Another parameter of interest not studied in this paper is the external path length $E_n^{(b)}$ which is the sum of all depths, that is, $E_n^{(b)} = \sum_{m=1}^{n} L_n^{(b)}(m)$.)

For the purpose of this analysis, we present below another representation of the above tree parameters. We start with the following definition.

DEFINITION 2. SELF-ALIGNMENTS

For suffixes $S_1, S_2, \ldots, S_n$, the *self-alignment* $C_{i_1\ldots i_{b+1}}$ between $b+1$ suffixes, say $S_{i_1}, \ldots, S_{i_{b+1}}$, is the length of the longest common prefix of all these $b+1$ suffixes.

Then, the following relationships are easy to establish (cf. [4], [36])

$$L_n^{(b)}(i_{b+1}) = \max_{1\le i_1,\ldots,i_b\le n}\{C_{i_1\ldots i_{b+1}}\} + 1, \tag{2.3a}$$

$$H_n^{(b)} = \max_{1\le i_1,\ldots,i_{b+1}\le n}\{C_{i_1\ldots i_{b+1}}\} + 1, \tag{2.3b}$$

$$L_n^{(b)} = \max_{1\le i_1,\ldots,i_b\le n}\{C_{i_1\ldots i_b,n+1}\} + 1, \tag{2.3c}$$

$$\tilde{s}_n^{(b)} = \min_{1\le i_{b+1}\le n}\{L_n^{(b)}(i_{b+1})\} = \min_{1\le i_{b+1}\le n}\ \max_{1\le i_1,\ldots,i_b\le n}\{C_{i_1\ldots i_{b+1}}\} + 1. \tag{2.3d}$$

In passing, we note that for a stationary (infinite) ergodic sequence $\{X_k\}$, the self-alignment $C_{i_1\ldots i_{b+1}}$ does not depend explicitly on $i_1, \ldots, i_{b+1}$ but rather on the differences $d_k = i_{k+1} - i_k$. So, we also write $C_{1,1+d_1,\ldots,1+d_1+\cdots+d_b}$.

Our plan is to investigate the behavior of a random b-suffix tree in a general probabilistic framework. We could only assume that $\{X_k\}_{k=1}^{\infty}$ is a *stationary ergodic* sequence of symbols generated from a finite alphabet Σ, but this is too strong for our purpose. Therefore, we adopt the following two weaker probabilistic models.

(A1) MIXING MODEL. The sequence $\{X_k\}_{k=-\infty}^{\infty}$ satisfies the so-called mixing condition [5], that is, there exist two constants $0 < c_1 \le c_2$ and an integer d such that for all $-\infty \le m \le m + d \le n$ the following holds

$$c_1 \Pr\{\mathcal{A}\}\Pr\{\mathcal{B}\} \le \Pr\{\mathcal{AB}\} \le c_2 \Pr\{\mathcal{A}\}\Pr\{\mathcal{B}\} \tag{2.4a}$$

with $\mathcal{A} \in \mathcal{F}_{-\infty}^{m}$ and $\mathcal{B} \in \mathcal{F}_{m+d}^{\infty}$ where $\mathcal{F}_m^n$ is a σ-field generated by $\{X_k\}_{k=m}^{n}$ for $m \le n$.

In some statements of our results, we need a stronger form of the above mixing condition, which are defined in sequel.

(A2) STRONG MIXING MODEL. The sequence $\{X_k\}_{k=-\infty}^{\infty}$ satisfies the so-called *strong α-mixing condition* if (2.4a) is replaced by

$$(1 - \alpha(d))\Pr\{\mathcal{A}\}\Pr\{\mathcal{B}\} \le \Pr\{\mathcal{AB}\} \le (1 + \alpha(d))\Pr\{\mathcal{A}\}\Pr\{\mathcal{B}\}, \tag{2.4b}$$

where the function $\alpha(d)$ is such that $\alpha(d) \to 0$ as $d \to \infty$.

In words, model (A1) says that the dependency between $\{X_k\}_{k=-\infty}^{m}$ and $\{X_k\}_{k=m+d}^{\infty}$ is rather weak (note that when the sequence $\{X_k\}$ is independently and identically distributed, then $\Pr\{\mathcal{AB}\} = \Pr\{\mathcal{A}\}\Pr\{\mathcal{B}\}$). Assumption (A2) says that this dependency is weaker and weaker as d becomes larger. The "quantity" of dependency is characterized by $\alpha(d)$.

Finally, for compact suffix trees (i.e., PAT trees) we need one more assumption, which strengthens (A2).

(P) CONTRACTIVE MIXING MODEL. Let $\omega_i \in \Sigma$ for $1 \le i \le n$, and define the probability $P(\omega_1, \ldots, \omega_n) = \Pr\{X_1 = \omega_1, \ldots, X_n = \omega_n\}$. Then, for PAT trees we shall require the following condition

$$P(\omega_1, \ldots, \omega_n) \le \rho P(\omega_1, \ldots, \omega_{n-1}) \tag{2.5}$$

for some $0 < \rho < 1$.

Under (A1), which is stronger than plain stationarity and ergodicity of $\{X_k\}$, we can define some parameters needed for the formulation of our results. First of all, let $X_m^n = (X_m, \ldots, X_n)$ for $m < n$, and let for every $n \ge 1$ the nth order probability distribution for $\{X_k\}$ be

$$P(X_1^n) = \Pr\{X_k = x_k, 1 \le k \le n, x_k \in \mathcal{A}\}.$$

Then, the *entropy* of $\{X_k\}$ is defined in a standard manner as (cf. [5])

$$h = \lim_{n\to\infty} \frac{E \log P^{-1}(X_1^n)}{n}.$$

The next three parameters are well defined under our assumption (A1) (cf. [10], [30]).

DEFINITION 3. RÉNYI'S ORDER ENTROPY. For $-\infty \le b \le \infty$, define the bth *order Rényi entropy* as

$$h_2^{(b)} = \lim_{n\to\infty} \frac{\log(E\{P^b(X_1^n)\})^{-1}}{(b+1)n} = \lim_{n\to\infty} \frac{\log\left(\sum_{X_1^n} P^{b+1}(X_1^n)\right)^{-1/(b+1)}}{n}. \tag{2.6}$$

In particular, we set

$$h_1 = \lim_{b\to-\infty} h_2^{(b)} \quad \text{and} \quad h_3 = \lim_{b\to\infty} h_2^{(b)}.$$

Note that by the *inequality on means* [28], we can equivalently express the last two parameters as follows

(2.7a)
$$h_1 = \lim_{n\to\infty} \frac{\max\{\log P^{-1}(X_1^n),\, P(X_1^n) > 0\}}{n} = \lim_{n\to\infty} \frac{\log(1/\min\{P(X_1^n),\, P(X_1^n) > 0\})}{n},$$

(2.7b)
$$h_3 = \lim_{n\to\infty} \frac{\min\{\log P^{-1}(X_1^n),\, P(X_1^n) > 0\}}{n} = \lim_{n\to\infty} \frac{\log(1/\max\{P(X_1^n),\, P(X_1^n) > 0\})}{n},$$

as already defined in Pittel [30]. Note also that $h_3 \le h_2^{(b)} \le h \le h_1$. (Actually, the Rényi entropies are defined as $(b+1)/b \cdot h_2^{(b)}$, but it is more convenient for us to use definition (2.6).)

Remark 1.

(i) *Bernoulli model.* In this widely used model (cf. [4], [6], [9], [11], [16], [17], [23], [31], [36], and [37]), symbols from the alphabet Σ are generated independently, that is, $P(X_1^n) = P^n(X_1^1)$. In particular, the ith symbol from the alphabet Σ is generated according to the probability p_i, where $1 \le i \le V$ and $\sum_{i=1}^V p_1 = 1$. Then, $h = \sum_{i=1}^V p_i \log p_i^{-1}$ ([5]), and the Rényi entropies become $h_1 = \log(1/p_{\min})$, $h_3 = \log(1/p_{\max})$, and $h_2^{(b)} = 1/(b+1)\log(1/P_b)$ where $p_{\min} = \min_{1\le i\le V}\{p_i\}$, $p_{\max} = \max_{1\le i\le V}\{p_i\}$, and $P_b = \sum_{i=1}^V p_i^{b+1}$. The probability P_b can be interpreted as the probability of a match of $b+1$ strings in a given position (cf. [36]).

(ii) *Markovian model.* In this model (cf. [18], [21], [30], [34]), the sequence $\{X_k\}$ forms a stationary Markov chain, that is, the $(k+1)$st symbol in $\{X_k\}$ depends on the previously selected symbol, and the transition probability becomes $p_{i,j} = \Pr\{X_{k+1} = j \in \Sigma | X_k = i \in \Sigma\}$. Clearly, $P(X_1^n) = P(X_1)\Pr\{X_2|X_1\}\cdots\Pr\{X_n|X_{n-1}\}$. It is well known (cf. [5]) that the entropy h can be computed as $h = -\sum_{i,j=1}^V \pi_i p_{i,j}\log p_{i,j}$, where π_i is the stationary distribution of the Markov chain. The other quantities are a little harder to evaluate. Szpankowski [36] and Pittel [30] for $b = 1$, evaluated the height of a regular tries with Markovian dependency, and show that the bth order Rényi entropy $h_2^{(b)}$ is a function of the largest eigenvalue θ_b of the matrix $\mathbf{P}_{[b+1]} = \mathbf{P}\circ\mathbf{P}\ldots\circ\mathbf{P}$, where $\mathbf{P} = \{p_{i,j}\}_{i,j=1}^V$ is the transition matrix of the underlying Markov chain and $\circ$ represents the Schur product of $b+1$ matrices $\mathbf{P}$ (i.e., elementwise product). More precisely, $h_2^{(b)} = 1/(b+1)\cdot\log\theta_b^{-1}$. With respect to h_1 and h_3 we need a result from digraphs (cf. Romanovski [33], Karp [22]). Consider a digraph on $\Sigma = \{1,\ldots,V\}$ with weights equal to $-\log p_{i,j}$, where $i, j \in \Sigma$. Define a cycle $\mathcal{C} = \{\omega_1, \omega_2, \ldots, \omega_v, \omega_1\}$ for some $v \le V$ such that $\omega_i \in \Sigma$, and let $\ell(\mathcal{C}) = -\sum_{i=1}^v \log(p_{\omega_i,\omega_{i+1}})$ (with $\omega_{v+1} = \omega_1$) be the total weight of the cycle $\mathcal{C}$. The quantities $\min_{\mathcal{C}}\{\ell(\mathcal{C})/v\}$ and $\max_{\mathcal{C}}\{\ell(\mathcal{C})/v\}$ are known as the minimum and maximum cycle mean, respectively. Karp [22] showed how to compute them efficiently. Clearly, $h_1 = \min_{\mathcal{C}}\{\ell(\mathcal{C})/|\mathcal{C}|\}$ and $h_3 = \max_{\mathcal{C}}\{\ell(\mathcal{C})/|\mathcal{C}|\}$.

2.2. Formulation of main results. Now, we present our first main result concerning the typical height and the shortest path, which is further used to prove our next findings. The proof of Theorem 1 is delayed till §3, except part (ii) regarding PAT trees, which is a simple consequence of part (i), and it is proved in Remark 2 (iii) below. For the reader's convenience, we recall that we write $X_n \to \beta$ (pr.) for a sequence of random variables X_n and a constant β if for every $\epsilon > 0$ the following holds: $\lim_{n\to\infty}\Pr\{|X_n/\beta - 1| > \epsilon\} = 0$; and similarly $X_n \to \beta$ (almost surely) if for every $\epsilon > 0$ we have $\lim_{N\to\infty}\Pr\{\sup_{n>N}|X_n/\beta - 1| > \epsilon\} = 0$. A sufficient condition for the almost sure convergence can be obtained from the Borel–Cantelli lemma (cf. [5]). In particular, the following suffices for (a.s.): $\sum_{n=0}^\infty \Pr\{|X_n/a - 1| > \epsilon\} < \infty$.

THEOREM 1. *Let $\{X_k\}$ be a stationary ergodic sequence satisfying the strong α-mixing condition as in* (A2) *together with $h_1 < \infty$ and $h_2 > 0$.*

(i) b-SUFFIX TREES. *Fix b. Then*

(2.8a) $$\lim_{n\to\infty}\frac{s_n^{(b)}}{\log n}=\frac{1}{h_1}\quad (a.s.)$$

(2.8b) $$\lim_{n\to\infty}\frac{\tilde{s}_n^{(b)}}{\log n}=\frac{1}{h_1}\quad (pr.)$$

provided

(2.9) $$\alpha(d)=O(n^\beta\rho^d)$$

for some constants $0<\rho<1$ *and* $\beta>0$. *For the height* $H_n^{(b)}$ *we have*

(2.10) $$\lim_{n\to\infty}\frac{H_n^{(b)}}{\log n}=\frac{1}{h_2^{(b)}}\quad (a.s.)$$

provided the mixing coefficient $\alpha(d)$ *fulfills the following*

(2.11) $$\sum_{d=0}^{\infty}\alpha^2(d)<\infty.$$

(ii) COMPACT SUFFIX TREE. *Almost sure behavior of PAT follows from the* (*a.s.*) *behavior of b-suffix trees by taking in* (2.8) *and* (2.10) *the limit as* $b\to\infty$, *that is,*

(2.12) $$\lim_{n\to\infty}\frac{s_n^{PAT}}{\log n}=\frac{1}{h_1}\quad (a.s.)\quad \lim_{n\to\infty}\frac{H_n^{PAT}}{\log n}=\frac{1}{h_3},$$

provided (P) *holds together with condition* (2.9) *for* s_n^{PAT} *and condition* (2.11) *for* H_n^{PAT}, *respectively.*

Our next main results deal with the depths $D_n^{(b)}$, $M_n^{(b)}$, $L_n^{(b)}(m)$ (for fixed m), and the depth of insertion $L_n^{(b)}$. The proof of Theorem 2 is presented in §3.3 except part (iii), which is discussed in Remark 2 (ii).

THEOREM 2. *Let* $\{X_k\}$ *be a stationary ergodic and mixing sequence in the strong sense of* (A2), *and let* (2.9) *hold too. Assume also that* $1\le b<\infty$.

(i) CONVERGENCE IN PROBABILITY. *For* $h<\infty$ *and m fixed, we have*

(2.13) $$\lim_{n\to\infty}\frac{L_n^{(b)}(m)}{\log n}=\lim_{n\to\infty}\frac{L_n^{(b)}}{\log n}=\lim_{n\to\infty}\frac{D_n^{(b)}}{\log n}=\frac{1}{h}\quad (\text{pr.}).$$

The same holds for the compact suffix tree provided (2.5) *in* (P) *is fulfilled* (*i.e., we may take* $b\to\infty$ *in the above*).

(ii) ALMOST SURE CONVERGENCE. *Let, in addition, the probability* $P(B_n)$ *of "bad states" in the Shannon–McMillan–Breiman Theorem* (*more precisely: in the so-called asymptotic equipartition property*) [5] *be summable* (*cf.* §3.3), *that is,*

(2.14) $$\sum_{n=1}^{\infty}P(B_n)<\infty.$$

Then, for fixed m

(2.15) $$\lim_{n\to\infty}\frac{L_n^{(b)}(m)}{\log n}=\frac{1}{h}\quad (a.s..)$$

The above is true also for the compact suffix tree provided (2.5) *in* (P) *is satisfied.*

(iii) ALMOST SURE OSCILLATIONS. *As in* (ii) *we assume strong mixing condition* (2.4b) *together with* $h_1 < \infty$ *and* $h_2 > 0$. *Then, for* $b < \infty$ *we have the following result concerning the depth of insertion and the typical depth*

$$\liminf_{n\to\infty} \frac{L_n^{(b)}}{\log n} = \frac{1}{h_1} \quad \text{(a.s.)} \quad \limsup_{n\to\infty} \frac{L_n^{(b)}}{\log n} = \frac{1}{h_2^{(b)}}. \tag{2.16a}$$

$$\liminf_{n\to\infty} \frac{D_n^{(b)}}{\log n} = \frac{1}{h_1} \quad \text{(a.s.)} \quad \limsup_{n\to\infty} \frac{D_n^{(b)}}{\log n} = \frac{1}{h_2^{(b)}}. \tag{2.16b}$$

For the compact suffix tree, (2.16a) *and* (2.16b) *hold with* $h_2^{(b)}$ *replaced by* h_3, *that is, we formally obtain almost sure behavior for PAT by taking* $b \to \infty$ *and assuming* (2.5) *of* (P).

Remark 2.

(i) *How restrictive are conditions* (2.9) *and* (2.14)? Let us first deal with (2.9). Note that (2.9) holds in many interesting cases including the Bernoulli model and the Markovian model. Naturally, (2.9) is true for the Bernoulli model since in this case $\alpha(d) = 0$. In the Markovian model, it is known (cf. [5]) that for a finite state Markov chain the coefficient $\alpha(d)$ decays exponentially; that is, for some $c > 0$ and $\rho < 1$ we have $\alpha(d) = c\rho^d$, as needed for (2.9). Regarding (2.14), we know that it holds at least for the Bernoulli and Markovian models but generally not for all ergodic stationary sequences (cf. [10]). We believe that (2.14) is included in (2.9). In passing, we also note that condition (2.5) holds in both of the models above. In the Markovian model, however, one needs the additional assumption that all transition probabilities are positive and strictly smaller than one.

It should be mentioned, however, that condition (2.9) probably cannot be improved. This is due to recent results of Shields [34] who proved that the normalized external path length $E_n^{(1)}/n \log n$ converges almost surely to $1/h$ in the Bernoulli and Markovian models. But, the author of [34] also constructed an ergodic stationary (non-Markovian) sequence for which the external path length $E_n^{(b)}$ does not converge even in probability. The same construction can be used to show nonconvergence results for other tree parameters considered in this paper. Hence, some kind of restrictions for the mixing coefficient $\alpha(d)$ is necessary.

(ii) *How do we prove part* (iii) *of Theorem* 2? One can view the behavior of $L_n^{(b)}(m)$ and $L_n^{(b)}$ as a surprise. The main reason for the oscillation of $L_n^{(b)}$ is a "tiny" unbalance in the height and the shortest feasible path discovered in Theorem 1. The almost sure behavior of $L_n^{(b)}(m)$ is guaranteed by the fact that it is a nondecreasing sequence. In passing, we note that the only b-suffix tree that has (a.s.) limit for the depth of insertion $L_n^{(b)}$ is the PAT tree with the *symmetric* alphabet (i.e., $p_i = 1/V$ for $1 \le i \le V$). Indeed, in this case by Theorem 2 (iii) $\lim_{n\to\infty} L_n^{PAT}/\log n = \log V$ (a.s.).

To prove formally Theorem 2 (iii) for $L_n^{(b)}$ we argue as in Pittel [30] (cf. [38]). Provided Theorem 1 *is granted*, we note that almost surely $L_n^{(b)} = H_n^{(b)}$ whenever $H_{n+1}^{(b)} > H_n^{(b)}$, which happens infinitely often (a.s.) since $H_n^{(b)} \to \infty$ (a.s.), and $\{X_k\}$ is an ergodic sequence. This establishes the lim sup part of $L_n^{(b)}$. For the lim inf of $L_n^{(b)}$ we consider the shortest feasible path $s_n^{(b)}$ and repeat the above arguments. The same is true for the typical depth $D_n^{(b)}$ since it represents the length of a *randomly* selected external node, so $\tilde{s}_n^{(b)} \le D_n^{(b)} \le H_n^{(b)}$. But, $s_n^{(b)} = \tilde{s}_n^{(b)}$ infinitely often; hence, (2.15) follows from Theorem 1, too. Note that $\tilde{s}_n^{(b)}$ is *not* a monotone sequence, the property needed to estimate the almost sure convergence of $s_n^{(b)}$ (for more details see §3.1).

(iii) *Compact suffix tree as a limit of b-suffix tree.* We prove now results for PAT trees provided the corresponding results for b-suffix trees are true (see §3). We are not able to prove in general that for any parameter (appropriately normalized) of b-suffix tree, say $P_n^{(b)}$, its corresponding parameter P_n^{PAT} of the PAT tree can be obtained as a limit when b tends to infinity as $n \to \infty$. However, we conjecture that there exists a sequence $a_n = o(n)$ (e.g., in the case of parameters discussed in this paper we have $a_n \sim \log n$) such that

$$\lim_{n\to\infty} P_n^{PAT}/a_n = \lim_{b\to\infty}\lim_{n\to\infty} P_n^{(b)}/a_n. \tag{2.17a}$$

(The condition $a_n = o(n)$ seems to be important since the above does not hold for the size of b-suffix trees, i.e., number of internal nodes, which grows asymptotically as n/h while for the PAT tree the size is $O(n)$; e.g., for the binary alphabet the size is $n-1$. However, we can easily give a formal proof of this fact for every parameter discussed in this section. We first consider all parameters except the height. Using the *Sample Path Theorem* of the stochastic dominance relationship [35], we can prove that $P_n^{(b)}$ is a decreasing sequence with respect to b. Hence, in particular,

$$\lim_{n\to\infty} \frac{P_n^{PAT}}{\log n} \le \lim_{n\to\infty} \frac{P_n^{(1)}}{\log n}. \tag{2.17b}$$

This immediately establishes the upper bound part of (2.17a) for the above parameters (excluding the height). For the height H_n^{PAT}, following Pittel [30] we note that the event $\{H_n^{PAT} > k+b\}$ implies that there exists a set of b suffixes such that all of them share the same first k symbols. In other words, the event $\{H_n^{PAT} > k+b\}$ implies $\{H_n^{(b)} > k\}$. Therefore,

$$\lim_{n\to\infty} \frac{H_n^{PAT}}{\log n} \le \lim_{b\to\infty}\lim_{n\to\infty} \frac{H_n^{(b)}}{\log n} = \frac{1}{h_3}. \tag{2.17c}$$

This completes the upper bound in (2.17a).

For the lower bound, we use condition (2.5) of (P). We need a separate discussion for every parameter. Following Pittel [30], for the height and the shortest path we argue as follows. We try to find a path in a suffix tree such that its length is (a.s.) asymptotically equal to $\log n/h_3$ and $\log n/h_1$, respectively. But this is immediate from (2.7a) and (2.7b), and Pittel's [30] Lemma 2. For the depth, we consider a path for which the initial segment of length $O(\log n)$ is such that all nodes are branching (i.e., no unary nodes occur in it). Naturally, such a path after compression will not change, and the depth in the compact suffix tree is at least as large as the length of this path. Copying our arguments from §3 and using Pittel [30], we establish that almost surely such a path is at least $\log n/h$, which completes the lower bound arguments in the proof for the depth. The details are left for the interested reader.

Despite our formal proof, it is important to understand intuitively why a compact suffix tree can be considered as a limit of b-suffix trees as $n \to \infty$. There are at least three reasons supporting this claim: (1) b-suffix trees do not possess unary nodes in any place that is b levels above the fringe of the noncompact suffix tree (cf. Fig. 1); (2) unary nodes tend to occur more likely at the bottom of a suffix tree, and it is very unlikely in a typical suffix tree to have unary nodes close to the root (e.g., in the Bernoulli model the probability that the root is unary node is equal to $\sum_{i=1}^{V} p_i^n$); (3) on a typical path the compression is of order of $O(1)$. For example, comparing the depth of regular tries and PATRICIA we know that $ED_n^P - ED_n^T = O(1)$ [36], [37], but for the height we have $EH_n^P - EH_n^T \sim \log n$ [30], and therefore, we can expect trouble only with the height. This is, in fact, confirmed by our analysis.

(iv) *Finite strings.* In several computer science applications (cf. [1]–[3], [9], [12]) the string $\{X_k\}_{k=1}^n$ has a finite length n, and is terminated by a special symbol that does not belong to the alphabet Σ, e.g., $X\$$ with $\$ \notin \Sigma$. Most of our results, however, can be directly applied to such strings. Let s'_n, H'_n, and D'_n denote the shortest feasible path, the height, and the depth in a suffix tree (b-suffix tree or compact suffix tree) built over such a finite word, respectively. Then, it is easy to see that $s'_n = 1$, but the other two parameters have exactly the same asymptotics as for the infinite string case, that is, $H'_n/\log n \sim 1/h_2^{(b)}$ (a.s.) and $D'_n/\log n \sim 1/h$ (pr.) under hypotheses of Theorems 1 and 2. Indeed, assume for simplicity $b = 1$ and define new self-alignments C'_{ij} as $C'_{ij} = \min\{C_{ij}, n - i, n - j\}$, where C_{ij} is the self-alignment between the i and j suffixes for the *infinite* string $\{X_k\}_{k=1}^{\infty}$. But, our analysis reveals that only the last $O(\log n)$ suffixes may have any impact on the self-alignments C'_{ij}. Hence, building a suffix tree from the first $n' = n - O(\log n)$ suffixes will lead to the same asymptotics as for an infinite string. Details are left to the interested reader.

2.3. Applications and further discussions. Theorems 1 and 2 find several applications in combinatorial problems on words, data compression, and molecular biology. In general, our findings can be used widely in problems dealing with repeated patterns and other regularities on strings. As an illustration, we solve some problems on words using Theorem 2. Two of them deal with data compression, and the others concern pattern matching. The first data compression example solves the conjecture of Wyner and Ziv [43] and was already reported in Szpankowski [38] while here we present some further extensions. The second one identifies the (a.s.) behavior of the block length in the well-known parsing algorithm of Lempel and Ziv [25].

PROBLEM 1. *Wyner–Ziv conjecture for data compression schemes.* The following idea is behind most data compression schemes. Consider a "data base" sequence of length n, which is known to both the sender and the receiver. Instead of transmitting the next L_n symbols (not in the data base), the sender can "look backward" into the data base and verify whether these L_n symbols have already appeared in the data base. If this is the case, then the sender transmits only the location of these L_n symbols in the data base and the length of L_n. More precisely, let the data base be represented by a subsequence $\{X_k\}_{k=1}^{n}$ of a stationary ergodic sequence $\{X_k\}_{k=1}^{\infty}$. For every n, let L_n be the smallest integer $L > 0$ such that $X_m^{m+L} \neq X_{n+1}^{n+1+L}$ for all $1 \le m \le n$. Wyner and Ziv [43] asked about almost sure behavior of L_n. The authors of [43] proved that $L_n \sim \log n/h$ in probability (pr.), and they conjectured that this can be extended to the almost sure (a.s.) convergence. Szpankowski in [38] showed that the parameter L_n is equal to the depth of insertion $L_n^{(1)}$ in a noncompact suffix tree ($b = 1$). Hence, the convergence in probability of $L_n/\log n$ is demonstrated in Theorem 1(i). But our Theorem 2(iii) settles the Wyner–Ziv conjecture in the negative (in the so-called right domain asymptotics; see for details [38]), and we know that $L_n/\log n$ does not converge (a.s.) but rather oscillates between $1/h_1$ and $1/h_2^{(1)}$.

In the same paper, Wyner and Ziv [43] considered another quantity, namely, N_ℓ that can be defined as the smallest N such that $X_1^\ell = X_N^{N+\ell-1}$ (i.e., the word of length ℓ is found for the first time in a data base of size N_ℓ). Using the suffix tree representation of the sequence $\{X_k\}_{k=1}^{N_\ell}$ one can express N_ℓ in terms of the depth of the associated suffix tree. Indeed, N_ℓ is the size of a suffix tree for which the depth of the first suffix is equal to ℓ, that is, in our notation $L_{N_\ell}^{(1)}(1) = \ell$. But, by (2.15) of Theorem 2(ii), we have $\ell/\log N_\ell \to 1/h$ (a.s.); hence,

$$\lim_{\ell\to\infty} \frac{\log N_\ell}{\ell} = h \quad \text{(a.s.)}$$

provided (2.9) holds. This settles in the positive the second conjecture of Wyner and Ziv [43] for the Markovian model. (See also [29].)

PROBLEM 2. *Block length in the Lempel–Ziv parsing algorithm.* The heart of the Lempel–Ziv compression scheme is a method of parsing a string $\{X_k\}_{k=1}^{n}$ into blocks of different words. The precise scheme of parsing the first n symbols of a sequence $\{X_k\}_{k=1}^{\infty}$ is complicated and can be found in [25]. The main idea of the parsing is to divide the sequence into pairwise distinct blocks such that each block that occurs in the parsing has already been seen somewhere to the left (overlapping is allowed as in Grassberger [15]). For example, for $\{X_k\} = 110101001111\ldots$ the parsing looks like (1)(10)(10100)(111)(1...), that is, the first block has length one, the second block length is two, the next one is of length five since $X_2^5 = X_4^7$, and so on. In Fig. 2, we show how to perform the parsing using a sequence of noncompact suffix trees (cf. [15]). Note that the length of a block is a subsequence of depth of insertions $L_{n_k}^{(1)}$. More precisely, if ℓ_n is the length of the nth block in the Lempel–Ziv parsing algorithm, then Fig. 2 suggests the following relationship $\ell_n = L^{(1)}_{\sum_{k=0}^{n-1}\ell_k}$. For example, in Fig. 1 we have $\ell_0 = L_0^{(1)} = 1$, $\ell_1 = L_1^{(1)} = 2$, $l_2 = L^{(1)}_{\ell_0+\ell_1} = L_3^{(1)} = 5$, and $\ell_3 = L_{1+2+5} = 3$, and so forth. To obtain (a.s.) behavior of the block length ℓ_n, we note that

$$\lim_{n\to\infty}\frac{\ell_n}{\log n} = \lim_{n\to\infty}\frac{L_{\sum_{k=0}^{n-1}\ell_k}}{\log\left(\sum_{k=0}^{n-1}\ell_k\right)}\cdot\frac{\log\left(\sum_{k=1}^{n-1}\ell_k\right)}{\log n}. \tag{2.18}$$

We first estimate the second term in (2.18). One immediately obtains

$$1 \le \frac{\log\left(\sum_{k=1}^{n-1}\ell_k\right)}{\log n} \le \frac{\log\left(\sum_{m=0}^{n}L_n^{(1)}(m)\right)}{\log n} \to 1 \quad \text{(a.s.)},$$

where the right-hand side of the above is a direct consequence of a result concerning the external path length $E_n^{(1)}$ proved in Shields [34] (in fact, a slight extension of our proof of Theorem 2 (ii) leads to the same result). Then, by (2.18) (cf. also [38]) we obtain the following corollary.

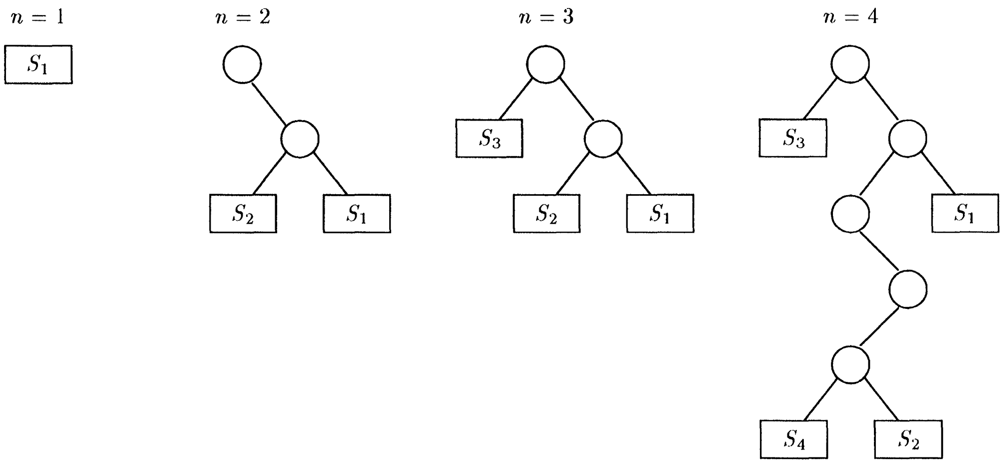


FIG. 2. *First four suffix trees used to parse the sequence* $X = 110101001111\ldots$.

COROLLARY 3. *Let* $\{X_k\}_{k=1}^{\infty}$ *be a strongly mixing stationary sequence as in* (A2) *with the mixing coefficient* $\alpha(d)$ *satisfying* (2.9) *and* (2.14). *Then almost surely*

$$\frac{1}{h_1} \le \liminf_{n\to\infty}\frac{\ell_n}{\log n} \le \limsup_{n\to\infty}\frac{\ell_n}{\log n} \le \frac{1}{h_2^{(1)}}$$

provided $h_1 < \infty$ *and* $h_2^{(1)} > 0$.
We conjecture that the lim sup and lim inf are attained at $1/h_2^{(1)}$ and $1/h_1$ (a.s.), respectively.

PROBLEM 3. *String matching algorithms.* Recently, Chang and Lawler [9] demonstrated how to use PAT trees to design *practical* and still efficient algorithms for approximate string matching algorithms. They formulated several conclusions based on a heuristic analysis of PAT trees under the *symmetric* Bernoulli model. Our Theorems 1 and 2 immediately generalize results of [9] to a more general probabilistic model and additionally provide stronger results. For example, consider the exact string matching algorithm (cf. §2.3 in [9]) in which we search for all occurrences of the pattern string P of length m in the text string T of length n. The heart of the Chang–Lawler algorithm is an analysis of $d_{m,n}$ that is the length of a substring of the text T, which is not a substring of the pattern P. This can be verified by building a compact suffix tree for P, and then comparing substrings of T with suffixes of P. But then, one may observe that $d_{m,n}$ is equivalent to the typical depth D_n^{PAT} in such a suffix tree, and therefore, $d_{m,n} \sim (1/h)\log m$ (pr.). This further implies that the complexity C_n of the algorithm becomes $C_n \sim O(n/(hm)\log m)$ (pr.), which is a stronger version of the Chang–Lawler result for a more general probabilistic model. In passing, we note that our findings can be also used to estimate the time-complexity for the Knuth–Morris–Pratt algorithm [24] and the Boyer–Moore algorithm [7]. Several other approximate string matching algorithms can be analyzed in a similar manner. The reader is referred to Apostolico and Szpankowski [4] for more algorithmic examples.

PROBLEM 4. *A molecular biology problem: Rare subsequences.* Biologists often need a (shortest) subsequence of a sequence $\{X_k\}_{k=1}^n$ (e.g., DNA or RNA) that determines (identifies) uniquely this sequence or that occurs very rarely in the underlying sequence. Sometimes, they also need to find the shortest subsequence, which does not occur in the sequence (cf. [13]). (In fact, biologists allow gaps, but we will not treat this case here.) These problems can be solved with a clever use of the suffix tree data structure (no gaps are allowed!). We illustrate here only how our result can be used to solve the latter problem. Call the shortest subsequence that does not occur in the underlying sequence as U_n. A question arises regarding what is the typical length of U_n and how to construct U_n. It should be clear that the length of U_n cannot be too short (e.g., single characters or pairs of characters occur too often!). If one builds a noncompact suffix tree of $\{X_k\}_{k=1}^n$, then certainly all subsequences up to level $s_n^{(1)}$ occur in $\{X_k\}$ since the suffix tree is a complete tree up to this level. Hence, the length of U_n should be equal to $s_n^{(1)}$, and (a.s.) its length is asymptotically equal to $(1/h_1)\log n$. Moreover, U_n can be discovered by taking any subsequence that leads to the closest "hole" in the associated suffix tree.

Finally, we would like to offer some remarks regarding further implications and generalizations of our results.

Remark 3.

(i) *Convergence in distributions.* In this paper, we deal only with the almost sure convergence. One may ask about the limiting distribution of $L_n^{(b)}$, $H_n^{(b)}$, and so forth. At this time, we have very limited knowledge about the limiting distribution of the above parameters. In fact, only the typical depth in the Bernoulli model of noncompact suffix tree ($b = 1$) was analyzed in the past. Jacquet and Szpankowski [19] proved that the distribution $F_n^T(x)$ of the typical depth in *independent tries* and the distribution $F_n^S(x)$ of the typical depth $D_n^{(1)}$ in suffix trees, do not differ too much. More precisely, in [19] it is proved that for large n there exist such $\beta > 1$ and $\epsilon > 0$ that

$$|F_n^T(k) - F_n^S(k)| = O\left(\frac{1}{n^\epsilon \beta^k}\right). \tag{2.19}$$

This establishes similarities between a trie and a noncompact suffix tree. Therefore, using well-known results for independent tries (cf. [31]), it is easy to show that for an asymmetric alphabet Σ, the normalized depth $(D_n^{(1)} - ED_n^{(1)})/varD_n^{(1)}$ converges *in distribution* to the standard normal distribution $\mathcal{N}(0, 1)$ with mean and variance as below

$$ED_n^{(1)} = \frac{1}{h} \cdot \left\{ \log n + \gamma + \frac{R}{2h} \right\} + P_1(\log n) + O\left(\frac{1}{n^\epsilon}\right), \tag{2.20a}$$

$$varD_n^{(1)} = \frac{R - h^2}{h^3} \log n + C + P_2(\log n) + O\left(\frac{1}{n^\epsilon}\right), \tag{2.20b}$$

for some $\epsilon > 0$, where $R = \sum_{i=1}^V p_i^2 \log p_i$, and $P_1(x)$, $P_2(x)$ are fluctuating periodic functions with small amplitudes, and an explicit formula for the constant C can be found in [37]. In the symmetric case, the variance becomes

$$varD_n^{(1)} = \frac{\pi^2}{6 \log^2 V} + \frac{1}{12} + O\left(\frac{1}{n^\epsilon}\right). \tag{2.20c}$$

Moreover, in the symmetric case the distribution of $D_n^{(1)}$ is no longer asymptotically normal, but rather resembles one of extreme distribution. More precisely, in this case we have (cf. [31])

$$\lim_{n\to\infty} \sup_x |\Pr\{D_n^{(1)} \le x\} - \exp(-nV^{-x})| = 0. \tag{2.20d}$$

We conjecture that the same type of limiting distributions can be obtained in the Markovian model. The is due to the fact that (2.19) seems to hold in the Markovian case. If so, we can apply the recent result of Jacquet and Szpankowski [18] regarding the limiting distribution of the depth for the Markovian model of independent tries. Furthermore, one may investigate the limiting distribution for the height and the external path length. We conjecture that b-suffix trees do not differ too much from b-tries in the sense of (2.19), and therefore, the limiting law for the height can be obtained from the one for b-tries (cf. [31]) and so on.

The compact suffix tree is more intricate. Only very recently some results regarding limiting law for the depth in PATRICIA have been obtained (cf. [32]). Using this result, Jacquet, Rais, and Szpankowski [20] proved that the limiting distribution for the depth in PAT tree under an *asymmetric* Bernoulli model is asymptotically normal. There is, however, no result regarding the limiting law of the height. These seem to be difficult problems.

(ii) *How well is a suffix tree balanced?* In the worst case, a suffix tree may degenerate, and the worst-case height can be as much as n. But, our analysis indicates that this happens very, very rarely. In fact, our Theorem 2 shows that the typical depth of a suffix is equal to $(1/h)\log n$ (pr.). The best balanced tree built over n external nodes is a complete tree (cf. [1]), and the depth for every external node in such a complete tree is equal to $\log_V n$. We note that for the symmetric alphabet a typical shape of suffix tree resembles that of a complete tree since the depth $D_n^{(b)}$ with high probability is equal to $\log_V n$ and almost surely is not greater than $H_n^{(b)} \sim (1 + 1/b)\log_V n$ but not smaller than $s_n \sim \log_V n$. Such a tree can be called highly balanced (in a probability sense), and, as our analysis shows, there is no need, in most practical cases, for additional rebalancing of this tree in order to assure a nice behavior in the worst case, as is done in AVL-tree and other balanced trees.

3. Analysis and proofs. We first present a formal proof of Theorem 1 concerning the height $H_n^{(b)}$ and the shortest feasible path $s_n^{(b)}$. Then, we establish parts (i) and (ii) of Theorem

2 for the typical depth $D_n^{(b)}$, and the depth of the mth suffix $L_n^{(b)}(m)$. We remind the reader that Theorem 2 (iii) was already proved in Remark 2 (ii), and the compact suffix tree was discussed in Remark 2 (iii). Therefore, hereafter, we fix $b < \infty$. Also, for simplicity of presentation we drop the upper index b in the notation of the tree parameters (e.g., we write H_n instead of $H_n^{(b)}$).

Throughout the proof we use a technique that encompasses the mixing condition and another technique called the *string-ruler approach* that was already applied by Pittel in his seminal paper [30], and extended by Jacquet and Szpankowski [19] (cf. [38]). The idea of the string-ruler approach is to measure a correlation between words by another *nonrandom* word w belonging to a set of words $\mathcal{W}$. Usually, we deal with fixed length rulers w_k where k is the length of the string-ruler. Let $\mathcal{W}_k$ be the set of all strings w_k, that is, $\mathcal{W}_k = \{w \in \Sigma^k : |w| = k\}$, where $|w|$ is the length of w. We write w_k^ℓ to mean a concatenation of ℓ strings w_k from $\mathcal{W}_k$, and if $X_m^{m+k} = w_k$, then we denote $P(w_k) = P(X_m^{m+k})$. Finally, we adopt the following rule regarding sums over a set of string-rulers: if $f(w_k)$ is a function of w_k, then $\sum_{\mathcal{W}_k} f(w_k) = \sum_{w_k \in \mathcal{W}_k} f(w_k)$, where the sum is over all strings w_k of length k.

The usefulness of the string-ruler approach stems from the fact that we can express the self-alignment $C_{i_1,\dots,i_{b+1}}$ in terms of w_k. The following lemma is of prime importance to the analysis of suffix trees and other combinatorial structures on words.

LEMMA 4. *Let $d_1, \dots, d_b$ and k be such that*

$$d_0 = 0 \le d_1 \le \cdots \le d_i \le k \le d_{i+1} \le \cdots \le d_b. \tag{3.1}$$

Define d as the greatest common divisor of $\{d_i\}_{i=1}^b$, that is, $d = \gcd(d_1, \dots, d_b)$. Then, the self-alignment $C_{1,1+d_1,\dots,1+d_1+\cdots+d_b}$ satisfies

$$\begin{aligned}\Pr\{C_{1,1+d_1,\dots,1+d_1+\cdots+d_b} \ge k\} &= \sum_{\mathcal{W}_d} P\left(w_d^{\lfloor \frac{k}{d}\rfloor + \frac{d_1+\cdots+d_i}{d}} \overline{w}_d (w_d^{\lfloor \frac{k}{d}\rfloor}\overline{w}_d)^{b-i}\right)\\ &= \sum_{\mathcal{W}_d} P\left(w_d^{(b+1-i)\lfloor \frac{k}{d}\rfloor + \frac{d_1+\cdots+d_i}{d}} \overline{w}_d^{b+1-i}\right),\end{aligned} \tag{3.2a}$$

where $\overline{w}_d$ is a prefix of w_d, and $\lfloor x \rfloor$ is the floor function. Two cases are of particular interest, namely: (i) *if $k \le d_1 \le \cdots \le d_b$, then*

$$\Pr\{C_{1,1+d_1,\dots,1+d_1+\cdots+d_b} \ge k\} = \sum_{\mathcal{W}_k} P(w_k^{b+1}); \tag{3.2b}$$

(ii) *if $d_i \le \cdots \le d_b \le k$, then*

$$\Pr\{C_{1,1+d_1,\dots,1+d_1+\cdots+d_b} \ge k\} = \sum_{\mathcal{W}_d} P\left(w_d^{\lfloor \frac{k}{d}\rfloor + \frac{d_1+\cdots+d_b}{d}} \overline{w}_d\right). \tag{3.2c}$$

Proof. It is illustrative to start with $b = 1$. In this case, it is well known [26] that for any pair of suffixes S_1 and S_{1+d} there exists a word w_d such that the common prefix Z_k of length k of S_1 and S_{1+d} can be represented as $Z_k = w_d^{\lfloor k/d \rfloor}\overline{w}_d$. Then (3.2) (in fact (3.2c)) is a simple consequence of the above. The above rule is easy to extend to b suffixes. Let Z_k be the common prefix of length k of the following b suffixes $\{S_1, S_{1+d_1}, \dots, S_{1+d_1+\cdots+d_b}\}$. To avoid heavy notation, we consider three cases separately. If $k \le d_1 \le \cdots \le d_b$, then all suffixes are separated by more than k symbols, so certainly there exists a word w_k such that $Z_k = w_k^{b+1}$, which further implies (3.2b). Let us not consider the case $d_1 \le \cdots \le d_b \le k$; that is, there are mutual overlaps between any two consecutive suffixes. Then, there must exist a word w_d

of length $d = \gcd(d_1, \ldots, d_b)$ such that $Z_k = w_d^{\lfloor k/d \rfloor + (d_1 + \cdots + d_b/d)} \overline{w}_d$, which leads to (3.2c). The general solution (3.2a) is a combination of the above two cases.

Finally, in the proof below we often use another representation for the bth order Rényi entropy, namely (we drop the index b according to our convention), under assumption (A1) we have

$$h_2 = \lim_{n \to \infty} \frac{\log\left(\sum_{\mathcal{W}_n} P^{b+1}(w_n)\right)}{(b+1)n}. \tag{3.3}$$

Indeed, the above is a simple consequence of the weak mixing condition (2.4a) of (A1) and the fact that b is fixed.

3.1. Height of b-suffix trees. We now prove Theorem 1(i) formula (2.10) concerning the (a.s.) behavior of the height. We discuss separately the upper and the lower bounds for the convergence in probability. Finally, (a.s.) convergence is established.

Upper bound. We start with the representation (2.3b) for the height H_n. By Boole's inequality, the tail of the height distribution can be bounded from above by a sum of marginal distributions of the self-alignments. In other words, using Lemma 4 we have

$$\Pr\{H_n \ge k\} = \Pr\{\max_{1 \le d_1, \ldots, d_b \le n} \{C_{1,1+d_1,\ldots,1+d_1+\cdots+d_b}\} \ge k\}$$

$$\le \sum_{d_1=1}^{n} \cdots \sum_{d_b=1}^{n} \Pr\{C_{1,1+d_1,\ldots,1+d_1+\cdots+d_b} \ge k\}$$

$$\le n \sum_{i=1}^{b} \binom{b}{i} k^i n^{b-i} \sum_{\mathcal{W}_d} P\left(w_d^{\lfloor \frac{k}{d} \rfloor + \frac{d_1 + \cdots + d_i}{d}} \overline{w}_d w_k^{b-i} \right).$$

The last sum can be estimated as follows

$$\sum_{\mathcal{W}_d} P\left(w_d^{\lfloor \frac{k}{d} \rfloor + \frac{d_1 + \cdots + d_i}{d}} \overline{w}_d w_k^{b-i} \right) \le^{(A)} c^{b-i} \sum_{\mathcal{W}_k} P(w_k) P^{b+1-i}(w_k)$$

$$\le^{(B)} c^{b-i} \left(\sum_{\mathcal{W}_k} P^{b+1}(w_k) \right)^{(b+1-i)/(b+1)},$$

where the first inequality (A) comes from the strong mixing condition of assumption (A1), and the fact that the set of words of the form $w_d^{\lfloor k/d \rfloor} \overline{w}_d$ is a subset of all words of length k (i.e., $\mathcal{W}_k$). The inequality (B) is a consequence of the well-known *inequality on means* (cf. [28]). So, finally for some constant c, we have

$$\Pr\{H_n \ge k\} \le c \sum_{i=1}^{c} \binom{b}{i} k^i n^{b-i+1} \left(E P^b(w_k) \right)^{(b+1-i)/(b+1)}. \tag{3.4}$$

Now, let $k = \lfloor (1+\epsilon) \log n / h_2 \rfloor$. Then, definition (2.6), identity (3.3), and the above prove the following upper bound

$$\Pr\left\{ H_n \ge (1+\epsilon) \frac{\log n}{h_2} \right\} \le c \sum_{i=1}^{b} \binom{b}{i} k^i n^{b+1-i} \frac{1}{n^{(1+\epsilon)(b+1-i)}}$$

$$\le \frac{c}{n^\epsilon} (k + n^{-\epsilon})^b \sim \frac{c(1+\epsilon)^b \log^b n}{n^\epsilon h_2^b}.$$

This completes our arguments for the upper bound of the height for the convergence in probability. The (a.s.) convergence will be established after the proof of the lower bound.

Lower bound. The lower bound is more intricate. The idea, however, is quite simple. At first, we construct another b-suffix tree with height that is smaller than in the original b-suffix tree, but which resembles independent tries (i.e., strings stored in such a suffix tree are less correlated than in the original b-suffix tree). Secondly, we apply the *second moment method* [36] to the modified suffix tree. The second moment method gives a sharp lower bound for $\Pr\{H_n > k\}$. In particular, using this method, we prove that $\Pr\{H_n > k\} \to 1$ for $k = \lfloor (1-\epsilon)\frac{1}{h_2}\log n \rfloor$.

To fulfill this plan, we start with a construction of the modified b-suffix tree. We partition the sequence X_1^n into m parts each composed of k consecutive symbols followed by a gap of size d. Therefore, the size of each part is $k+d$ and $m = \lfloor n/(k+d) \rfloor$. In the following, we assume that $k = O(\log n)$ as well as $d = O(\log n)$; hence, $m = O(n/\log n)$. We define new strings $Y(1), \ldots, Y(m)$ as $Y(i) = X_{(i-1)(k+d)+1}^{\infty} - X_{ik+(i-1)d+1}^{i(k+d)}$ where "$-$" means deletion, that is, $Y(i)$ is the $((i-1)(k+d)+1)$st suffix of $\{X_k\}$ with a gap of length d between the $(ik+(i-1)d+1)$st symbol and the $(i(k+d))$th symbol. For example, the first string $Y(1)$ consists of the first k symbols followed by all symbols after the $(k+d)$th symbol (the first gap between $k+1$ and $k+d$ is omitted). The second string $Y(2)$ starts at position $k+d+1$ and continues for the next k symbols after which the next d symbols of the second gap are eliminated, and then the strings expand up to infinity. We build a b-suffix tree out of these m strings $Y(1), \ldots, Y(m)$. We denote such a b-suffix tree as $\mathcal{T}_m$ since for a typical sequence $\{X_k\}$ these m strings resemble independent keys in a b-trie; that is, they are weakly dependent on their first k symbols.

We denote by H_m the height of the modified b-suffix tree $\mathcal{T}_m$. Certainly, this height is stochastically smaller than the height H_n in the original tree. This can be proved formally by the *Sample Path Theorem* [35]. As a simple consequence of this fact, we have

$$\Pr\{H_n \ge k\} \ge \Pr\{H_m \ge k\} \quad \text{for} \quad m \le n. \tag{3.5}$$

We estimate the probability $\{H_m \ge k\}$ by the second moment method (cf. [8], [36]). We need some additional notation. Let $\mathbf{i} = (i_1, i_2, \ldots, i_{b+1})$ be a $b+1$ dimensional vector, and define a set D as $D = \{\mathbf{i} : 1 \le i_j \le m \text{ for } 1 \le j \le b+1\}$. Let also $D^2 = D \times D$, which we additionally partition into two sets D_1^2 and D_2^2 such that

$$D_1^2 = \{(\mathbf{i},\mathbf{j}) : (i_1, \ldots, i_{b+1}) \cap (j_1, \ldots, j_{b+1}) = \emptyset\}, \tag{3.6}$$

and D_2^2 contains the other pairs $(\mathbf{i},\mathbf{j})$ of D^2. Now, let us define an event $A_{\mathbf{i}} = \{C_{\mathbf{i}} \ge k\}$, where we use $C_{\mathbf{i}}$ as a short notation for the self-alignment $C_{i_1,\ldots,i_{b+1}}$. Note that $\Pr\{H_m \ge k\} = \Pr\{\cup_{\mathbf{i}\in D} A_{\mathbf{i}}\}$. Then, the second moment method asserts that (cf. [8])

$$\Pr\{H_m \ge k\} = \Pr\{\bigcup_{\mathbf{i}\in D} A_{\mathbf{i}}\} \ge \frac{\left(\sum_{\mathbf{i}\in D}\Pr\{A_{\mathbf{i}}\}\right)^2}{\sum_{\mathbf{i}\in D}\Pr\{A_{\mathbf{i}}\} + \sum_{(\mathbf{i},\mathbf{j})\in D^2}\Pr\{A_{\mathbf{i}} \cap A_{\mathbf{j}}\}}. \tag{3.7}$$

We will show that for $k = \lfloor (1-\epsilon)\frac{1}{h_2}\log n \rfloor$ the right-hand side of (3.7) tends to one, hence also by (3.5) $\Pr\{H_n \ge (1-\epsilon)\frac{1}{h_2}\log n\} \to 1$ as $n \to \infty$, which is the desired inequality.

We must now evaluate the terms in the right-hand side of (3.7). Using the strong α-mixing condition of (A2) and arguing as in the upper bound case, we immediately show that for $k = O(\log n)$ (cf. [38])

$$(m^b - o(b^b))(1-\alpha(d_n))^b EP^b(w_k) \le \sum_{\mathbf{i}\in D}\Pr\{A_{\mathbf{i}}\} \le (m^b - o(m^b))(1+\alpha(d_n))^b EP^b(w_k),$$

where the length of the gap d_n is explicitly shown to be a function on n. The probability $\Pr\{A_{\mathbf{i}} \cap A_{\mathbf{j}}\}$ is more difficult to estimate. However, on the set D_1^2, suffixes of $\mathcal{T}_m$ do not coincide; hence, we have (cf. [38])

$$\sum_{(\mathbf{i},\mathbf{j})\in D_1^2} \Pr\{A_{\mathbf{i}} \cap A_{\mathbf{j}}\} \le (1+\alpha(d_n))^{2b+1} E^2 P^b(w_k).$$

In the set D_2^2, there exists at least one pair of suffixes that is the same for $\mathbf{i}$ and $\mathbf{j}$. For example, if $\mathbf{i} = (1,5)$ and $\mathbf{j} = (1,6)$, then $\Pr\{A_{\mathbf{i}} \cap A_{\mathbf{j}}\} = \sum_{\mathcal{W}_k} P(w_k^3)$, since the first suffix is common to $\mathbf{i}$ and $\mathbf{j}$. In general, the following is true:

$$\begin{aligned}\Pr\{A_{\mathbf{i}} \cap A_{\mathbf{j}}\} &= \sum_{\mathcal{W}_k} P(w_k^{2b+1}) \le^{(A)} c_1 \sum_{\mathcal{W}_k} P^{2b+1}(w_k) \\ &\le^{(B)} c_1 \left(\sum_{\mathcal{W}_k} P^{b+1}(w_k)\right)^{(2b+1)/(b+1)} = c_1 \left(EP^b(w_k)\right)^{2-1/(b+1)},\end{aligned}$$

where (A) follows from the strong mixing conditions, and (B) is a consequence of the following known inequality (cf. [21], [36])

$$\ell \ge r \quad \Rightarrow \quad \left(\sum_{\mathcal{W}_k} P^\ell(w_k)\right)^{1/\ell} \le \left(\sum_{\mathcal{W}_k} P^r(w_k)\right)^{1/r}.$$

Putting everything together, the inequality (3.7) becomes for $k = \lfloor (1-\epsilon)\frac{1}{h_2}\log n \rfloor$

$$\Pr\{H_m \ge k\} \ge \left(\frac{n^{(b+1)(1-\epsilon)}}{m^{b+1}(1-\alpha(d_n))^b} + [1 - O(m^{-1})]\frac{(1+\alpha(d_n))^{2b+1}}{(1-\alpha(d_n))^b} + c\frac{n^{1-\epsilon}}{m}\right)^{-1}.$$

Substituting $m = \Theta(n/\log n)$ and $d_n = \Theta(\log n)$, we finally obtain
(3.8)

$$\Pr\left\{H_m \le (1-\epsilon)\frac{1}{h_2}\log n\right\} \le c_1 \frac{\log^{b+1} n}{n^{(b+1)\epsilon}} + c_2 \frac{\log n}{n^\epsilon} + c_3(2b+1)b\alpha^2(\log n) + O(\alpha^3(d_n)),$$

which proves the lower bound for H_m, and hence, by (3.5) also for our original b-suffix tree. In summary, the upper bound (3.5) and the above lead to the following:

$$\Pr\left\{\left|\frac{H_n}{\log n} - \frac{1}{h_2}\right| \ge \epsilon\right\} \le c_1 \frac{\log^b n}{n^\epsilon} + c_2\alpha^2(\log n) \to 0 \tag{3.9}$$

for some constants c_1 and c_2. This proves $H_n/\log n \to 1/h_2$ (pr.).

Almost sure convergence. The rate of convergence in (3.9) does not yet warrant the application of the Borel–Cantelli lemma to prove almost sure convergence. But due to the fact that H_n is nondecreasing in n and $a_n = \frac{1}{h_2}\log n$ is a slowly increasing function of n, we can establish (a.s.) convergence for the height. Indeed, as in [11] (cf. also [38]), we note that $H_n > a_n$ infinitely often (i.o.) if $H_{2^r} > a_{2^{r-1}}$ (i.o.) in r, and similarly $H_n < a_n$ (i.o.) if $H_{2^r} < a_{2^{r+1}}$ (i.o.). But the latter events hold indeed infinitely often due to (3.9) and the Borel–Cantelli lemma since

$$\sum_{r=0}^{\infty} \Pr\left\{\left|\frac{H_{2^{r\pm1}}}{\log(2^{r\pm1})} - \frac{1}{h_2}\right| \ge \epsilon\right\} < \infty \tag{3.10}$$

provided

$$\sum_{r=0}^{\infty} \alpha^2(r) < \infty.$$

This completes the proof of Theorem 1(i) concerning the height $H_n^{(b)}$ of a b-suffix tree.

3.2. The shortest feasible path of b-suffix trees. For the upper bound we use the fact that $s_n^{(b)}$ is nonincreasing in b; that is, $s_n^{(b)} \leq s_n^{(1)}$. Note that the first s_n levels of any suffix tree are "filled" with internal nodes (i.e., there is no "hole" in the tree up to this level). In other words, up to the level s_n a suffix tree resembles a complete tree. This fact was used in [38] (cf. [30]) to establish the following bound

$$\Pr\left\{s_n^{(1)} > (1+\epsilon)\frac{1}{h_1}\log n\right\} \leq \frac{c}{n^\epsilon}. \tag{3.11}$$

This upper bound holds also for b-suffix trees since the parameter h_1 does not depend on b.

The rest of this section is devoted to the lower bound for $s_n^{(b)}$. As in the case of the height, we drop hereafter the upper index b in the notation of the shortest feasible path. We proceed as in the case of the lower bound for the height; that is, we define the modified suffix tree $\mathcal{T}_m$ composed of m weakly dependent strings $Y(1), \ldots, Y(m)$, which are defined precisely in §3.1. Again, by the *Sample Path Theorem* we conclude that the shortest feasible path s_m in $\mathcal{T}_m$ is stochastically smaller than the shortest feasible path s_n in the original b-suffix tree, which implies the following

$$\Pr\{s_n < k\} \leq \Pr\{s_m < k\}. \tag{3.12}$$

To estimate the probability $\Pr\{s_m < k\}$ in the modified tree $\mathcal{T}_m$, we need some more notation. Let $p_{\min}(k) = \min_{w_k \in \mathcal{W}_k}\{P(w_k)\}$ and $C_{\mathbf{i}}(w_k)$ be the length of the longest prefix of the word w_k and the $b+1$ suffixes belonging to $\mathbf{i} = (i_1, \ldots, i_{b+1})$. We assume that $\mathbf{i} \in D$, where D is the set of all $(b+1)$-tuples from the set $\{1, \ldots, m\}$. Note now that $\{s_m < k\}$ implies that there must exist a word $w_k \in \mathcal{W}_k$ such that for all $\mathbf{i} \in D$ the self-alignment $C_{\mathbf{i}}$ is smaller than k; that is, $C_{\mathbf{i}} < k$. Using the strong α-mixing condition of (A2) we have

$$\begin{aligned}\Pr\{s_m < k\} &\leq \sum_{\mathcal{W}_k} \Pr\{\bigcap_{\mathbf{i}\in D}[C_{\mathbf{i}}(w_k) < k]\} \leq \sum_{\mathcal{W}_k}(1+\alpha(d_n))^{m^b}(1-P(w_k^b))^{m^b} \\ &\leq (1+\alpha(d_n))^{m^b}\sum_{\mathcal{W}_k}(1-cp_{\min}^b(k))^{m^b} \leq V^k(1+\alpha(d_n))^{m^b}(1-p_{\min}^b(k))^{m^b}.\end{aligned}$$

Now, let $k = \lfloor (1-\epsilon)\frac{1}{h_1}\log n \rfloor$ and $m = \Theta(n/\log n)$ while $d_n = \Theta(\log n)$. Then,

$$\Pr\left\{s_n < (1-\epsilon)\frac{1}{h_1}\log n\right\} \leq (1+\alpha(\log n))^{m^b}\exp(-n^{\epsilon b/2}/\log^b n);$$

and therefore, together with condition (2.9), this leads to the lower bound of the form

$$\Pr\left\{s_n < (1-\epsilon)\frac{1}{h_1}\log n\right\} \leq cn^\beta \exp(-n^{\epsilon b/2}/\log^b n). \tag{3.13}$$

The upper bound (3.11) and the lower bound (3.13) establish the convergence in probability of the shortest feasible path s_n in a b-suffix tree. The almost sure convergence can be derived in an identical manner as for the height since s_n is nondecreasing in n, and for $n = s2^r$ with some fixed s we can apply the Borel–Cantelli lemma (cf. also [38]).

The proof for the shortest depth $\tilde{s}_n$ is simple. Since $s_n \le \tilde{s}_n$, we need only an upper bound. Clearly, a result for $b = 1$ suffices for the proof. Note that either $\tilde{s}_n = s_n$ or, in the same branch (where s_n is located) there are two suffixes, say numbers one and two, with common prefix of length greater than $\tilde{s}_n$. Hence,

$$\Pr\{\tilde{s}_n > k\} \le \Pr\{s_n > k\} + \Pr\{C_{1,2} > k\}.$$

Then, (3.12) and the bound for the self-alignment derived in §3.1 (just above (3.4)), for $k = \lfloor \frac{1}{h_1} \log n \rfloor$ lead to the following estimate

$$\Pr\{\tilde{s}_n > k\} \le \frac{1}{n^{\epsilon}} + \frac{1}{n^{h_2(1+\epsilon)/h_1}},$$

which completes the proof of Theorem 1. □

3.3. The typical depth in b-suffix trees. In this section, we prove Theorem 2(i) and 2(ii). We start with the convergence in probability (pr.) for the depth of insertion L_n. This will also prove the convergence in probability for the typical depth D_n and the depth of a given suffix $L_n(m)$, since all of these quantities are asymptotically equally distributed. The last assertion is easy to prove. Roughly speaking, it must hold in the suffix tree $\mathcal{T}_m$ defined in §3.1, at least when (2.9) takes place. Indeed, consider for example D_n and L_n. In $\mathcal{T}_m$ the next inserted suffix is "almost" independent of the previous suffixes stored already in $\mathcal{T}_m$. Hence, it randomly selects an external node which implies that L_m and D_m are distributed in a similar manner. But, as it is easy to see, the typical depths and depths of insertion in $\mathcal{T}_m$ and $\mathcal{T}_n$ are asymptotically equally distributed. Details are left to the interested reader.

The idea of the proof in this section is quite different from the one discussed before, and it resembles Pittel's proof [30] of the convergence in probability of the depth in an independent trie. It is based on counting, and it is quite typical for the information theory community. For the convenience of the reader, we briefly review the *asymptotic equipartition property* (AEP) [5], [43], which is a direct consequence of the Shannon–McMillan–Breiman theorem [5]: *For a stationary and ergodic sequence* $\{X_k\}_{k=1}^n$, *the state space* Σ^n *can be partitioned into two sets, namely, "good states" set* G_n *and "bad states" set* B_n *such that for* $X_1^n \in G_n$ *and for sufficiently large n we have* $P(X_1^n) \ge 1 - \epsilon$ *for any* $\epsilon > 0$, *and* $P(B_n) \le \epsilon$. Moreover, the nth order probability distribution of $X_1^n \in G_n$ is bounded as $e^{-n(h+\epsilon)} \le P(X_1^n) \le e^{-n(h-\epsilon)}$, where h is the entropy.

We concentrate on L_n. Define and event A_n such that

$$A_n = \{X_1^\infty : |L_n/\log n - 1/h| \ge \epsilon/h\}. \tag{3.14a}$$

For Theorem 2 (i) it suffices to prove that $\Pr\{A_n\} \to 0$ as $n \to \infty$. Also, for some $\epsilon_1 > 0$ and $n_0 \ge n$ we define another event (i.e., set of "good states")

$$G_{n_0} = \{X_1^\infty : |n^{-1} \log P^{-1}(X_1^n) - h| < \epsilon_1 h, \quad n > n_0\}. \tag{3.14b}$$

We partition A_n to obtain

$$P(A_n) \le \Pr\{A_n \textbf{ and } G_{n_0} \textbf{ and } L_n \le \delta \log n\} + \Pr\{L_n \ge \delta \log n\} + P(B_{n_0}), \tag{3.15a}$$

where $\delta > 1/h_2$ and

$$B_{n_0} = \sup_{n \ge n_0} \{X_1^\infty : |n^{-1} \log P^{-1}(X_1^n) - h| \ge \epsilon_1 h, \quad n > n_0\}. \tag{3.15b}$$

By AEP, we have $\lim_{n_0\to\infty} P(B_{n_0}) = 0$. In addition, from the proof of the upper bound for the height H_n we know that $\Pr\{L_n \geq \delta \log n\} \leq c/n^{\delta - 1/h_2}$ for $\delta > 1/h_2$; hence, the second probability in the above also tends to zero.

In view of the above, we can now deal only with the first term in (3.15a), which we denote for simplicity by $P_1(A_nG_n)$. This probability can be estimated as follows

$$P_1(A_nG_n) \leq \sum_{r\in C_n} \Pr\{L_n = r; |\log(P^{-1}(X_1^r))/r - h| < \epsilon_1 h, r \geq n_0\} = \sum_{r\in C_n} P_n^{(r)}, \tag{3.16a}$$

where

$$C_n = \{r : |r/\log n - 1/h| \geq \epsilon/h \quad \textbf{and} \quad r \leq \delta \log n\}. \tag{3.16b}$$

Note that in (3.16) we restrict the summation only to "good states" represented by G_n. Therefore, for a word $w_r \in G_n$ we have with high probability

$$c_1 \exp\{-(1+\epsilon_1)hr\} \leq P(w_r) \leq c_2 \exp\{-(1-\epsilon_1)hr\}. \tag{3.17}$$

The next step is to estimate the probability $\Pr\{L_n = r\}$. But the event $\{L_n = r\}$ takes place if: (i) there exists an $\mathbf{i} = (i_1, \ldots, i_b, n)$ and w_{r-1} such that $C_\mathbf{i} = w_{r-1}$ (call this event F_n^1); and (ii) for all other $\mathbf{j} = (j_1, \ldots, j_b, n) \neq \mathbf{i}$, and all w_r, we have $C_\mathbf{j} \neq w_r$ (call this event F_n^2). Then,

$$\Pr\{L_n = r\} \leq cn^b \sum_{\mathcal{W}_r} P(F_n^1 \cap F_n^2). \tag{3.18}$$

Now, we are in position to prove Theorem 2(i). We first establish the upper bound. Set $r \geq (1+\epsilon)\frac{\log n}{h}$. Hence, by the right-hand side of (3.17) we have $P(w_{r-1}^b) \leq 1/n^{b(1+\epsilon)(1-\epsilon_1)}$. But, using the mixing conditions of (A1) we have $P(F_n^1) \leq cP(w_r)P(w_{r-1}^b)$, and this together with the above leads to (for $\epsilon' \leq \epsilon(1-\epsilon_1) - \epsilon_1$)

$$P_n^{(r)} \leq \frac{c}{n^{\epsilon'}}; \tag{3.19}$$

and therefore, by (3.16) and the fact that the cardinality of C_n is smaller than $\log n$, we have $P(A_n) \leq c \log n / n^{\epsilon'}$ as needed for the upper bound.

Now we consider the lower bound. We apply here the same approach as adopted in the previous lower bounds. So, let $\mathcal{T}_m$ be the suffix tree built from the strings $Y(1), \ldots, Y(m)$ as defined before. In particular, the depth of a given suffix, say the first one, $L_m(1)$ in $\mathcal{T}_m$ is bounded from above by the depth $L_n(1)$ in the original b-suffix tree. Then,

$$\Pr\left\{L_n \leq (1-\epsilon)\frac{\log n}{h}\right\} \leq \Pr\left\{L_m \leq (1-\epsilon)\frac{\log n}{h}\right\}, \tag{3.20}$$

since L_n and $L_n(1)$ have asymptotically the same distribution.

Now, we pick up the derivation at (3.18) in which the first n^b should be replaced by m^b. We estimate the probability $P(F_n^1 \cap F_n^2)$ as follows. Using the strong α-mixing condition of (A2), we have

$$P(F_m^1 \cap F_m^2) \leq cP(w_r)P(w_{r-1}^b)(1+\alpha(d_n))^{m^b}(1 - P(w_r^b))^{m^b}.$$

Let now $r \leq (1-\epsilon)\frac{\log n}{h}$; hence, by the left-hand side of (3.17) and (3.18) and the same argument as in the lower bound for the shortest feasible path, we finally obtain in (3.16a) for $m = O(n/\log n)$

$$P_n^{(r)} \leq c\exp(-n^{b\epsilon/2}/\log^b n) \tag{3.21}$$

provided (2.9) holds.

Putting everything together, we note that the cardinality of the set C_n in (3.16b) is bounded from above by $\delta \log n$; hence, by (3.19) and (3.21), our estimate (3.15) becomes

$$P(A_n) \leq c \log n \left(\exp(-n^{b\epsilon/2}/\log^b n) + n^{-\epsilon'}\right) + P(B_{n_0}), \tag{3.22}$$

which suffices for the proof of Theorem 2(i).

To complete the proof of Theorem 2, we need to establish the almost sure convergence for the depth $L_n(m)$. But this is an immediate consequence of the fact that the depth $L_n(m)$ is a nondecreasing sequence in n. The formal proof is along the same lines as for the height, and is omitted. This completes the proof of Theorem 2 and the entire analysis. □

In passing, we note that a slight extension of the above proof will directly lead to Shields's result concerning the external path length, namely, $E_n^{(b)}/n \log n \rightarrow 1/h$ (a.s.) for Bernoulli and Markovian models.

Acknowledgment. I thank two referees for detailed comments that led to an improvement of the presentation in this paper. I also thank Dr. Pavel Pevzner for pointing out reference [22]. And last but not least, I am grateful to Professor Boris Pittel for numerous discussions concerning this research.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. Apostolico, *The myriad virtues of suffix trees*, Combinatorial Algorithms on Words, Springer-Verlag, ASI F12 (1985), pp. 85–96.

[3] A. Apostolico and F. P. Preparata, *Optimal off-line detection of repetitions in a string*, Theoret. Comput. Sci., 22 (1983), pp. 297–315.

[4] A. Apostolico and W. Szpankowski, *Self-alignments in words and their applications*, J. Algorithms, 13 (1992), pp. 446–467.

[5] P. Billingsley, *Ergodic Theory and Information*, John Wiley & Sons, New York, 1965.

[6] A. Blumer, A. Ehrenfeucht, and D. Haussler, *Average size of suffix trees and DAWGS*, Discrete Appl. Math., 24 (1989), pp. 37–45.

[7] R. Boyer and J. Moore, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762–772.

[8] K. L. Chung and P. Erdös, *On the application of the Borel–Cantelli lemma*, Trans. Amer. Math. Soc., 72 (1952), pp. 179–186.

[9] W. Chang and E. Lawler, *Approximate string matching in sublinear expected time*, Proc. 1990 FOCS, St. Louis, MO, 1990, pp. 116–124.

[10] I. Csiszár and J. Körner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*, Academic Press, New York, 1981.

[11] L. Devroye, W. Szpankowski, and B. Rais, *A note on the height of suffix trees*, SIAM J. Comput., 21 (1992), pp. 48–53.

[12] Z. Galil and K. Park, *An improved algorithm for approximate string matching*, SIAM J. Comput., 19 (1990), pp. 989–999.

[13] P. Gilham and H. L. Weith, personal communications.

[14] G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1991.

[15] P. Grassberger, *Estimating the information content of symbol sequences and efficient codes*, IEEE Trans. Inform. Theory, 35 (1991), pp. 669–675.

[16] L. Guibas and A. Odlyzko, *Periods in strings*, J. Combin. Theory Ser. A, 30 (1981), pp. 19–43.

[17] L. Guibas and A. Odlyzko, *String overlaps, pattern matching, and nontransitive games*, J. Combin. Theory Ser. A, 30 (1981), pp. 183–208.

[18] P. Jacquet and W. Szpankowski, *Analysis of digital tries with Markovian dependency*, IEEE Trans. Inform. Theory, 37 (1991), pp. 1470–1475.

[19] ———, *Autocorrelation on words and its applications: Analysis of suffix trees by string-ruler approach*, J. Combin. Theory, Ser. A, to appear.

[20] P. JACQUET, B. RAIS, AND W. SZPANKOWSKI, *Compact Suffix Trees Resemble PATRICIA Tires: Limiting Distribution of Depth*, CSD-TR-92-048, Purdue University, West Lafayette, IN, 1992.

[21] S. KARLIN AND F. OST, *Counts of long aligned word matches among random letter sequences*, Adv. in Appl. Probab., 19 (1987), pp. 293–351.

[22] R. KARP, *A characterization of the minimum cycle mean in a digraph*, Discrete Mathematics, 23 (1978), pp. 309–311.

[23] D. KNUTH, *The Art of Computer Programming. Sorting and Searching*, Vol. III, Addison-Wesley, Reading, MA, 1973.

[24] D. KNUTH, J. MORRIS, AND V. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.

[25] A. LEMPEL AND J. ZIV, *On the complexity of finite sequences*, IEEE Inform. Theory, 22 (1976), pp. 75–81.

[26] M. LOTHAIRE, *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1982.

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

[28] G. HARDY, J. E. LITTLEWOOD, AND G. PÓLYA, *Inequalities*, Cambridge University Press, MA, 1989.

[29] D. ORNSTEIN AND B. WEISS, *Entropy and Data Compression Schemes*, IEEE Trans. Inform. Theory, 39 (1993), pp. 78–83.

[30] B. PITTEL, *Asymptotic growth of a class of random trees*, Ann. Probab., 18 (1985), pp. 414–427.

[31] ———, *Paths in a random digital tree: Limiting distributions*, Adv. in Appl. Probab., 18 (1986), pp. 139–155.

[32] B. RAIS, P. JACQUET, AND W. SZPANKOWSKI, *Limiting distribution for the depth of PATRICIA tries*, SIAM J. Discrete Math., 6 (1993), pp. 197–213.

[33] L. V. ROMANOVSKI, *Optimization of stationary control of a discrete deterministic process*, Cybernetics, 3 (1967), pp. 52–62.

[34] P. SHIELDS, *Entropy and prefixes*, Ann. Probab., 20 (1992), pp. 403–409.

[35] D. STOYANA, *Comparison Methods for Queues and Other Stochastic Models*, John Wiley & Sons, Chichester, 1983.

[36] W. SZPANKOWSKI, *On the height of digital trees and related problems*, Algorithmica, 6 (1991), pp. 256–277.

[37] ———, *Patricia tries again revisited*, J. Assoc. Comput. Mach., 37 (1991), pp. 691–711.

[38] ———, *Asymptotic properties of data compression and suffix trees*, IEEE Trans. Inform. Theory, (1993), to appear.

[39] ———, *(Un)Expected behavior of typical suffix trees*, Third Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 422–431.

[40] M. WATERMAN, *Mathematical Methods for DNA Sequences*, CRC Press, Inc., Boca Raton, FL, 1991.

[41] P. WEINER, *Linear Pattern Matching Algorithms*, Proc. 14th Annual Symposium on Switching and Automata Theory, 1973, pp. 1–11.

[42] U. VISHKIN, *Deterministic sampling—A new technique for fast pattern matching*, SIAM J. Comput., 20 (1991), pp. 22–40.

[43] A. WYNER AND J. ZIV, *Some asymptotic properties of the entropy of a stationary ergodic data source with applications to data compression*, IEEE Trans. Inform. Theory, 35 (1989), pp. 1250–1258.

[44] J. ZIV AND A. LEMPEL, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23, 3 (1977), pp. 337-343.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1199–1217, December 1993

FINDING THE HIDDEN PATH: TIME BOUNDS FOR ALL-PAIRS SHORTEST PATHS*

DAVID R. KARGER†, DAPHNE KOLLER†, AND STEVEN J. PHILLIPS†

Abstract. The all-pairs shortest-paths problem in weighted graphs is investigated. An algorithm—the Hidden-Paths Algorithm—that finds these paths in time $O(m^*n + n^2 \log n)$, where m^* is the number of edges participating in shortest paths, is presented. The algorithm is a practical substitute for Dijkstra's algorithm. It is argued that m^* is likely to be small in practice since $m^* = O(n \log n)$ with high probability for many probability distributions on edge weights. An $\Omega(mn)$ lower bound on the running time of any path-comparison-based algorithm for the all-pairs shortest-paths problem is also proved. Path-comparison-based algorithms form a natural class containing the Hidden-Paths Algorithm, as well as the algorithms of E. W. Dijkstra [*Numer. Math.*, 1 (1959), pp. 269–271] and R. W. Floyd [*Comm. ACM*, 5 (1962), p. 345]. Lastly, generalized forms of the shortest-paths problem are considered, and it is shown that many of the standard shortest-paths algorithms are effective in this more general setting.

Key words. all-pairs shortest paths, weighted graphs, lower bound, path comparison, algorithm, random graphs

AMS subject classifications. 68R10, 05C85

1. Introduction.

Oh what a peaceful life is theirs
Who worldly strife and noise have flown
And follow the hidden path from cares
Which none but the wise of the world have known.

Fray Luis de Leon, c. 1527–1591

Let G be a weighted directed graph with n vertices and m edges. The *all-pairs shortest-paths problem* is to find a shortest path between each pair of vertices in G.

Our contributions to this problem lie in three areas. In §2 we present a new algorithm for all-pairs shortest paths, called the *Hidden-Paths Algorithm*, which works on graphs with nonnegative edge weights. Let an edge be called *optimal* if it is a shortest path, and let m^* be the number of optimal edges in the graph. The Hidden-Paths Algorithm runs in time $O(m^*n + n^2 \log n)$ if we use a Fibonacci heap [12] to implement a priority queue; the running time increases to $O(m^*n \log n)$ if a standard heap is used instead. The algorithm operates by running Dijkstra's single-source shortest-paths algorithm [7] in parallel from all nodes in the graph, where information gained at one node is used to reduce the work done at other nodes. Our algorithm is practical and simple to implement. Is is also likely to be fast in practice because it is known [14], [18], [26] that $m^* = O(n \log n)$ with high probability when the input graph is the complete graph with edge weights chosen independently from any of a large class of probability distributions, including the uniform distribution on the real interval [0, 1] or the uniform distribution on the range $\{1, \dots, n^2\}$. The Hidden-Paths Algorithm is also useful when the graph has (possibly negative) integer edge weights. Such a graph can be reweighted by using an $o(mn)$ run-time scaling algorithm for single-source shortest paths (see for example, Gabow and Tarjan [15], Goldberg [16]) before applying the Hidden-Paths Algorithm. If the edge weights are polynomial-sized integers, the combined running time of Goldberg's scaling algorithm and the Hidden-Paths Algorithm is $O(m^*n + n^2 \log n + \sqrt{n}m \log n)$.

Our second contribution is a new lower bound, given in §3. Many algorithms for the all-pairs shortest-paths problem use the edge-weight function only in comparing the weights

*Received by the editors September 3, 1991; accepted for publication (in revised form) July 22, 1992. This research was supported by a National Science Foundation Graduate Fellowship and by National Science Foundation grants CCR-8858030-03 and CCR-9010517. Parts of it were performed while D. Koller was at IBM Almaden Research Center, San Jose, California. A preliminary version of this paper appeared in the Proceedings of the IEEE 32nd Annual Symposium on Foundations of Computer Science, 1991, pp. 560–568.

†Department of Computer Science, Stanford University, Stanford, California 94305.

of paths in the graph. We call such algorithms *path-comparison based*, and we prove that on directed graphs any path-comparison-based algorithm requires $\Omega(mn)$ time in the worst case (in the worst case $m^* = m$). The idea of the lower bound is to construct a weighted directed graph with $\Theta(mn)$ directed paths. If an algorithm $\mathcal{A}$ fails to examine any path, we show how to modify the weight function so that the unexamined path is optimal, without $\mathcal{A}$ detecting the change. The directed graph we use is acyclic, so that the lower bound holds even for this restricted class of directed graphs.

In proving the lower bound we show that any path-comparison-based algorithm requires $\Omega(mn)$ time even to verify the output of an all-pairs shortest-paths algorithm. Thus in the path-comparison model verification is as hard as finding the paths. We also show a lower bound of $\Omega(n^3)$ for verifying that the edge weights of a graph satisfy the triangle inequality. These lower bounds also hold for randomized path-comparison-based algorithms.

Finally, we investigate generalized shortest-paths problems in which the weight of a path is not necessarily the sum of the weights of its edges. For example, one might define the weight of a path to be the maximum of the weights of its edges. In §4 we show that many all-pairs shortest-paths algorithms work, even for generalized path-weight functions, in their normal time bounds. We also extend our lower-bound proof to show an $\Omega(mn)$ lower bound on the running time of any algorithm that solves the all-pairs shortest-paths problem for certain generalized path-weight functions.

Previous work. The most widely known algorithms for the all-pairs shortest-paths problem are those of Floyd [9] and of Dijkstra [7]. Floyd's algorithm works by dynamic programming and runs in time $\Theta(n^3)$. On graphs with nonnegative edge weights Dijkstra's algorithm for the single-source shortest-path problem can be run from each vertex (as noted by Johnson [21]), resulting in a running time of $\Theta(mn + n^2 \log n)$ if Fibonacci heaps [12] are used to implement priority queues. A variant of Dijkstra's algorithm developed by Spira [30] has an expected running time of $O(n^2 \log^2 n)$ if the edge weights are independently and identically distributed random variables. Bloniarz [3] provided an algorithm with an expected running time of $O(n^2 \log n \log^* n)$. Another algorithm, developed by Frieze and Grimmet [14], achieves an expected running time of $O(n^2 \log n)$, but it is suitable only for random graphs. All of these algorithms are path-comparison based, so that by the lower bound of §3 they have a worst-case running time of $\Omega(mn)$. Fast algorithms exist for special cases of the all-pairs shortest-paths problem, e.g., for cases in which the graph is unweighted [8] or planar [10].

Other researchers have looked at shortest paths from the perspective of matrix multiplication. Fredman [11] showed that $O(n^{5/2})$ comparisons between sums of edge weights suffice to solve the all-pairs shortest-paths problem. He used this fact to do preprocessing and produced an algorithm that runs in time $O(n^3(\log\log n \,/\, \log n)^{1/3})$. Fredman's algorithm was simplified and the running time was decreased slightly by Takaoka [32]. For the important special cases in which the graph is unweighted or the edge weights are bounded integers, the algorithms of Alon, Galil, and Margalit [1] and of Seidel [29] use fast matrix multiplication to find all-pairs shortest distances very quickly. For example, Seidel's algorithm finds all-pairs shortest distances in an unweighted undirected graph in time $n^\omega \log n$, where ω is the exponent of matrix multiplication (the current bound is $\omega < 2.376$, due to Coppersmith and Winograd [5]). Alon, Galil, Margalit, and Naor [2] showed how to extend these algorithms to find the paths, rather than just the distances, with a polylogarithmic slowdown. These results for matrix-based algorithms should be contrasted with the lower bound in §3; such algorithms are not path-comparison based, since they perform comparisons involving sums of weights of edges that do not form a path. The use of such comparisons distinguishes the algebraic decision tree model from the path-comparison-based model. It is surprising that one must allow such comparisons in order to improve on the $\Omega(n^3)$ bound.

An algorithm similar to the Hidden-Paths Algorithm, with the same time bound, has been developed independently by McGeoch [27]. A variant of our algorithm has been developed independently by Jakobsson [20] as a transitive closure algorithm. Both of these algorithms require more complex data structures than those used by the Hidden-Paths Algorithm.

Lower bounds on the computational complexity of the all-pairs shortest-paths problem have been proved in some other models. Kerr [22] has shown that if the permissible operations are addition and minimum in a straight-line computation, any algorithm requires $\Omega(n^3)$ running time. Regarding algebraic decision tree complexity, Spira and Pan [31] showed that $\Omega(n^2)$ comparisons between sums of edge weights are necessary for solving the single-source shortest-paths problem.

Generalized weight functions have been studied extensively in various contexts, and standard algorithms have been extended to work in generalized settings (see, for example, Frieze [13]). In particular, the all-pairs shortest-path problem has been studied in arbitrary semirings, generalizing the semiring of reals with minimum and addition (see Zimmerman [34] for a survey and further references). This generalization provides a common framework for such problems as the construction of regular expressions for the languages accepted by finite automata [19]. Lengauer and Theune [25] extended standard algorithms to a yet more general framework. Knuth [23] generalized the notion of paths to allow for compound edges, extending Dijkstra's algorithm to apply to derivations in context-free grammars.

2. The Hidden-Paths Algorithm. In this section we describe the Hidden-Paths Algorithm, which solves the all-pairs shortest-paths problem in a directed graph $G = (V, E)$ with nonnegative edge weights. In order to present our algorithm we need to make the following definitions.

2.1. Preliminary definitions. We use $(u_1, u_2, \dots, u_k)$ to denote a path from u_1 to u_k that goes through the vertices $u_2, \dots, u_{k-1}$. The symbol $(u \rightsquigarrow v)$ denotes some path from u to v (which may be the edge (u, v) if it exists). The symbol $(u \rightsquigarrow v \rightsquigarrow w)$ denotes the concatenation of the paths represented by $(u \rightsquigarrow v)$ and $(v \rightsquigarrow w)$, and $(u, v \rightsquigarrow w)$ denotes the concatenation of the edge (u, v) to the path $(v \rightsquigarrow w)$. The *length* of a path $(u_1, \dots, u_k)$ is

$$|(u_1, \dots, u_k)| = k - 1.$$

Let $\|(u, v)\|$ denote the *weight* of the edge (u, v). We extend the weight function by setting $\|(u, v)\| = \infty$ for any $(u, v) \notin E$ and by setting $\|(u, u)\| = 0$, so that $\|(u, v)\|$ is defined for all pairs u, v. The weight of a path $(u_1, \dots, u_k)$ is

$$\|(u_1, \dots, u_k)\| = \sum_{i=1}^{k-1} \|(u_i, u_{i+1})\|.$$

DEFINITION 1. A path $(u \rightsquigarrow v)$ is *optimal* if for any other path $(u \rightsquigarrow' v)$, $\|(u \rightsquigarrow v)\| \leq \|(u \rightsquigarrow' v)\|$.

DEFINITION 2. An edge is *optimal* if it is an optimal path between its endpoints.

DEFINITION 3. The number of optimal edges in G is denoted by $m^*(G)$.

FACT 2.1. *An edge is optimal if and only if it participates in some shortest path.*

Note that in an unweighted graph each edge is optimal, and so $m^* = m$ and our algorithm provides no improvement.

DEFINITION 4. The *distance* $d(u, v)$ between two vertices u and v is the weight of an optimal path between them.

2.2. Description of the algorithm. The Hidden-Paths Algorithm presented in this section finds a shortest path between every pair of vertices in a directed graph. Essentially, it runs Dijkstra's single-source shortest-paths algorithm in parallel for all points in the graph. The different single-source threads are integrated in a way that permits the use of intermediate results from one thread to reduce the work done by another. There is a similarity here to the all-pairs min-cut algorithm of Gomory and Hu [17], which uses the information gained during one min-cut computation to speed up the other computations. In a sense, the Hidden-Paths Algorithm discovers the hidden "shortest-path structure" of the graph by pruning away the unnecessary edges. We note that the algorithm actually constructs each path in reverse order, by adding edges to the tail of the path. This facilitates forward traversal on the constructed paths. It is simple to modify the algorithm to construct the paths in the form typically used in Dijkstra's algorithm. Here we give an intuitive description of the Hidden-Paths Algorithm; a precise presentation can be found in Figs. 1 and 2.

- For each vertex v a list E_v^* of the optimal edges directed into v that have been found so far.
- A path array P, which for every pair u, v describes the current best path from u to v. Each entry $P[u, v]$ consists of
  first The first vertex on this path (other than u itself).
  weight The weight of the path.
  heaptr A pointer to the heap entry for this path, if one exists.
- A heap H of pointers to candidate paths that appear in the path array, ordered by weight.

FIG. 1. *Data structures maintained by the Hidden-Paths Algorithm.*

```
Initialize
    Initialize the heap of pairs $u \neq v$ ordered by the weight $\|(u, v)\|$.
    For all $u, v$
        Set $P[u, v].weight := \|(u, v)\|$.
        Set $P[u, v].heaptr$ to point to appropriate heap entry.
    For all $v$ set $E_v^*$ to be empty.

While (heap not empty and weight of top item $\neq \infty$) do
    Step 1  Remove the top element $(u \rightsquigarrow v)$ from the heap.
    Step 2  If $(u \rightsquigarrow v)$ is an edge, then
                Add $(u, v)$ to $E_v^*$.
                For each $w$, Update$(u, v, w)$.
    Step 3  For all edges $(t, u) \in E_u^*$
                Update$(t, u, v)$.

Procedure Update$(x, y, z)$:
    If $P[x, y].weight + P[y, z].weight < P[x, z].weight$, then
        Set $P[x, z].weight := P[x, y].weight + P[y, z].weight$.
        Set $P[x, z].first := y$.
        Change the priority of $x, z$ in the heap to $P[x, z].weight$.
        Modify $P[x, z].heaptr$ accordingly.
```

FIG. 2. *The Hidden-Paths Algorithm.*

The Hidden-Paths Algorithm maintains a heap containing, for every two vertices $u \neq v$, the best path from u to v found so far. The heap is ordered according to path weight. It is initialized to contain for each pair $u \neq v$ a path of weight $\|(u, v)\|$ (recall that if $(u, v) \notin E$, then $\|(u, v)\| = \infty$). The path at the top of the heap is always an optimal path.

At each iteration the algorithm removes a path, say, $(u \leadsto v)$, from the top of the heap (a delete-min operation). This is an optimal path from u to v. This path is now used to construct a set of new *candidate paths*. If a new candidate path $(x \leadsto z)$ is shorter, it replaces the current best path from x to z in the heap. This maintains the optimality of the path at the top of the heap.

The complexity of the algorithm is a function of the number of candidate paths created, and so we do not want to create too many. If $(u \leadsto v)$ is an edge, then the candidate paths are all those paths of the form $(u, v \leadsto w)$.[1] If $(u \leadsto v)$ is a path that is not an edge, then the candidate paths are all those of the form $(t, u \leadsto v)$, where (t, u) is an optimal edge that has already been found. Note that in the construction of candidate paths the edge is always concatenated to the tail.

The following theorem shows that constructing this limited set of candidate paths suffices for finding a shortest path between each pair of vertices in the graph. The analysis in §2.3 shows that the resulting complexity is $O(m^*n + n^2 \log n)$.

Before we state the theorem it is convenient to make one more definition.

DEFINITION 5. For any two paths p and q, $p \prec q$ if $(\|p\|, |p|) < (\|q\|, |q|)$ according to lexicographic order.

THEOREM 2.2. *The Hidden-Paths Algorithm finds an optimal path between every connected pair of vertices in the graph. Furthermore, it discovers them in order of increasing weight.*

Proof. Let *Opt* be the set of optimal paths found so far. We prove the theorem by induction on the size of *Opt*. The inductive hypothesis is that at the beginning of each iteration, when p is the item at the top of the heap, the following hold:

1. p is an optimal path.
2. *Opt* contains an optimal path between each pair of vertices of distance less than $\|p\|$.
3. For each pair of vertices u and v for which an optimal path has not yet been found the heap contains a path of minimal weight among those of the form (i) the edge (u, v) and (ii) paths having the form $(u, w \leadsto v)$ for (u, w) and $(w \leadsto v)$ in *Opt*.

The inductive hypothesis holds trivially at the beginning of the first iteration, since at that time p is the shortest edge in the graph, *Opt* is empty, and the heap contains all the edges. The construction of candidate paths in Steps 2 and 3 of the algorithm (see Fig. 2) ensures that condition 3 always holds at the beginning of an iteration.

It remains to prove that when $p = (u \leadsto v)$ is the item on top of the heap, conditions 1 and 2 hold for p. This can be false only if there exists an optimal path $r = (w \leadsto y)$, such that $\|r\| < \|p\|$, and no optimal path from w to y has been placed in the heap. Condition 1 is violated if $w = u$ and $y = v$; condition 2 is violated otherwise. Let r be a smallest such path according to $\prec$. Since all edges were placed in the heap during the initialization step, r must be a path of length at least two. Assume $r = (w, x \leadsto y)$. It is clear that $(w, x) \prec r$ and $(x \leadsto y) \prec r$ and that both are necessarily optimal. By our choice of r some optimal path $(x \leadsto' y)$ must have been placed in the heap. The edge (w, x) was placed in the heap during initialization and was never replaced by a shorter path. Since $\|(x \leadsto' y)\| = \|(x \leadsto y)\| \leq \|r\| < \|p\|$, the path $(x \leadsto' y)$ must have already been deleted from the heap and placed in *Opt*. Similarly, the edge (w, x) must also be in *Opt*. Therefore, by condition 3 of the inductive hypothesis the

[1]A minor optimization is to consider only vertices w such that $(v \leadsto w)$ has already been discovered to be optimal. The algorithm remains correct, and fewer arithmetic operations are performed. An extra field *opt* in the path array P can be used to mark whether or not a path has already been found to be optimal.

TABLE 1
Running time of the Hidden-Paths Algorithm.

Operation	No. of ops	Cost for std. heap	Cost for Fib. heap
Create	$O(n^2)$	$O(n^2)$	$O(n^2)$
Priority change	$O(m^*n)$	$O(m^*n \log n)$	$O(m^*n)$
Delete-min	$O(n^2)$	$O(n^2 \log n)$	$O(n^2 \log n)$
Total	—	$O(m^*n \log n)$	$O(m^*n + n^2 \log n)$

path $(w, x \rightsquigarrow' y)$, whose weight is equal to $\|r\|$, must have been constructed as a candidate path, and thus a path from w to y of no greater weight must be in the heap. This path is an optimal path from w to y, contradicting the assumption that no such path was placed in the heap. □

2.3. Analysis of running time. Let us count the number of operations used by the algorithm. In the initialization step a heap of size $O(n^2)$ is created; it is clear that the cost of the other operations in this step is subsumed by the cost of the heap operations. The While loop is iterated at most $n(n-1)$ times since in each iteration an optimal path is found. Therefore, the algorithm executes at most $n(n-1)$ delete-min operations in Step 1.

It remains to count only the time taken by Steps 2 and 3. Note that in each of these steps the calls to Procedure Update subsume all other operations. It therefore suffices to count calls to Procedure Update. We amortize the calls over the edges of the graph. In Step 2 charge the $O(n)$ calls to the edge (u, v). In Step 3 charge a call to Update(t, u, v) to the edge (t, u). Observe that in either case it is an optimal edge that is being charged. We claim that, in fact, only $O(n)$ updates are charged to any edge. This is clear for Step 2 since no edge is removed from the heap more than once. For Step 3 note that at most $n - 1$ optimal paths $(u \rightsquigarrow v)$ are found leaving any particular vertex u and that Procedure Update is called only when $(u \rightsquigarrow v)$ is such an optimal path. Since only optimal edges are charged and each is charged $O(n)$ updates, the total number of updates charged is $O(m^*n)$. Procedure Update requires a constant number of primitive operations plus at most one priority change operation.

The complexity of the algorithm depends on the implementation of the heap (see Table 1). If a standard heap implementation is used, the time for a priority change operation is $O(\log n)$ and we get a total complexity of $\Theta(m^*n \log n)$. If Fibonacci heaps (described in [12]) are used, priority change operations take constant amortized time, and we therefore get a complexity of $\Theta(n^2 \log n + m^*n)$.

The Hidden-Paths Algorithm is also very simple and easy to implement, and it thus provides a practical substitute for Dijkstra's algorithm.

2.4. Extensions and refinements. The Hidden-Paths Algorithm can easily be transformed to one that finds the k nearest pairs of vertices in the graph. The revised algorithm will initialize the heap to contain only the actual edges in the graph. Then, when candidate paths are compared to existing paths (in Procedure Update), the algorithm will sometimes have to do an insert operation rather than a priority change operation. The algorithm terminates when $|Opt|$ reaches k. Because at most k^2 candidate paths are created, the running time of this algorithm (when Fibonacci heaps are used) is $O(m + k \log n + k^2)$.

The Hidden-Paths Algorithm requires that the graph have nonnegative edge weights. However, in the case of an integer edge-weight function several scaling algorithms (such as Gabow and Tarjan [15] and Goldberg [16]) transform the weight function into a nonnegative weight function, which induces the same shortest-paths structure on the graph. We can solve the shortest-paths problem on such graphs by first using one of these algorithms to make the edge weights positive and then applying the Hidden-Paths Algorithm. Let $-N$, $N > 0$, be the weight of the smallest (most negative) edge in the graph. The combined running time of Goldberg's scaling algorithm and the Hidden-Paths Algorithm is $O(m^*n + n^2 \log n + \sqrt{n}m \log N)$. Thus the Hidden-Paths Algorithm may also improve on the $O(mn)$ upper bound in graphs with negative edge weights.

Although Fibonacci heaps have a better asymptotic running time than do standard heaps, they may be undesirable in applications because of their greater complexity. It is therefore useful to point out cases in which the Hidden-Paths Algorithm can be used with regular heaps without increasing the asymptotic complexity. This can happen if not every candidate path created necessitates a priority change operation. We show below that this situation occurs when the edge weights in the graph are small integers.[2] Such graphs are an interesting special case that can occur frequently in practice.

LEMMA 2.3. *Let $(u \rightsquigarrow v)$ be a nonedge candidate path in the heap. Let a be the weight of the longest optimal edge in the graph. Then*

$$\|(u \rightsquigarrow v)\| \leq d(u, v) + a.$$

Proof. This is clear if $(u \rightsquigarrow v)$ is optimal. On the other hand, a nonoptimal path $(u \rightsquigarrow v) = (u, w \rightsquigarrow v)$ can become a path in the heap only if at some stage of the algorithm (u, w) and $(w \rightsquigarrow v)$ are in *Opt* but no optimal u, v path is in *Opt*. The edge (u, w) is optimal, and therefore $\|(u, w)\| \leq a$. Since $(w \rightsquigarrow v)$ is optimal and in *Opt* and since optimal paths are discovered in increasing order, we conclude that

$$\|(w \rightsquigarrow v)\| \leq d(u, v).$$

Therefore,

$$\|(u \rightsquigarrow v)\| = \|(u, w)\| + \|(w \rightsquigarrow v)\| \leq a + d(u, v). \qquad \square$$

THEOREM 2.4. *If the edge weights are integers and a is the largest weight of an optimal edge, then there can be at most $(a + 1)n^2$ priority changes and the Hidden-Paths Algorithm has a running time of $O(an^2 \log n + m^*n)$ when an ordinary heap is used.*

Proof. We will prove that for any pair of vertices u and v the entry for u, v in the heap can be modified at most $a + 1$ times. The original entry for u, v in the heap is the edge, (u, v). This entry can be replaced only by a nonedge path $(u \rightsquigarrow v)$. By Lemma 2.3, $\|(u \rightsquigarrow v)\|$ is at most $d(u, v) + a$. By integrality the priority of (u, v) can then be decreased at most a times. Overall, there can be at most $a + 1$ priority change operations for the pair u, v. $\square$

In fact, an even better bound can be achieved if we modify the algorithm slightly. The modified algorithm is similar to Dial's implementation [6] of Dijkstra's algorithm. Note that if the maximum edge weight is a, the weight of any optimal path is at most $a(n - 1)$. Rather than using a heap to store candidate paths, we can use an array of size $a(n - 1) + 1$. Paths of weight w will be kept in a bucket at index w of the array. Initially, only edges are inserted. Later, each time a candidate path $(u \rightsquigarrow v)$ improves the path from u to v we remove the old path from the array and insert the new path at index $\|(u \rightsquigarrow v)\|$. Rather than iteratively deleting entries from the heap, we traverse the array from index 0 to index $a(n-1)$. As before, each path we encounter as we traverse the array will be an optimal path. Also as before, we construct $O(m^*n)$ candidate paths. Thus the total running time of this modified algorithm is $O(an + m^*n)$.

If space is a concern, it can be conserved in the following fashion. It is easy to prove, by using the techniques of Lemma 2.3, that if p is the path at the top of the heap, then there is no

[2]It actually suffices that the weights of the optimal edges be small integers.

nonedge path of weight exceeding $\|p\| + a$ in the heap. It follows that at any given time we need use only $a + 1$ buckets of our array to store candidate paths. Therefore, our array can be mapped onto a circular array of size $a + 1$ such that no collision of buckets ever takes place. It follows that only $a + 1$ space is needed to store the buckets. If $a = O(n^2)$, it follows that the space used is no more than that needed in the heap implementation; if, in addition, the diameter by weight of the graph is $O(m^*n)$ (and certainly if $a = O(m^*)$), then the running time of the algorithm remains $O(m^*n)$.

Another easy consequence is the following:

LEMMA 2.5. *If* $(u \rightsquigarrow v)$ *is a nonedge nonoptimal candidate path in the heap, then*

$$\|(u \rightsquigarrow v)\| \leq 2\, d(u, v).$$

Proof. As in Lemma 2.3, the path $(u \rightsquigarrow v) = (u, w \rightsquigarrow v)$ can be placed in the heap only if (u, w) and $(w \rightsquigarrow v)$ are both found before any optimal u, v path. This implies that

$$\begin{aligned} \|(u, w)\| &\leq d(u, v), \\ \|(w \rightsquigarrow v)\| &\leq d(u, v) \end{aligned}$$

and therefore that

$$\|(u \rightsquigarrow v)\| = \|(u, w)\| + \|(w \rightsquigarrow v)\| \leq 2\, d(u, v). \qquad \square$$

This shows that at any stage in the algorithm any nonedge path in the heap has at most twice the optimal weight. This property is useful for "anytime" applications, which might require the algorithm to be stopped in the middle of execution with good intermediate results.

Finally, we note that the Hidden-Paths Algorithm considers some unnecessary candidate paths. One possible improvement, which was also developed independently by Jakobsson [20], creates only candidate paths of which every subpath is optimal. More specifically, a path $p = (u, v \rightsquigarrow w, t)$ is made a candidate path if and only if $(u, v \rightsquigarrow w)$ and $(v \rightsquigarrow w, t)$ are already known to be optimal. This will clearly reduce the number of candidate paths formed, but there does not seem to be a simple expression for the reduced running time achieved by this algorithm. More complex data structures are required, but it is still possible to achieve a running time of $\Theta(c + n^2 \log n)$, where c is the number of candidate paths formed.

2.5. m^* **in a random graph.** To predict the behavior of the Hidden-Paths Algorithm in practice, we need to study the quantity m^* for typical graphs. It is easy to construct graphs for which $m^* = O(n)$, while m is $\Theta(n^2)$. It is also easy to construct graphs for which $m^* = m$. In this section we note that for a large class of probability distributions on random graphs, $m^* = O(n \log n)$ with high probability.

Consider a distribution F on nonnegative edge weights, which does not depend on n, such that $F(0) = 0$ and $F'(0)$ exists and is positive. In particular, the uniform distribution on [0, 1] and the exponential distribution with mean λ both satisfy these conditions. The work of Frieze and Grimmet [14] implies that $m^*(G) = O(n \log n)$ with high probability. In particular, they proved the following result:

THEOREM 2.6 (Frieze and Grimmet). *Let G be a complete directed graph whose edge weights are chosen independently according to F. Consider the set S of edges defined by placing (v, w) in S if and only if it is one of the p shortest edges originating at v, where* $p = \min\{n - 1, 20 \log_2 n\}$. *Then with probability* $1 - O(n^{-1})$, *S contains every optimal edge in G.*

Hence, under the conditions of the theorem, $m^*(G) = O(n \log n)$ with probability $1 - O(n^{-1})$ (and therefore $E[m^*(G)] = O(n \log n)$). Similar results were derived in a different context by Luby and Ragde [26]. Hassin and Zemel [18] proved a similar theorem (with a different constant) for both directed and undirected graphs, when the edge weights are uniformly distributed. The constant factors given by these analyses are small. In fact, empirical studies by McGeoch [27] indicate that when the edge weights are uniformly distributed $m^*(G)$ grows approximately as $0.5n \ln n + 0.3n$. Furthermore, Theorem 2.6 holds for some discrete edge-weight distributions, for instance, if the edge weights are integers chosen uniformly and independently from the range $1, \ldots, n^2$.

COROLLARY 2.7. *If the edge weights of G are chosen independently according to F, then with high probability the running time of the Hidden-Paths Algorithm is* $O(n^2 \log n)$.

This time bound is an improvement over earlier algorithms by Spira [30] and by Bloniarz [3], and it matches the performance of the algorithm of Frieze and Grimmet. However, the Hidden-Paths Algorithm can be effectively used in any situation in which m^* is significantly less than m, whereas the algorithm of Frieze and Grimmet is designed specifically for random graphs.

3. A lower bound.

3. A lower bound. Many algorithms for the shortest-paths problem use edge weights only to compute and compare the weights of paths. We therefore define a version of the decision tree model that captures this behavior.

DEFINITION 6. A *path-comparison-based* all-pairs shortest-paths algorithm $\mathcal{A}$ accepts as input a graph G and a weight function. The algorithm $\mathcal{A}$ can perform all standard operations. However, the only way it can access the edge weights is to compare the weights of two different paths.

We can think of a path-comparison-based algorithm as being given only the graph and a black-box path-weight comparator. The path weights can be accessed only through the black box. The algorithm must output a reasonable encoding of the shortest paths in G.[3]

It should be noted that the algorithms of Floyd, Dijkstra, Spira, Bloniarz, and Frieze and Grimmet, as well as the Hidden-Paths Algorithm, all fit into this path-comparison-based model. On the other hand, Fredman's $o(n^3)$ algorithm [11] is not path-comparison-based because it adds weights of edges that do not form a single path. This algorithm conforms to the more general algebraic decision tree model.

We show a lower bound of $\Omega(mn)$ on the running time of any path-comparison-based algorithm running on a graph with n vertices and m edges. For simplicity we first show a lower bound of $\Omega(n^3)$ on the running time of any path-comparison-based shortest-paths algorithm running on a certain graph of $\Theta(n^2)$ edges. We then show how the construction can be modified to yield a lower bound of $\Omega(mn)$ for a graph with m edges, where on these graphs $m = m^*$. An obvious modification allows us to construct graphs with arbitrary values of m^* and m ($m^* \leq m$), for which $\Omega(m^*n)$ time would be required.

To show the $\Omega(n^3)$ lower bound we construct a directed graph of $3n$ vertices on which any path-comparison-based shortest-paths algorithm must perform $\Omega(n^3)$ comparisons. The directed graph G has $\Omega(n^3)$ paths. We show that if $\mathcal{A}$ fails to examine one of these paths, then we can modify the weight function to make that path optimal without $\mathcal{A}$ being able to detect the change.

The graph G is a directed tripartite graph on vertices u_i, v_j, and w_k, where i, j, and k range from 0 to $n - 1$. The edge set for G is $\{(u_i, v_j)\} \cup \{(v_j, w_k)\}$ (see Fig. 3). Therefore, the only paths are individual edges and paths (u_i, v_j, w_k) of length two.

[3]We require that the output have no information about path weights. For example, the weighted graph itself is not a reasonable encoding of the solution.

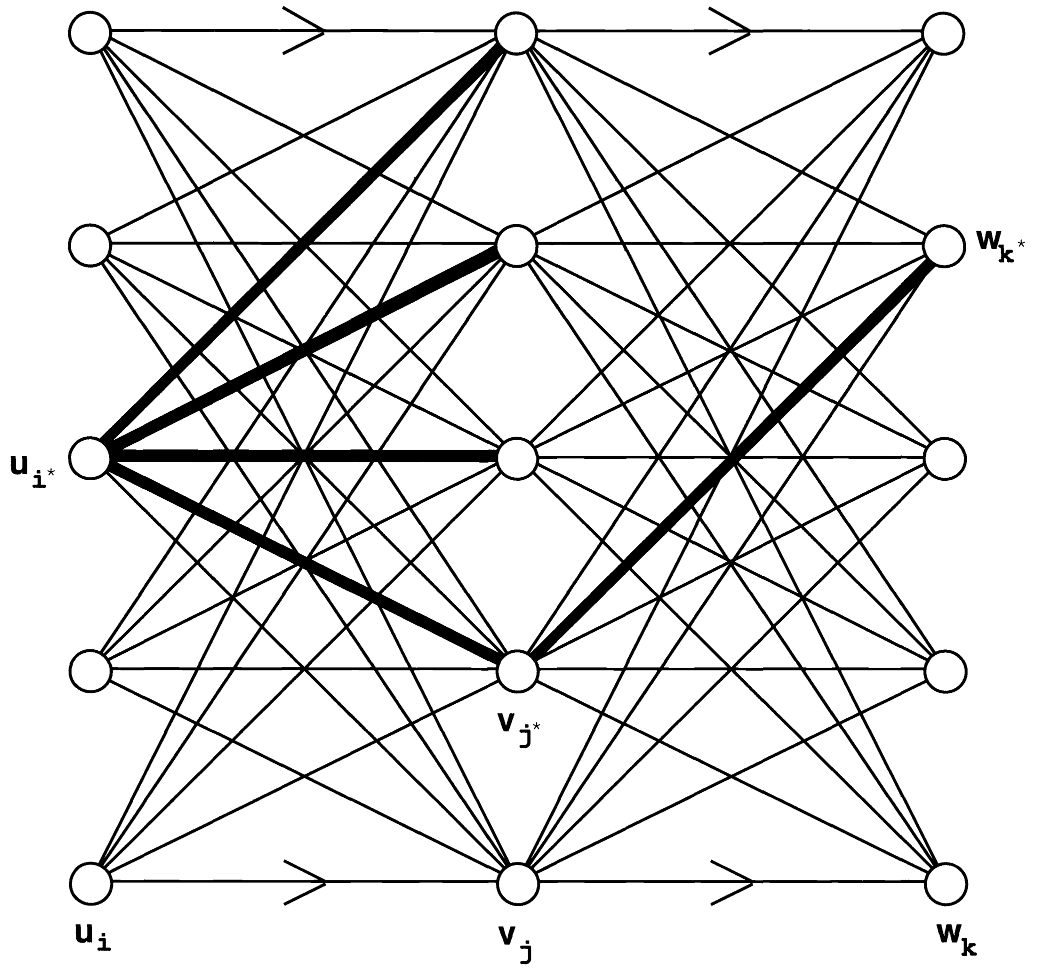


FIG. 3. *Graph used in the lower bound.*

To define the weight function we work in base $n+1$ notation generalized to allow negative digits. Define

$$[a_r, \ldots, a_0]_b = \sum_{i=0}^{r} a_i b^i.$$

The edge weights are

$$\begin{aligned} \|(u_i, v_j)\| &= [1,\ 0,\ i,\ 0,\ j,\ 0,\ 0]_{n+1}, \\ \|(v_j, w_k)\| &= [0,\ 1,\ 0,\ k,\ 0,\ -j,\ 0]_{n+1}, \end{aligned}$$

and thus

$$\|(u_i, v_j, w_k)\| = [1,\ 1,\ i,\ k,\ j,\ -j,\ 0]_{n+1}.$$

Note that we allow negative digits to appear in the numbers. The standard positive-digit representation of these numbers would require that a carry be taken from the next number to the left. This does not affect the correctness of the upcoming proofs. The following lemma is an immediate consequence of the definitions:

LEMMA 3.1. *Let $<$ denote the lexicographic ordering on tuples of integers, with the leftmost integer being the most significant. For all i, j, j', k, k' the following hold*:

1. $\|(u_i, v_j)\| < \|(u_{i'}, v_{j'})\|$ *if and only if* $(i, j) < (i', j')$;
2. $\|(v_j, w_k)\| < \|(v_{j'}, w_{k'})\|$ *if and only if* $(k, -j) < (k', -j')$;
3. $\|(v_j, w_k)\| < \|(u_i, v_{j'})\|$;
4. $\|(u_{i'}, v_{j'})\| < \|(u_i, v_j, w_k)\|$;
5. $\|(u_i, v_j, w_k)\| < \|(u_{i'}, v_{j'}, w_{k'})\|$ *if and only if* $(i, k, j) < (i', k', j')$.

Proof. The proof is immediate from the base $n + 1$ notation. For example, item 3 follows from the fact that $\|(v_j, w_k)\| < [1, 0, 0, 0, 0, 0, 0]_{n+1} \leq \|(u_i, v_j)\|$. □

It follows that the unique optimal path from u_i to w_k goes through v_0 and has weight $[1, 1, i, k, 0, 0, 0]_{n+1}$. Define L to be the set of optimal paths.

Consider providing $(G, \|\cdot\|)$ as input to $\mathcal{A}$, and suppose that $\mathcal{A}$ runs correctly. It must therefore output the set of optimal paths L. Suppose further that a nonoptimal path $p^* = (u_{i^*}, v_{j^*}, w_{k^*})$ with $j^* > 0$ was never one of the operands in any comparison operation performed by $\mathcal{A}$. We define a weight function $\|\cdot\|'$ in which p^* is the unique shortest path from u_{i^*} to w_{k^*}, but the ordering by weight of all the other paths remains the same. If we run $\mathcal{A}$ on $(G, \|\cdot\|')$, all path comparisons not involving p^* give the same result as they did when $\|\cdot\|$ was used. Therefore, since $\mathcal{A}$ never performed a comparison involving p^* while running on $\|\cdot\|$, we deduce that $\mathcal{A}$ still outputs L, which is now incorrect. If $\mathcal{A}$ never examined an optimal path (u_{i^*}, v_0, w_{k^*}), we can apply the above construction with $j^* = 1$. The algorithm will then fail because the only comparison that has a different result is between (u_{i^*}, v_0, w_{k^*}) and (u_{i^*}, v_1, w_{k^*}), which by hypothesis was not performed.

The weight function $\|\cdot\|'$ is $\|\cdot\|$ with the following modifications (in Fig. 3 the edges with modified weights are marked by thicker lines). For all $j \leq j^*$ we decrease the weight of the edge (u_{i^*}, v_j):

$$\|(u_{i^*}, v_j)\|' = [1, 0, i^*, 0, 0, j, j]_{n+1}.$$

We also decrease the weight of the edge (v_{j^*}, w_{k^*}):

$$\|(v_{j^*}, w_{k^*})\|' = [0, 1, 0, k^*, 0, -j^*, -n]_{n+1}.$$

Thus

$$\|(u_{i^*}, v_{j^*}, w_{k^*})\|' = [1, 1, i^*, k^*, 0, 0, j^* - n]_{n+1} < \|(u_{i^*}, v_0, w_{k^*})\|'.$$

LEMMA 3.2. *In G the conditions of Lemma* 3.1 *continue to hold for $\|\cdot\|'$, except that the single path $p^* = (u_{i^*}, v_{j^*}, w_{k^*})$ directly precedes (u_{i^*}, v_0, w_{k^*}) in the ordering. Thus under $\|\cdot\|'$ the path p^* is optimal.*

Proof. We show the conditions of Lemma 3.1 one at a time. Clearly, we need only consider comparisons for which one or both operands have changed, i.e., only comparisons between operands involving i^*, j^*, or k^*.

1. The four most significant digits of the base $n + 1$ representation of the weights remain unchanged, and the three least significant digits still increase with j.
2. Only $\|(v_{j^*}, w_{k^*})\|$ has changed, and only by n, but the edge whose weight was closest differed by $n + 1$.
3. The two most significant digits are unchanged.
4. This also is enforced by the two most significant digits.

5. If $i \neq i'$ or $k \neq k'$, the inequality is enforced by the four most significant digits. It is also simple to verify that for each i and k, $\|(u_i, v_j, w_k)\|$ increases with j (with the exception of $(u_{i^*}, v_{j^*}, w_{k^*})$). □

We have therefore proved the following:

THEOREM 3.3. *There exists a directed graph of* $3n$ *vertices on which any path-comparison-based shortest-paths algorithm must perform at least* $n^3/2$ *path-weight comparisons.*

Note that since all edge weights are polynomial in n, the input graph G is not hard merely because unusually large edge weights increase the input size. Because the graph constructed is a directed acyclic graph, the lower bound holds even for this restricted class of graphs.

We now adapt the above proof to show an $\Omega(mn)$ lower bound for graphs of m edges. Let $m \geq 4n$, and assume without loss of generality that $2n$ divides m. We perform the same construction as before, but of the middle vertices we use only $v_0, \ldots, v_{m/2n-1}$, connecting each of them to all the vertices u_i and w_k. This requires m edges and creates $mn/2$ two-edge paths. We also use the same weight function as before, restricted to the edges we include.

THEOREM 3.4. *There exists a directed graph with* $2n + m/2n$ *vertices and* m *edges, on which any path-comparison-based shortest-paths algorithm must perform at least* $mn/4$ *path-weight comparisons.*

COROLLARY 3.5. *If* $m = \Omega(n)$, *then there exists a directed graph* G *with* n *vertices and* m *edges on which any path-comparison-based shortest-paths algorithm must perform* $\Omega(mn)$ *path-weight comparisons.*

For $m = \Omega(n \log n)$ this lower bound is tight since it matches the upper bound achieved by Dijkstra's algorithm.

We can in fact show that even the shortest-paths *verification* problem requires $\Omega(mn)$ time for path-comparison-based algorithms. A verification algorithm $\mathcal{A}$ accepts as input a graph, a weight function (which we again think of as a black-box comparator), and an encoding L that describes, for each pair of vertices, a path between them. Note that the standard description of shortest paths can be encoded in $O(n^2)$ space, so that the input size imposes no nontrivial lower bound. The algorithm $\mathcal{A}$ accepts its input if and only if each path in L is a shortest path.

To show the lower bound, we use the same construction as before. We let L be the shortest paths under $\|\cdot\|$, i.e., $L = \{(u_i, v_0, w_k) \mid i, k = 0, \ldots, n-1\}$, and provide $(G, \|\cdot\|, L)$ as input to $\mathcal{A}$. If $\mathcal{A}$ accepts using fewer than mn comparisons, we use the same modification as before and pass $(G, \|\cdot\|', L)$ to $\mathcal{A}$, which will incorrectly accept this modified input.

COROLLARY 3.6. *Any path-comparison-based algorithm for verification of shortest paths requires time* $\Omega(mn)$ *on* G.

If we add edges from each u_i to each w_k (thus producing $\Omega(n^2)$ edges) and set $\|(u_i, w_k)\| = \|(u_i, v_0, w_k)\|$, we can similarly deduce the following:

COROLLARY 3.7. *Any path-comparison-based algorithm for verifying that the edge weights satisfy the triangle inequality requires time* $\Omega(n^3)$ *on graphs of* n *vertices.*

The construction of Theorem 3.4 can be applied to randomized algorithms for the shortest-paths problem. For example, suppose that the expected number of comparisons performed by such an algorithm is $o(mn)$. Then the probability that a randomly selected path is checked by the algorithm approaches 0 as n goes to ∞. Thus if we take the graph G and select a single path $(u_{i^*}, v_{j^*}, w_{k^*})$ uniformly at random and apply the $\|\cdot\|'$ construction, the algorithm detects our modification with probability approaching 0. We thus have the following theorem.

THEOREM 3.8. *If a randomized path-comparison-based shortest-paths algorithm performs* $o(mn)$ *expected comparisons on graphs with* m *edges and* n *vertices, then there is a weighted graph on which the algorithm will almost surely fail to be correct.*

The following corollaries are randomized counterparts of Corollaries 3.6 and 3.7:

COROLLARY 3.9. *Any randomized path-comparison-based shortest-paths verification algorithm must perform $\Omega(mn)$ expected comparisons.*

COROLLARY 3.10. *Any randomized path-comparison-based algorithm for verifying that all edge weights satisfy the triangle inequality must perform $\Omega(n^3)$ expected comparisons.*

We conjecture that the lower bounds in this section also hold for path-comparison-based algorithms running on undirected graphs, but this remains to be proved.

4. Generalized weight functions.

4.1. Definitions and basic properties.

4.1. Definitions and basic properties. Many shortest-paths algorithms, in fact, solve a much more general problem. In particular, we consider the following generalized shortest-paths problem: Given a graph G and a generalized weight function $\|\cdot\|$ that maps every path p to a weight $\|p\|$ in some totally ordered set (with the ordering denoted by $\leq$), find for each pair of vertices a path between them of minimum weight. To make this problem tractable we impose restrictions on the weight function.

DEFINITION 7. Consider a weight function $\|\cdot\|$:

- It is *monotonic* if for all u, v, w

$$\|(v \rightsquigarrow w)\| \leq \|(v \rightsquigarrow' w)\| \implies \|(u \rightsquigarrow v \rightsquigarrow w)\| \leq \|(u \rightsquigarrow v \rightsquigarrow' w)\|,$$

  and similarly for $\|(u \rightsquigarrow v)\| \leq \|(u \rightsquigarrow' v)\|$.
- It is *nonnegative* if for all u, v, w

$$\|(u \rightsquigarrow v \rightsquigarrow w)\| \geq \max(\|(u \rightsquigarrow v)\|, \|(v \rightsquigarrow w)\|).$$

- It is *acyclic* if for all v the empty path from v to v, denoted $v \overset{\emptyset}{\rightsquigarrow} v$, is optimal.
- It is *inductive* if there exists a *concatenation function* f such that for all u, v, w

$$\|(u \rightsquigarrow v \rightsquigarrow w)\| = f(\|u \rightsquigarrow v\|, \|v \rightsquigarrow w\|).$$

The standard path-weight function is monotonic and inductive. It is acyclic if there are no negative weight cycles. It is nonnegative if all path weights are nonnegative. An example of a monotonic, nonnegative, inductive, nonstandard weight function is one that assigns to every path a weight equal to the weight of the maximal edge on the path. Solving single-source shortest paths under this weight function is referred to as the *bottleneck path problem* in [33].

In the literature on generalizing shortest paths to semirings (see [34]) the semiring axioms imply both inductiveness and monotonicity of the weight function. Frieze [13] considered a narrower class, which essentially consists of monotonic, acyclic, inductive weight functions over the reals. Lengauer and Theune [25] studied extensions of the shortest-paths problem to situations in which the path weights are only partially ordered; this allowed them to deal with certain nonmonotonic weight functions.

FACT 4.1. *Any nonnegative weight function is also acyclic.*

Proof. Let v be any vertex, and let $(v \rightsquigarrow v)$ be any cycle. Then

$$\begin{aligned} \|(v \rightsquigarrow v)\| &= \|(v \rightsquigarrow v \overset{\emptyset}{\rightsquigarrow} v)\| \\ &\geq \max(\|v \rightsquigarrow v\|, \|v \overset{\emptyset}{\rightsquigarrow} v\|) \qquad \text{(by nonnegativity)} \\ &\geq \|(v \overset{\emptyset}{\rightsquigarrow} v)\| . \end{aligned}$$

Therefore, $(v \overset{\emptyset}{\rightsquigarrow} v)$ is optimal. □

FACT 4.2. *If $\|\cdot\|$ is monotonic and acyclic, then any path can be trimmed, by removing cycles, to obtain a simple path of smaller or equal weight.*

Proof. If a path p contains a cycle, replace the cycle with the appropriate empty path, which by acyclicity has no greater weight than the cycle; by monotonicity this replacement does not increase the weight of the path. Continue this procedure until no cycles remain. □

FACT 4.3. *Under a monotonic and acyclic weight function, any connected pair of vertices is connected by a simple optimal path.*

Proof. If two vertices are connected, the set of simple paths between them is nonempty and finite and thus must contain some path of minimal weight. By Fact 4.2 this path is also shorter than all nonsimple paths between these vertices, and this path is thus optimal. □

FACT 4.4. *The shortest paths under any monotonic acyclic weight function can be encoded in the standard manner in $O(n^2)$ space.*

Proof. In the (u, v) entry store the first edge on any minimal-length shortest path from u to v. □

We argue that monotonicity is the major defining characteristic of the shortest-paths problem because it ensures that the shortest path between two vertices can be constructed from other shortest paths. The property of acyclicity is also important since it ensures that a simple optimal path exists between every pair of connected vertices. We shall therefore restrict our attention to monotonic acyclic weight functions.

The following definition will be used in our extension of the Hidden-Paths Algorithm to generalized weight functions.

DEFINITION 8. A path is *taut* if every edge in it is optimal. A weight function is taut if every connected pair of vertices is connected by a taut optimal path.

LEMMA 4.5. *Any monotonic and acyclic weight function is taut.*

Proof. Define a nonoptimal edge (u, v) to be *strongly nonoptimal* if there is no taut optimal path from u to v, i.e., if every optimal path from u to v contains a nonoptimal edge. For the lemma to be false there must be some pair of vertices all of whose optimal paths contain a strongly nonoptimal edge. Otherwise, using the monotonicity condition, we could take a path containing no strongly nonoptimal edge and replace each nonoptimal edge with a taut optimal subpath, thus producing a taut optimal path. It therefore suffices to prove that no strongly nonoptimal edges exist.

Assume that there exists some strongly nonoptimal edge (u, v). By the same argument as above, an edge is strongly nonoptimal if and only if every optimal path between its endpoints contains a strongly nonoptimal edge. Since by Fact 4.3 there exists an optimal path between every pair of connected vertices, we can construct an infinite sequence of edges $\{(u, v) = (u_0, v_0), (u_1, v_1), (u_2, v_2), \ldots\}$ such that (u_{i+1}, v_{i+1}) is a strongly nonoptimal edge contained in some optimal path $(u_i \overset{i}{\rightsquigarrow} u_{i+1}, v_{i+1} \overset{i}{\rightsquigarrow} v_i)$ from u_i to v_i. Since the set of edges is finite, we have that $(u_i, v_i) = (u_{i+k}, v_{i+k})$ for some i, k. Now observe that by induction on j and by using the monotonicity condition, $(u_i \overset{i}{\rightsquigarrow} u_{i+1} \overset{i+1}{\rightsquigarrow} \cdots \overset{i+j-1}{\rightsquigarrow} u_{i+j}, v_{i+j} \overset{i+j-1}{\rightsquigarrow} \cdots \overset{i+1}{\rightsquigarrow} v_{i+1} \overset{i}{\rightsquigarrow} v_i)$ is an optimal path. In particular, this is true for $j = k$. Since $u_{i+k} = u_i$, this means we have an optimal path of the form $(u_i \rightsquigarrow u_i, v_i \rightsquigarrow v_i)$. By the conditions of monotonicity and acyclicity this implies that $(u_i \overset{\emptyset}{\rightsquigarrow} u_i, v_i \overset{\emptyset}{\rightsquigarrow} v_i) = (u_i, v_i)$ must be optimal, contradicting the fact that (u_i, v_i) is strongly nonoptimal. □

4.2. Algorithms.

An algorithm for a generalized shortest-paths problem receives as input the graph and a black box for the weight function. We assume that the black box takes constant time to compute the weight of any path. Many path-comparison-based shortest-paths algorithms also work for generalized weight functions. We consider here Floyd's algorithm, Dijkstra's algorithm, and the Hidden-Paths Algorithm.

THEOREM 4.6. *Floyd's algorithm works on any monotonic and acyclic weight function.*

Proof. Recall that Floyd's algorithm iteratively finds for each i the best path between every pair of vertices u, v that uses (except for the endpoints) only the first i vertices in the graph. It does this by comparing, at stage i, the best path that uses only the first $i - 1$ vertices with all paths of the form $(u \rightsquigarrow w \rightsquigarrow v)$, where w is the ith vertex and both $(u \rightsquigarrow w)$ and $(w \rightsquigarrow v)$ use only the first $i - 1$ vertices.

The proof is by induction on i. The inductive hypothesis is that after stage i the algorithm has found for every pair of vertices u, v a best path among those using only the first i vertices. Note that once an optimal path is found, it is never replaced.

The base case, $i = 0$, is obvious. Assume that the inductive hypothesis holds for stage $i - 1$, and let w be the ith vertex. Let u, v be a pair of vertices for which any best path that uses only the first i vertices uses the vertex w. Let this path be $(u \rightsquigarrow v) = (u \rightsquigarrow w \rightsquigarrow v)$. By Fact 4.2 any cycle in a path can be eliminated without increasing the weight of the path, so that we may assume that neither $(u \rightsquigarrow w)$ nor $(w \rightsquigarrow v)$ contains w as an interior point. In other words, the paths $(u \rightsquigarrow w)$ and $(w \rightsquigarrow v)$ use only the first $i - 1$ vertices. By the inductive hypothesis, by the end of stage $i - 1$ Floyd's algorithm found a best path $(u \rightsquigarrow' w)$ among all the paths from u to w using only the first $i - 1$ vertices. Therefore,

$$\|(u \rightsquigarrow' w)\| \leq \|(u \rightsquigarrow w)\|.$$

Similarly, the algorithm found a path $(w \rightsquigarrow' v)$ such that

$$\|(w \rightsquigarrow' v)\| \leq \|(w \rightsquigarrow v)\|.$$

Using monotonicity, we deduce that

$$\|(u \rightsquigarrow' w \rightsquigarrow' v)\| \leq \|(u \rightsquigarrow w \rightsquigarrow' v)\| \leq \|(u \rightsquigarrow w \rightsquigarrow v)\|.$$

Therefore, in phase i Floyd's algorithm finds a path from u to v that is optimal among paths using only the first i vertices.

At the conclusion of stage n the algorithm has found for each pair u, v a path that is best among those using all vertices, i.e., an optimal path. □

Note as well that Floyd's algorithm can be used to verify the acyclicity of a path-weight function since it will find a cycle that is better than an empty path if such a cycle exists.

THEOREM 4.7. *Dijkstra's algorithm works on any monotonic nonnegative weight function.*

Proof. Dijkstra's algorithm is run from a single *source* vertex s and can be thought of as maintaining a set Opt of optimal paths $(s \rightsquigarrow v)$ from s to some of the other vertices in the graph. Let $V(Opt)$ denote the set of endpoints of paths in Opt. At each iteration the algorithm adds to Opt a path $(s \rightsquigarrow u, v)$ that minimizes $\{\|(s \rightsquigarrow' u, v)\| \mid (s \rightsquigarrow' u) \in Opt, (u, v) \in E, v \notin V(Opt)\}$. It suffices to show by induction that the path $(s \rightsquigarrow u, v)$ is in fact an optimal path. To show this, consider some other path from s to v, $(s \rightsquigarrow x, y \rightsquigarrow v)$, where (x, y) is the first edge on the path such that $x \in V(Opt)$ and $y \notin V(Opt)$. Such an edge must exist since $s \in V(Opt)$ and $v \notin V(Opt)$. Since $x \in V(Opt)$, there exists an optimal path $(s \rightsquigarrow' x) \in Opt$. Then

$$\begin{aligned} \|(s \rightsquigarrow x, y \rightsquigarrow v)\| &\geq \|(s \rightsquigarrow x, y)\| \quad \text{(by nonnegativity)} \\ &\geq \|(s \rightsquigarrow' x, y)\| \quad \text{(by monotonicity)} \\ &\geq \|(s \rightsquigarrow' u, v)\| \quad \text{(by choice of } u, v\text{)}. \end{aligned}$$

Thus $\|(s \leadsto' u, v)\|$ is in fact optimal, as desired. □

THEOREM 4.8. *The Hidden-Paths Algorithm works on any monotonic nonnegative weight function.*

Proof. We modify the inductive hypothesis in the proof of Theorem 2.2 as follows:

1. If p is the item at the top of the heap, then p is an optimal path.
2. *Opt* contains a taut optimal path between each pair of vertices of distance less than $\|p\|$.
3. For each pair of vertices u and v for which an optimal path has not yet been found, the heap contains a path of minimal weight among those of the form (i) the edge (u, v) and (ii) taut paths having the form $(u, w \leadsto v)$ for (u, w) and $(w \leadsto v)$ in *Opt*.

An examination of the proof shows that the nature of the weight function is used only in the proof of conditions 1 and 2. Assume one of them to be false, and let $r = (x \leadsto w)$ be a minimal (under $\prec$) taut path such that no optimal path from x to w has yet been placed in the heap. As before, r cannot consist of a single edge, and so assume that $r = (x, y \leadsto w)$. Because of nonnegativity $(x, y) \prec r$ and $(y \leadsto w) \prec r$. Therefore, $d(y, w) \leq \|r\|$. If $d(y, w) < \|r\|$, then by the minimality of r and Lemma 4.5 there exists an optimal path $(y \leadsto' w)$ in *Opt*. If $d(y, w) = \|r\|$, then $(y \leadsto w)$ is taut and optimal and we can again deduce by the choice of r and the definition of $\prec$ that an optimal path $(y \leadsto' w)$ must already be in *Opt*. The edge $(x, y) \prec r$ and is optimal by the tautness of r, and it must therefore also be in *Opt*. But then $(x, y \leadsto' w)$, which because of monotonicity is also an optimal path from x to w, must have been placed in the heap. This contradicts the assumption that no optimal path from x to w has been found. □

4.3. Lower bounds. The lower bound in §3 can be adapted to the situation of generalized weight functions. We show a lower bound for the class of monotonic nonnegative weight functions, even on undirected graphs.

THEOREM 4.9. *Any algorithm for solving the generalized shortest-paths problem for arbitrary monotonic nonnegative weight functions requires $\Omega(mn)$ path-weight queries.*

Proof. We consider a modified version of the construction from §3. Use the same graph G with middle vertices $v_0, \ldots, v_{m/2n-1}$ but with undirected edges, and let $\|\cdot\|$ be defined as follows:

$$\begin{aligned} \|(v, v)\| &= 0 \text{ for all vertices } v, \\ \|(u_i, v_j)\| &= 2, \\ \|(v_j, w_k)\| &= 2, \\ \|(u_i, v_j, w_k)\| &= 4. \end{aligned}$$

All other paths have length 5. Suppose as before that some path $(u_{i^*}, v_{j^*}, w_{k^*})$ does not have its weight queried. Change the weight of this path to be 3. It is simple to verify that the modified weight function remains monotonic and nonnegative. □

This lower bound can also be extended to the case of inductive weight functions studied in [13]. To do this, assign to each edge e in the graph a unique weight $\|e\|$. We are then free to assign arbitrary weights to paths of length 2 because each such path contains a different pair of subpath weights. Our concatenation function is defined by the weights we want to assign to these length-2 paths. To assign weight ω to the path $(u \leadsto v \leadsto w)$, set $f(\|u \leadsto v\|, \|v \leadsto w\|) = \omega$. We can now proceed almost exactly as in the previous case. Assign weights $1, \ldots, n^2$ to the individual edges. Assign weight $4n^2$ to all paths of length 2, and weight $5n^2$ to all other paths (so that $f(4n^2, y) = f(x, 4n^2) = 5n^2$). If any path of length 2 is not examined, change the weight of this path to $3n^2$. It is trivial to verify that these weight functions are inductive (as well as nonnegative and monotonic). We have proved the following theorem:

THEOREM 4.10. *Any algorithm that solves the generalized shortest-paths problem for arbitrary monotonic nonnegative inductive weight functions requires $\Omega(mn)$ path-weight queries.*

Note that Theorem 4.9 holds true even when the weight function is restricted to take integer values in a bounded range. It is easy to see that Theorem 4.10 requires unbounded edge weights.

Corollaries parallel to Corollaries 3.6 and 3.7 also hold. Thus any subcubic solution to the standard shortest-paths problem must take advantage of more specific properties of path weights than monotonicity, acyclicity, nonnegativity, and induction.

5. Conclusion.

We have produced a new algorithm, the Hidden-Paths Algorithm, and we have identified a new measure m^*—the number of edges that participate in shortest paths. The Hidden-Paths Algorithm runs in time $O(m^*n + n^2 \log n)$. The following question arises: Are there finer measures of the shortest-paths difficulty of a graph? In particular, the Hidden-Paths Algorithm essentially runs in time proportional to the number of candidate paths formed. The improved algorithm mentioned in §2.4 forms fewer such paths than does the Hidden-Paths Algorithm. Is there a simple measure for this quantity? We conjecture that no path-comparison-based algorithm for all-pairs shortest paths can perform fewer path-weight comparisons than does the improved algorithm.

The expected value of m^* has been shown to be significantly less than m in the case of independent, uniformly distributed edge weights. This suggests that there are many situations in which the Hidden-Paths Algorithm will be significantly faster than Dijkstra's algorithm. One can think of the optimal edges as forming a *certificate* of the shortest-path structure of the graph, which must be revealed. The philosophy of the Hidden-Paths Algorithm is thus similar to that of recent algorithms for connectivity [4], [28] that work by first finding a sparse subgraph (or certificate) with the same connectivity.

We have shown a lower bound of $\Omega(mn)$ on the running time of path-comparison-based algorithms for all-pairs shortest paths. It is of particular interest that the construction and verification algorithms have the same worst-case complexity. Compare this to the situation for the minimum spanning tree problem, where there is a linear-time algorithm to verify a minimum spanning tree [24], although no algorithm is known that finds one in linear time. The comparison-based lower bound shows that any improvement in the worst-case complexity of shortest-paths algorithms, such as Fredman's $o(n^3)$ algorithm, must take advantage of the numerical aspects of the problem in addition to the ordering of path weights.

The obvious open problem arising from the lower bound is to extend the construction to the case of undirected graphs. Another goal would be to decrease the gap in the algebraic decision tree complexity, between Spira and Pan's $\Omega(n^2)$ lower bound and Fredman's $O(n^{5/2})$ upper bound. Also, our lower bound would be strengthened if we could show that it held for all graphs of a certain structure and varying weights, rather than for a single graph.

Finally, in §4 we introduced the notion of a generalized weight function and defined some natural properties of such functions. We showed that the $\Omega(mn)$ lower bound also holds for a certain class of generalized weight functions, even for undirected graphs. It would be interesting to find tighter classes of weight functions for which the lower bound still holds. We have shown that many existing all-pairs shortest-paths algorithms make little use of numerical properties of path weights and hence work even for generalized weight functions. These algorithms are all path-comparison based. We feel that there is a strong connection between the algorithmic property of being path-comparison based and the ability of an algorithm to work on generalized weight functions. Further work on this topic could lead to a better understanding of how properties of a path-weight function affect the complexity of graph algorithms and what properties of the standard weight function allow the path-comparison based lower bound to be circumvented.

Acknowledgments. We thank M. Luby and C. McGeoch for pointing out references to work on expected shortest-path lengths in graphs with random edge weights.

REFERENCES

[1] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1991, pp. 569–575.

[2] N. Alon, Z. Galil, O. Margalit, and M. Naor, *Witnesses for Boolean matrix multiplication and for shortest paths*, Tech. Report RJ 8744, IBM, White Plains, NY, 1992.

[3] P. A. Bloniarz, *A Shortest-Path Algorithm with Expected Time $O(n^2 \log n \log^* n)$*, Tech. Report 80-3, Department of Computer Science, State University of New York, Albany, NY, 1980.

[4] J. Cheriyan and R. Thurimella, *Algorithms for parallel k-vertex connectivity and sparse certificates*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1991, pp. 391–401.

[5] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[6] R. B. Dial, *Algorithm 360: Shortest path forest with topological ordering*, Comm. ACM, 12 (1969), pp. 632–633.

[7] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[8] T. Feder and R. Motwani, *Clique partitions, graph compression and speeding-up algorithms*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1991, pp. 123–133.

[9] R. W. Floyd, *Algorithm 97: Shortest path*, Comm. ACM, 5 (1962), p. 345.

[10] G. N. Frederickson, *Planar graph decomposition and all pairs shortest paths*, J. Comput. Mach., 38 (1991), pp. 162–204.

[11] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM J. Comput., 5 (1976), pp. 83–89.

[12] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 36 (1986), pp. 596–615.

[13] A. Frieze, *Minimum paths in directed graphs*, Oper. Res. Quart., 28 (1977), pp. 339–346.

[14] A. M. Frieze and G. R. Grimmet, *The shortest-path problem for graphs with random arc-lengths* , Discrete Appl. Math., 10 (1985), pp. 57–77.

[15] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput., 19 (1989), pp. 1013–1036.

[16] A. V. Goldberg, *Scaling Algorithms for the Shortest Paths Problem*, Tech. Report STAN-CS-92-1429, Stanford University, Stanford, CA, 1992.

[17] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, SIAM J. Appl. Math., 9 (1961), pp. 551–570.

[18] R. Hassin and E. Zemel, *On shortest paths in graphs with random weights*, Math. Oper. Res., 10 (1985), pp. 557–564.

[19] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Series in Computer Science, Addison-Wesley, Reading, MA, 1979.

[20] H. Jakobsson, *Mixed-approach algorithms for transitive closure*, in Proceedings of the 10th ACM Symposium on Principles of Database Systems, Association for Computing Machinery, New York, 1991, pp. 199–205.

[21] D. B. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1–13.

[22] L. R. Kerr, *The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplications*, Ph.D. thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1970.

[23] D. E. Knuth, *A generalization of Dijkstra's algorithm*, Inform. Process. Lett., 6 (1977), pp. 1–5.

[24] J. Komlos, *Linear verification for spanning trees*, Combinatorica, 5 (1985), pp. 57–65.

[25] T. Lengauer and D. Theune, *Efficient algorithms for path problems with general cost criteria*, in Proceedings of the 18th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol 510, Springer-Verlag, Berlin, New York, 1991, pp. 314–326.

[26] M. LUBY AND P. RAGDE, *A bidirectional shortest-path algorithm with good average case behavior*, Algorithmica, 4 (1989), pp. 551–567.

[27] C. C. MCGEOCH, *All-pairs shortest paths, and the essential subgraph*, Algorithmica, to appear.

[28] H. NAGAMOCHI AND T. IBARAKI, *A linear time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[29] R. SEIDEL, *On the all-pairs-shortest-path problem*, in Proceedings of the 24th ACM Symposium on Theory of Computing, Association for Computing Machinery, New York 1992, pp. 745–749.

[30] P. M. SPIRA, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time* $O(n^2 \log^2 n)$, SIAM J. Comput., 2 (1973), pp. 28–32.

[31] P. M. SPIRA AND A. PAN, *On finding and updating shortest paths and spanning trees*, in Proceedings of the 14th Annual Symposium on Switching and Automata Theory, IEEE Computer Society, Washington, DC, 1973, pp. 82–84.

[32] T. TAKAOKA, *A new upper bound on the complexity of the all pairs shortest path problem*, in Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Vol 570, Springer-Verlag, Berlin, New York, 1991, pp. 209–213.

[33] R. E. TARJAN, *Data Structures and Network Algorithms*, CBMS–NSF Regional Conference Series in Applied Mathematics, Vol. 44, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[34] U. ZIMMERMAN, *Linear and Combinatorial Optimization in Ordered Algebraic Structures*, Annals of Discrete Mathematics, Vol. 10, North-Holland, Amsterdam, 1981.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1218–1226, December 1993

ON FINDING THE RECTANGULAR DUALS OF PLANAR TRIANGULAR GRAPHS*

XIN HE†

Abstract. This paper presents a new linear-time algorithm for finding rectangular duals of planar triangular graphs. The algorithm is conceptually simpler than the previously known algorithm. The coordinates of the rectangular dual constructed by the new algorithm are integers and carry clear combinatorial meaning.

Key words. algorithm, planar graph, rectangular dual

AMS subject classifications. 05C05, 05C38, 68Q25, 68R10

1. Introduction. Let R be a rectangle. A *rectangular subdivision system* of R is a partition of R into a set $\Phi = \{R_1, R_2, \ldots, R_n\}$ of nonintersecting smaller rectangles such that no four rectangles in Φ meet at the same point. A *rectangular dual* of a graph $G = (V, E)$ is a rectangular subdivision system Φ and a one-to-one correspondence $f : V \rightarrow \Phi$ such that two vertices u and v are adjacent in G if and only if their corresponding rectangles $f(u)$ and $f(v)$ share a common boundary. Parts (1) and (2) of Fig. 1 show a graph G and its rectangular dual. If G has a rectangular dual, clearly G must be a planar graph.

The rectangular dual of a graph G finds applications in the floor planning of electronic chips and in architectural design [5], [9]. Each vertex of G represents a circuit module, and the edges represent module adjacencies. A rectangular dual provides a placement of the circuit modules that preserves the required adjacencies.

The problem of finding rectangular duals has been studied in [2], [3], [6], [8], [12]. A linear-time algorithm for this problem was given in [3]. This algorithm is rather complicated, and its correctness proof is incomplete. The algorithm requires real arithmetic for the coordinates of the rectangular dual. We present a new linear-time algorithm for solving this problem. The coordinates of the rectangular dual R constructed by our algorithm are integers and carry clear combinatorial meaning.

The rectangular dual is related to the *tessellation representation* of planar graphs discussed in [14], [15]. The tessellation representation of a plane graph G is a mapping that maps the vertices, edges, and faces of G to the rectangles of the plane such that the incidence relations of G correspond to the geometric adjacencies between the rectangles. A nice linear-time algorithm for constructing the tessellation representation was developed in [15]. Our rectangular-dual algorithm and the algorithm in [15] share certain similarities: both algorithms use two acyclic directed graphs derived from the input graph G. However, the rectangular dual differs from the tessellation representation in at least two aspects. Firstly, in the tessellation representation, the elements (vertices, edges, and faces) of distinct graph-theoretical properties are represented by the same geometric objects (rectangles). The rectangular dual is a more natural representation of planar graphs. Secondly, the rectangular dual is more general than the tessellation representation. Consider a plane graph G. Let G_1 be the plane graph obtained from G as follows: For each face f of G place a new vertex v_f in f and connect v_f to the vertices on the boundary of f. Then the rectangular dual of G_1 is a representation similar to the tessellation representation of G, i.e., the vertices and the faces of G are represented by rectangles and the edges of G are represented by degenerate rectangles (i.e., line segments). On the other hand, it does not seem possible to modify the tessellation representation algorithm so that it constructs the rectangular dual for planar graphs.

*Received by the editors October 2, 1989; accepted for publication (in revised form) May 19, 1992.

†Department of Computer Science, State University of New York, Buffalo, New York 14260. This research was supported in part by National Science Foundation grant CCR-9011214.

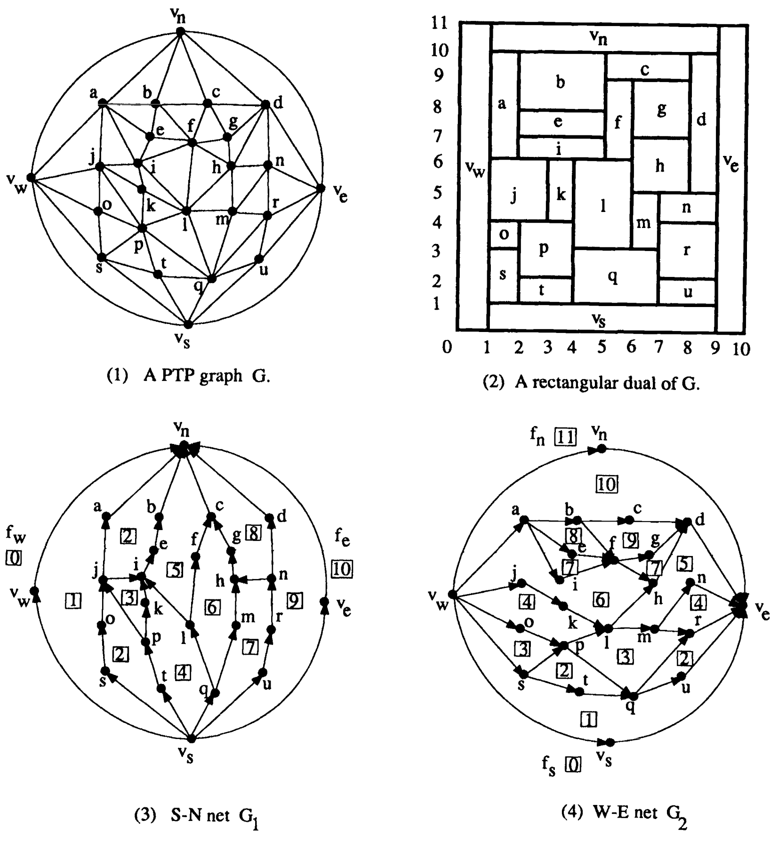


FIG. 1. *A* PTP *graph G and a rectangular dual of G.*

The present paper is organized as follows. Section 2 introduces some definitions and lemmas needed by our algorithm. Section 3 presents the algorithm. Section 4 proves its correctness. Section 5 concludes the paper.

2. Regular edge labeling of planar triangular graphs. Let $G = (V, E)$ be a planar graph. Consider a fixed plane embedding of G. The embedding divides the plane into a number of regions. The unbounded region is called the *exterior face*. Other regions are called *interior faces*. The vertices and the edges on the exterior face are called *exterior vertices* and *exterior edges*, respectively. A cycle C of G divides the plane into its interior region and exterior region. If C contains at least one vertex in its interior region, C is called a *separating cycle* of G. For each vertex v, $N(v)$ denotes the set of neighbors of v and Star(v) denotes the set of edges incident to v. Whenever these notations are used, it is understood that the members in the set are listed in counterclockwise order around v in the embedding.

Consider a planar graph $H = (V, E)$. Let v_0, v_1, v_2, v_3 be four vertices on the exterior face of H in counterclockwise order. Let P_i ($i = 0, 1, 2, 3$) be the four paths on the exterior face of H consisting of the vertices between v_i and v_{i+1} (where the addition is mod 4). We seek a rectangular dual R_H of H such that the four vertices v_0, v_1, v_2, v_3 correspond to the four corner rectangles of R_H and the vertices on P_0 (P_1, P_2, P_3, respectively) correspond to the rectangles located along the north (west, south, east, respectively) boundary of R_H. Necessary and sufficient conditions for testing whether H has a rectangular dual were discussed in [2], [3], [6]. These conditions, however, can be easily reduced to the following simpler form.

In order to simplify the problem, we modify H as follows: Add four new vertices v_N, v_W, v_S, v_E, and connect v_N (v_W, v_S, v_E, respectively) to every vertex on P_0 (P_1, P_2, P_3, respectively). Then add four new edges (v_N, v_W), (v_W, v_S), (v_S, v_E), (v_E, v_N). Let G be the resulting graph. It is easy to see that H has a rectangular dual R_H, with v_0, v_1, v_2, v_3 corresponding to the four corner rectangles of R_H if and only if G has a rectangular dual R with exactly four rectangles on the boundary of R. Without loss of generality, we will discuss only planar graphs with exactly four vertices on their exterior faces.

If G has a rectangular dual R, then every face of G, except the exterior face, must be a triangle (since no four rectangles of R meet at the same point). Moreover, since at least four rectangles are needed to fully enclose some nonempty area on the plane, any separating cycle of G must have length at least 4. The following theorem states that these two conditions are also sufficient for G to have a rectangular dual.

THEOREM 2.1 [6]. *A planar graph $G = (V, E)$ has a rectangular dual R with four rectangles on the boundary of R if and only if the following two conditions hold*: (1) *Every interior face of G is a triangle, and the exterior face of G is a quadrangle;* (2) *G has no separating triangles.*

A different form of Theorem 2.1 was given in [2], [3]. A graph satisfying the two conditions of Theorem 2.1 is called a *proper triangular planar* (PTP) graph. From now on we discuss only such graphs.

DEFINITION 1. A *regular edge labeling* (REL) of a PTP graph $G = (V, E)$ is a partition of the interior edges of G into two subsets $\{T_1, T_2\}$ of directed edges such that the following hold:

(1) For each interior vertex v the edges in Star(v) appear in counterclockwise order around v as follows: a set of edges in T_1 leaving v; a set of edges in T_2 entering v; a set of edges in T_1 entering v; a set of edges in T_2 leaving v.

(2) All interior edges incident to v_N are in T_1 and entering v_N. All interior edges incident to v_W are in T_2 and leaving v_W. All interior edges incident to v_S are in T_1 and leaving v_S. All interior edges incident to v_E are in T_2 and entering v_E.

From Theorem 2.1 we can easily prove the following.

THEOREM 2.2. *Every* PTP *graph $G = (V, E)$ has a* REL.

Proof. By Theorem 2.1, G has a rectangular dual R. For each $v \in V$ let $R(v)$ denote the rectangle in R corresponding to v. For each interior vertex v label each edge $(v, u) \in$ Star(v) as follows: If $R(u)$ is above $R(v)$, e is in T_1 and directed leaving v. If $R(u)$ is below $R(v)$, e is in T_1 and directed entering v. If $R(u)$ is to the left of $R(v)$, e is in T_2 and directed entering v. If $R(u)$ is to the right of $R(v)$, e is in T_2 and directed leaving v. This labeling satisfies the two conditions of Definition 1. $\square$

Although Theorem 2.2 is proved from Theorem 2.1, our algorithm goes another way around: We find a REL of G first and construct a rectangular dual of G from the REL. We first prove some properties of the REL.

Let $G = (V, E)$ be a PTP graph, and let $\{T_1, T_2\}$ be a REL of G. Let G_1 be the directed subgraph of G induced by the edges in T_1 and the four exterior edges directed as $v_S \to v_W$, $v_W \to v_N$, $v_S \to v_E$, $v_E \to v_N$. Let E_1 denote the edge set of G_1. (E_1 is the union of T_1 and the four exterior edges.) Let G_2 be the directed subgraph of G induced by the edges in T_2 and the four exterior edges directed as $v_W \to v_S$, $v_S \to v_E$, $v_W \to v_N$, $v_N \to v_E$. Let E_2 denote the edge set of G_2. (E_2 is the union of T_2 and the four exterior edges.) We will call G_1 the *S–N net* and G_2 the *W–E net* of G derived from the REL $\{T_1, T_2\}$.

Part (1) of Fig. 1 shows a PTP graph G. An S–N net G_1 and the corresponding W–E net G_2 are shown in parts (3) and (4). (Ignore the integers in the small boxes in parts (3) and (4) for now.)

LEMMA 2.3. (1) *G_1 is acyclic, with v_S as the only source and v_N as the only sink.*

(2) *G_2 is acyclic, with v_W as the only source and v_E as the only sink.*

Proof. The lemma is proved by way of contradiction. Suppose either G_1 or G_2 contains a directed cycle. Let $C = \{v_1, \ldots, v_l\}$ be such a cycle such that the total number of vertices that are on C or in the interior of C is minimized. Without loss of generality, suppose C is a cycle in G_1 and is directed in clockwise direction. (The proofs of other cases are similar.)

Case 1: C contains no vertices in its interior. If $l = 3$, then there is no edge in T_2 leaving v_2. This contradicts condition (1) of Definition 1. Suppose $l > 3$. Then there is an edge $e = (v_i, v_j) \in E$ contained in the interior of C. e cannot be in T_1 since otherwise we would have a smaller cycle in G_1, which contradicts the choice of C. So e must be in T_2. But regardless of the direction of e in T_2, condition (1) of Definition 1 is violated either at v_i or at v_j.

Case 2: C contains at least one vertex u_1 in its interior. Start at u_1; we can reach another vertex u_2 by using a T_2 edge. Similarly, from u_2 we can reach another vertex u_3 by using a T_2 edge. Since every vertex u has an incident edge in T_2 leaving u, this process can be repeated again and again. Since C is the smallest cycle in both G_1 and G_2, we cannot have a cycle in G_2 completely contained in the interior of C. Thus we must reach a vertex $v_j \in C$. Then condition (1) of Definition 1 is violated at v_j.

Since we get contradictions in all cases, both G_1 and G_2 are acyclic. Since every vertex v, other than v_S and v_N, has indegree and outdegree at least 1 in G_1, v_S is the only source and v_N is the only sink of G_1. Similarly, v_W is the only source and v_E is the only sink of G_2. □

Both G_1 and G_2 are the so-called *s–t planar graphs*. (An s–t planar graph is a directed acyclic planar graph with exactly one source s and exactly one sink t, and both s and t are on the exterior face of the graph.) The properties of these graphs have been studied in [7], [10], [13]. By using these properties, the structure of G_1 can be summarized as follows:

(a) For each vertex v other than v_S and v_N the edges entering v appear consecutively around v in G_1. The edges leaving v appear consecutively around v in G_1. Let e_1 and e_2 be the leftmost and the rightmost edges in G_1 entering v. Let e_3 and e_4 be the leftmost and the rightmost edges in G_1 leaving v. The face of G_1 with e_1 and e_3 on its boundary is denoted by left(v). The face of G_1 with e_2 and e_4 on its boundary is denoted by right(v). We use f_W to denote left(v_W) and f_E to denote right(v_E). (In other words, the exterior face is divided into two faces f_W and f_E.) For the vertices v_S and v_N define left$(v_S) =$ left$(v_N) = f_W$ and right$(v_S) =$ right$(v_N) = f_E$.

(b) For each interior face f of G_1 the boundary of f consists of two directed paths P_1 and P_2 starting at the same vertex and ending at the same vertex (see part (3) of Fig. 1).

Similarly, the structure of G_2 can be summarized as follows.

(c) For each vertex v other than v_W and v_E the edges entering v appear consecutively around v in G_2. The edges leaving v appear consecutively around v in G_2. Let e_1 and e_2 be the leftmost and the rightmost edges in G_2 entering v. Let e_3 and e_4 be the leftmost and the rightmost edges in G_2 leaving v. The face of G_2 with e_1 and e_3 on its boundary is denoted by above(v). The face of G_2 with e_2 and e_4 on its boundary is denoted by below(v). We use f_N to denote above(v_N) and f_S to denote below(v_S). (In other words, the exterior face is divided into two faces f_N and f_S.) For the vertex v_W and v_E, define above$(v_W) =$ above$(v_E) = f_N$ and below$(v_W) =$ below$(v_E) = f_S$.

(d) For each interior face g of G_2 the boundary of g consists of two directed paths P_1 and P_2 starting at the same vertex and ending at the same vertex (see part (4) of Fig. 1).

3. Algorithm. Let $G = (V, E)$ be a PTP graph, and let $\{T_1, T_2\}$ be a REL of G. Consider the S–N net G_1 derived from $\{T_1, T_2\}$. For each edge $e \in E_1$, let left(e) (right(e), respectively) denote the face of G_1 on the left (right, respectively) of e. Define the *dual graph* of G_1, denoted by G_1^*, as follows. The node set of G_1^* is the set of the interior faces of G_1 plus the two exterior faces f_W and f_E. For each edge $e \in E_1$ there is a corresponding arc e^* in G_1^* directed from the face left(e) to the face right(e). Since G_1 is an s–t graph, G_1^* is also an s–t graph [11]. Namely, G_1^* is a directed acyclic planar graph with f_W as the only source and f_E as the only sink.

Similarly, define the dual graph G_2^* of G_2 as follows. For each edge $e \in E_2$ let above(e) (below(e), respectively) denote the face of G_2 on the left (right, respectively) of e. The nodes of G_2^* are the interior faces of G_2 plus the two exterior faces f_S and f_N. For each edge $e \in E_2$ there is a directed arc e^* in G_2^* from the face below(e) to the face above(e). G_2^* is a directed acyclic planar graph with f_S as the only source and f_N as the only sink.

DEFINITION 2. A *consistent numbering of order* k_1 of G_1^* is a surjective mapping F_1 from the node set of G_1^* to the set of integers $\{0, 1, \ldots, k_1\}$ such that (1) $F_1(f_W) = 0$ and $F_1(f_E) = k_1$, and (2) if there is an arc from the node f to the node g in G_1^*, then $F_1(f) < F_1(g)$.

For an example, a topological ordering [1], [4] of G_1^* is a consistent numbering. As another example, if we define $F_1(f)$ to be the length of the longest path in G_1^* from f_W to f (with $F_1(f_W) = 0$), F_1 is also a consistent numbering. Define the *length* of G_1^* to be the length of the longest path from f_W to f_E in G_1^*. Note that if the length of G_1^* is k, then any consistent numbering of G has order at least k by Definition 2. The consistent numbering of G_2^* can be defined similarly. We now can present our algorithm as follows.

```
ALGORITHM DUAL:
Input: A PTP graph $G = (V, E)$.
   (1) Find a REL $\{T_1, T_2\}$ of $G$.
  (2a) Construct the S–N net $G_1$ derived from $\{T_1, T_2\}$ and its dual graph $G_1^*$.
  (2b) Compute a consistent numbering $F_1$ of $G_1^*$. Let $k_1 = F_1(f_E)$.
  (2c) For each vertex $v \in V$ other than $v_S$ and $v_N$ let $f_1 =$ left($v$) and $f_2 =$ right($v$) in
       $G_1$. Let $x_1(v) = F_1(f_1)$ and $x_2(v) = F_1(f_2)$. Define $x_1(v_N) = x_1(v_S) = 1$ and
       $x_2(v_N) = x_2(v_S) = k_1 - 1$.
  (3a) Construct the W–E net $G_2$ derived from $\{T_1, T_2\}$ and its dual graph $G_2^*$.
  (3b) Compute a consistent numbering $F_2$ of $G_2^*$. Let $k_2 = F_2(f_N)$.
  (3c) For each vertex $v \in V$ let $g_1 =$ below($v$) and $g_2 =$ above($v$) in $G_2$. Let $y_1(v) =$
       $F_2(g_1)$ and $y_2(v) = F_2(g_2)$.
   (4) For each vertex $v \in V$ assign $v$ a rectangle $R(v)$ bounded by two vertical lines with
       $x$-coordinates $x_1(v), x_2(v)$ and two horizontal lines with $y$-coordinates $y_1(v), y_2(v)$.
End.
```

In §4 we will prove that the algorithm DUAL correctly computes a $k_1 \times k_2$ rectangular dual of G. For an example, the rectangular dual shown in part (2) of Fig. 1 is constructed from the information indicated in parts (3) and (4). In this example $F_1(f)$ is the length of the longest path from f_W to f in G_1^*. $F_2(g)$ is the length of the longest path from f_S to g in G_2^*. In part (3) the integers in the small boxes are the F_1-numbers of the faces of G_1. In part (4) the numbers in the small boxes are the F_2-numbers of the faces of G_2.

To implement the algorithm DUAL, we assume the embedding of G is given. (If not, it can be computed by using the well-known linear-time planarity algorithms.) Step 1 can be carried out by using the $O(n)$ algorithm in [3]. (The algorithm in [3] finds the set T_1, which is called the *path digraph.*) For step (2a) the graph G_1 and the dual graph G_1^* can be constructed from the embedding information of G. The implementation of step (2b) depends on the choice of F_1. The most natural choice, the length of the longest path from f_S to f in G_1^*, can be calculated according to the topological ordering of G_1^* [1], [4]. For step (2c) the left face and the right face of each vertex can be determined from the embedding information. All these steps take $O(n)$ time. Step (3) can be implemented similarly. Step (4) clearly takes $O(n)$ time. Thus the total running time of the algorithm is $O(n)$.

4. Correctness proof.

Before we prove the correctness of the algorithm DUAL, we need several definitions. Consider an S–N net G_1 of G. An *S–N path* is a directed path P in G_1 from v_S to v_N. Let P_1 and P_2 be two S–N paths of G_1. (P_1 and P_2 are not necessarily edge disjoint.) We say P_2 is *to the right of* P_1 if every edge $e \in P_2$ is either on P_1 or to the right of P_1.

DEFINITION 3. A *path system* of G_1 is a collection $\{P_0, \dots, P_{l-1}\}$ of S–N paths of G_1 such that

(1) The union of the paths P_i $(0 \le i \le l-1)$ is the edge set E_1 of G_1;

(2) P_i is to the right of P_{i-1} for $1 \le i \le l-1$.

DEFINITION 4. Let F_1 be a consistent numbering of G_1^* of order k_1. For each $0 \le i \le k_1$

(1) define $\mathrm{FACE}_i = \{f \mid f \text{ is a face of } G_1 \text{ with } F_1(f) = i\}$;

(2) define $\mathrm{LB}_i = \{e \in E_1 \mid e \text{ is on the left boundary of a face } f \in \mathrm{FACE}_i\}$;

(3) define $\mathrm{RB}_i = \{e \in E_1 \mid e \text{ is on the right boundary of a face } f \in \mathrm{FACE}_i\}$;

(4) define the *standard path system* $\{P_0, \dots, P_{k_1-1}\}$ of G_1 as $P_0 = RB_0$ and $P_i = P_{i-1} - LB_i \cup RB_i$ for $1 \le i \le k_1 - 1$.

Note that $\mathrm{FACE}_0 = \{f_W\}$, $\mathrm{LB}_0 = \emptyset$, $\mathrm{RB}_0 = \{(v_S, v_W), (v_W, v_N)\}$ and that $\mathrm{FACE}_{k_1} = \{f_E\}$, $\mathrm{LB}_{k_1} = \{(v_S, v_E), (v_E, v_N)\}$, $\mathrm{RB}_{k_1} = \emptyset$.

We make the following observations. Consider any edge $e \in E_1$. Let $g_1 = \mathrm{left}(e)$, $g_2 = \mathrm{right}(e)$, $p = F_1(g_1)$, and $q = F_1(g_2)$. Since e is on the right boundary of g_1 and on the left boundary of g_2, $e \in \mathrm{RB}_p$ and $e \in \mathrm{LB}_q$. Since e's corresponding arc e^* is directed from g_1 to g_2 in G_1^*, we have $p < q$. So $\mathrm{LB}_i \cap \mathrm{RB}_i = \emptyset$ for all $0 \le i \le k_1$. Since each $e \in E_1$ is in exactly one RB_i $(0 \le i \le k_1 - 1)$, E_1 is the disjoint union of the sets RB_i $(0 \le i \le k_1 - 1)$. Similarly, E_1 is the disjoint union of the sets LB_i $(1 \le i \le k_1)$.

LEMMA 4.1. *Let F_1 be a consistent numbering of G_1^* of order k_1. Then the following hold:*

(a) *The standard path system $\{P_0, P_1, \dots, P_{k_1-1}\}$ in Definition 4 is a path system of G_1.*

(b) *For each vertex $v \in V$ let $f_1 = \mathrm{left}(v)$ and $f_2 = \mathrm{right}(v)$ in G_1. Define $x_1(v) = F_1(f_1)$ and $x_2(v) = F_1(f_2)$. Then v is on the path P_i if and only if $x_1(v) \le i \le x_2(v) - 1$.*

Proof. (a) We prove by induction that the following hold for each i $(0 \le i \le k_1 - 1)$: (1) P_i is an S–N path of G_1, and (2) $\mathrm{LB}_{i+1} \subseteq P_i$.

Base step $i = 0$. (1) $P_0 = \{(v_S, v_W), (v_W, v_N)\}$ is an S–N path of G_1.

(2) Let e be an edge in LB_1. Then e is on the left boundary of a face $f \in \mathrm{FACE}_1$. Let e^* be the arc in G_1^* corresponding to e. Since $F_1(f) = 1$, e^* must be directed from f_E to f in G_1^*. This implies $e \in RB_0 = P_0$. Since this is true for all $e \in \mathrm{LB}_1$, we have $\mathrm{LB}_1 \subseteq P_0$.

Induction step. Assume the claims (1) and (2) are true for $i - 1$; we show they are true for i.

(1) By the induction hypothesis P_{i-1} is an S–N path. Suppose $\mathrm{FACE}_i = \{h_1, \dots, h_l\}$ for some l. Let A_j and B_j be the left and the right boundary of h_j, respectively $(1 \le j \le l)$. Since (2) is true for P_{i-1}, the paths A_j $(1 \le j \le l)$ are subpaths of P_{i-1}. Since A_j and B_j $(1 \le j \le l)$ start at the same vertex and end at the same vertex and P_i is obtained from P_{i-1} by replacing each A_j with B_j, P_i is an S–N path of G_1.

(2) Consider any edge $e \in \mathrm{LB}_{i+1}$. Let $g_1 = \mathrm{left}(e)$ and $g_2 = \mathrm{right}(e)$. Since $e \in \mathrm{LB}_{i+1}$, $F_1(g_2) = i + 1$. Suppose $F_1(g_1) = q$ for some q. Then $e \in \mathrm{RB}_q$. Since e^* is directed from g_1 to g_2 in G_1^*, $q < i + 1$. By definition e is added into P_q and deleted when P_{i+1} is constructed. So e is in P_r for all $q \le r \le i$. In particular, $e \in P_i$. Thus $\mathrm{LB}_{i+1} \subseteq P_i$. This completes the induction.

Each $e \in E_1$ is in some RB_i $(0 \leq i \leq k_1 - 1)$ and hence in P_i. Therefore, E_1 is the union of P_i's $(i = 0, \dots, k_1 - 1)$. From the definition of P_i it is easy to see P_i is to the right of P_{i-1} for all $1 \leq i \leq k_1 - 1$. Thus $\{P_0, \dots, P_{k_1-1}\}$ is a path system of G_1.

(b) Since v is on the right boundary of f_1, it is added into the path $P_{x_1(v)}$. Since v is on the left boundary of f_2, it is removed when the path $P_{x_2(v)}$ is constructed. Hence v is on the paths P_i for exactly those indices i with $x_1(v) \leq i \leq x_2(v) - 1$. □

All of the preceding discussion can be repeated on the W–E net G_2 and its dual graph G_2^*. Let F_2 be a consistent numbering of G_2^* of order k_2. We can construct the standard path system $\{Q_0, \dots, Q_{k_2-1}\}$ of G_2 from F_2 similar to Definition 4. For each vertex v of G let $g_1 = \text{below}(v)$ and $g_2 = \text{above}(v)$ in G_2. Define $y_1(v) = F_2(g_1)$ and $y_2(v) = F_2(g_2)$. In a manner similar to that of Lemma 4.1, it can be shown that v is on the path Q_j if and only if $y_1(v) \leq j \leq y_2(v) - 1$.

LEMMA 4.2. *Let G_1 and G_2 be the S–N net and the W–E net derived from a* REL $\{T_1, T_2\}$ *of G. Let F_1 and F_2 be two consistent numberings of G_1^* and G_2^*, respectively. Let u and v be two vertices of G.*

(1) *If $(u, v) \in T_2$ and is directed from u to v in G_2, then $x_2(u) = x_1(v)$.*

(2) *If there is a directed path from u to v in G_2 with length at least* 2, *then $x_2(u) < x_1(v)$.*

(3) *If $(u, v) \in T_1$ and is directed from u to v in G_1, then $y_2(u) = y_1(v)$.*

(4) *If there is a directed path from u to v in G_1 with length at least* 2, *then $y_2(u) < y_1(v)$.*

Proof. We prove only (1) and (2). The proofs of (3) and (4) are similar.

(1) Suppose $(u, v) \in T_2$ and is directed from u to v. Let e_1 (e_2, respectively) be the rightmost outgoing (incoming, respectively) edge of u in G_1. Let e_3 (e_4, respectively) be the leftmost outgoing (incoming, respectively) edge of v in G_1. Let f be the face of G_1 with e_1, e_2, e_3, and e_4 on its boundary. Then $f = \text{right}(u) = \text{left}(v)$ and $x_2(u) = x_1(v) = F_1(f)$.

(2) Let $u = u_0, u_1, \dots, u_p = v$ $(p \geq 2)$ be a directed path in G_2 from u to v. By (1), $x_2(u_{l-1}) = x_1(u_l)$ for all $1 \leq l \leq p$. Since $x_1(u_l) < x_2(u_l)$ for all $0 \leq l \leq p$ and $p \geq 2$, we have $x_2(u) = x_2(u_0) < x_1(u_p) = x_1(v)$. □

From Lemmas 4.1 amd 4.2, we can prove the following.

THEOREM 4.3. *The algorithm* DUAL *correctly constructs a rectangular dual of G in $O(n)$ time.*

Proof. We have shown the algorithm can be implemented in linear time. We next prove the correctness of the algorithm. Let $\{P_0, \dots, P_{k_1-1}\}$ be the standard path system of G_1 derived from F_1 and let $\{Q_0, \dots, Q_{k_2-1}\}$ be the standard path system of G_2 derived from F_2. In the rectangular dual R constructed by the algorithm DUAL, each S–N path P_i $(0 \leq i \leq k_1 - 1)$ corresponds to a vertical strip bounded by the two vertical lines with x-coordinates i and $i + 1$. Each W–E path Q_j $(0 \leq j \leq k_2 - 1)$ corresponds to a horizontal strip bounded by the two horizontal lines with y-coordinates j and $j + 1$. Let $R(v)$ be the rectangle with coordinates $x_1(v), x_2(v), y_1(v), y_2(v)$. To show the set $\{R(v) \mid v \in V\}$ forms a rectangular dual of G, we need to prove the following claims.

(1) We show that each unit square R_{ij} $(0 \leq i \leq k_1 - 1$ and $0 \leq j \leq k_2 - 1)$ with x-coordinates $i, i + 1$ and y-coordinates $j, j + 1$ is occupied by a rectangle $R(v)$ for a unique $v \in V$. Consider the S–N path P_i and the W–E path Q_j. Except for the four special cases (a) $i = 0, j = 0$, (b) $i = k_1 - 1, j = 0$, (c) $i = 0, j = k_2 - 1$, (d) $i = k_1 - 1, j = k_2 - 1$, P_i and Q_j intersect at a unique vertex $v \in V$. By Lemma 4.1 (b), v is the unique vertex satisfying all of the following inequalities: $x_1(v) \leq i$, $i + 1 \leq x_2(v)$, $y_1(v) \leq j$, $j + 1 \leq y_2(v)$. Hence $R(v)$ is the unique rectangle occupying R_{ij}. For the four special cases this claim is not true. (For example, both v_S and v_W belong to the intersection of P_0 and Q_0.) The four special cases correspond to the four corner unit squares of R. However, the special definition $x_1(v_S) = x_1(v_N) = 1$ and $x_2(v_S) = x_2(v_N) = k_1 - 1$ at step (2b) of the algorithm DUAL

ensures that each of the four unit corner squares of R is occupied by one of $R(v_W)$, $R(v_E)$.

(2) We show if $e = (u, v)$ is an edge in G, then the corresponding rectangles $R(u)$ and $R(v)$ share a common boundary. If e is an exterior edge, this is ensured by the definition of $R(v_N)$, $R(v_W)$, $R(v_S)$, $R(v_E)$. So assume e is an interior edge. Suppose $e \in T_1$ and is directed from u to v. (Other cases are similar.) Let P_i be an S–N path containing e. By Lemma 4.1 (b), $x_1(u) \le i \le x_2(u) - 1$ and $x_1(v) \le i \le x_2(v) - 1$. By Lemma 4.2 (3), $y_2(u) = y_1(v) = j$ for some j. Thus $R(u)$ and $R(v)$ have the line segment connecting two points (i, j) and $(i + 1, j)$ as their common boundary.

(3) We show that if two rectangles $R(u)$ and $R(v)$ share a common boundary, then (u, v) is an edge in G. Assume the common boundary of $R(u)$ and $R(v)$ contains a horizontal line segment I connecting two points (i, j) and $(i + 1, j)$. (Other cases are similar.) Since $x_1(u) \le i, i + 1 \le x_2(u)$ and $x_1(v) \le i, i + 1 \le x_2(v)$, both u and v are on the S–N path P_i. We need to show (u, v) is an edge on P_i. If not, there exists a directed path from u to v in G_1 of length at least 2. By Lemma 4.2 (4) we have $y_2(u) < y_1(v)$. This contradicts the assumption that $R(u)$ and $R(v)$ share I as their common boundary.

Thus $e = (u, v)$ is an edge of G if and only if $R(u)$ and $R(v)$ share a common boundary. Hence $\{R(v) \mid v \in V\}$ form a rectangular dual of G. □

5. Conclusion. A new linear-time algorithm for finding a rectangular dual R of a proper triangular planar graph G is presented. The algorithm is based on new understanding of the structure of PTP graphs, which is of independent interest. Our algorithm is conceptually simple. The coordinates of the rectangles of the rectangular dual produced by the algorithm are integers and carry clear combinatorial meaning. This allows us to discuss the heuristics for reducing the width and the height of the rectangular dual R. Several related optimization problems are interesting and deserve further study. Let $w(R)$ and $h(R)$ denote the width and the height of R. How do we find a rectangular dual R of G so that $w(R)$ is minimized, $w(R) + h(R)$ is minimized, or $w(R)h(R)$ is minimized?

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] J. Bhasker and S. Sahni, *A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph*, Networks, 17 (1987), pp. 307–317.

[3] ———, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica, 3 (1988), pp. 247–278.

[4] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Potomac, MD, 1988.

[5] W. R. Heller, G. Sorkin, and K. Mailing, *The planar package planner for system designers*, in Proc. 19th Annual IEEE Design Automation Conference, Institute of Electrical and Electronics Engineers, New York, 1982, pp. 253–260.

[6] K. Koźmiński and E. Kinnen, *Rectangular duals of planar graphs*, Networks, 15 (1985), pp. 145–157.

[7] D. Kelly and I. Rival, *Planar lattices*, Canad. J. Math., 27 (1975), pp. 636–665.

[8] Y.-T. Lai and S. M. Leinwand, *A theory of rectangular dual graphs*, Algorithmica, 5 (1990), pp. 467–483.

[9] K. Mailing, S. H. Mueller, and W. R. Heller, *On finding most optimal rectangular package plans*, in Proc. 19th Annual IEEE Design Automation Conference, Institute of Electrical and Electronics Engineers, New York, 1982, pp. 663–670.

[10] F. P. Preparata and R. Tamassia, *Fully dynamic techniques for point location and transitive closure in planar structures*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1988, pp. 558–567.

[11] P. Rosenstiehl and R. E. Tarjan, *Rectilinear planar layouts and bipolar orientations of planar graphs*, Discrete Comput. Geom., 1 (1985), pp. 343–353.

[12] C. Thomassen, *Interval representations of planar graphs*, J. Combin. Theory Ser. B, 40 (1986), pp. 9–20.

[13] R. TAMASSIA AND J. S. VITTER, *Optimal parallel algorithms for transitive closure and point location in planar structures*, in Proc. 1989 ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1989, pp. 399–407.

[14] R. TAMASSIA, *Drawing algorithms for planar s-t graphs*, Australasian J. Combin., 2 (1990), pp. 217–236.

[15] R. TAMASSIA AND I. TOLLIS, *Tessellation representations of planar graphs*, in Proc. 27th Allerton Conference, University of Illinois at Urbana, Champaign, 1989, pp. 48–57.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1227–1250, December 1993

FAST AND EFFICIENT PARALLEL SOLUTION OF SPARSE LINEAR SYSTEMS*

VICTOR PAN† AND JOHN REIF‡

Abstract. This paper presents a parallel algorithm for the solution of a linear system $A\mathbf{x} = \mathbf{b}$ with a sparse $n \times n$ symmetric positive definite matrix A, associated with the graph $G(A)$ that has n vertices and has an edge for each nonzero entry of A. If $G(A)$ has an $s(n)$-separator family and a known $s(n)$-separator tree, then the algorithm requires only $O(\log^3 n)$ time and $(|E| + M(s(n)))/\log n$ processors for the evaluation of the solution vector $\mathbf{x} = A^{-1}\mathbf{b}$, where $|E|$ is the number of edges in $G(A)$ and $M(n)$ is the number of processors sufficient for multiplying two $n \times n$ rational matrices in time $O(\log n)$. Furthermore, for this computational cost the algorithm computes a recursive factorization of A such that the solution of any other linear system $A\mathbf{x} = \mathbf{b}'$ with the same matrix A requires only $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors.

Key words. parallel algorithms, sparse linear systems, nested dissection, parallel complexity, graph separators

AMS subject classifications. 68Q25, 68Q22, 65Y05, 65Y20, 65F05, 65F50

1. Introduction. Recently, it has become feasible to construct computer architectures with a large number of processors. We assume the parallel RAM machine model of [BGH] (see also [EG] and [KR]), where in each step each processor can perform a single addition, subtraction, multiplication, or division over the rationals. It is natural to study the efficient use of this parallelism for solving fundamental numerical problems such as the following.

(1) MULT: Given a pair of $n \times n$ rational matrices A and B, output AB.

(2) INVERT: Given an $n \times n$ rational matrix $A = (a_{ij})$, output A^{-1} within a prescribed error bound ϵ if A is well conditioned (see Definition 2.1 below); else output: ILL-CONDITIONED.

(3) LINEAR-SOLVE: Given a well-conditioned $n \times n$ matrix A and a column vector $\mathbf{b}$ of dimension n, find $\mathbf{x} = A^{-1}\mathbf{b}$ within a prescribed error bound ϵ.

Here and hereafter we assume that $\tilde{A}^{-1}$ and $\tilde{\mathbf{x}}$ approximate A^{-1} and $\mathbf{x}$ within ε if $\|\tilde{A}^{-1} - A^{-1}\| \leq \varepsilon\|A^{-1}\|$ and $\|\tilde{\mathbf{x}} - \mathbf{x}\| \leq \varepsilon\|A^{-1}\|\ \|\mathbf{b}\|$ for a fixed pair of consistent matrix and vector norms.

Our main result in this paper is a numerically stable parallel algorithm for LINEAR-SOLVE for a sparse symmetric positive definite linear system that can be solved by a sequential algorithm based on the generalized nested dissection of the associated graph. In the remainder of this introduction and, more formally, in §3 we compare our algorithm with the previous results, specify the complexity estimates, show our improvement, and list some applications. In particular, in §3 we comment on the classes of the LINEAR-SOLVE instances for which our algorithm is effective.

The nested dissection techniques were first proposed in [Ge] for grid graphs and were then extended to graphs with small separators in [LRT] (see also [R] and the excellent text [GeL]

*Received by the editors March 16, 1987; accepted for publication (in revised form) June 9, 1992. The results of the preliminary version of this paper were presented at the 17th Annual ACM Symposium on Theory of Computing, 1985.

†Department of Mathematics, Lehman College, City University of New York, New York, New York 10468, and Department of Computer Science, State University of New York at Albany, Albany, New York 12222. (vpan@lcvax.bitnet). The work of this author was supported by National Science Foundation grants MCS-8203232, DCR-8507573, CCR-8805782, and CCR-9020690 and by Professional Staff Congress–City University of New York awards 661340, 668541, and 669290.

‡Department of Computer Science, Duke University, Durham, North Carolina 27706. The work of this author was supported by National Science Foundation grant CCR-83-09911, Office of Naval Research contracts N00014-87-K-0310 and N00014-88-K-0458, U.S. Air Force Office of Scientific Research contract AFOSR-87-0386, Defense Advanced Research Projects Agency contracts DAAL03-88-K-1095 and N0014-88-K-0458, and National Aeronautics and Space Administration CESDIS subcontract 550-63 5-30428 URSA.

and an alternative version of the nested dissection algorithm in [GT]). Many applications to the sciences and engineering require the solution of such large linear systems; such systems are frequently so large that parallel implementation of the (generalized) nested dissection algorithms is necessary in order to make the solution feasible. (We recall some examples of such problems in §3.)

Work on parallel sparse matrix algorithms can be traced back, at least, to [Ca]. The extension of the idea of nested dissection from the sequential to the parallel case was not immediate since many sets of separators must be eliminated in each parallel step. Linear-time parallel algorithms based on the nested dissection of grids were first described in [Li1] and [Ga]. The survey paper [OV] gives references to early attempts at parallelizing the LINEAR-SOLVE algorithms by nested dissection. Here and hereafter, by "parallel nested dissection" we mean a parallel algorithm for solving sparse linear systems and not a parallel algorithm for computing a dissection ordering. The subsequent literature on the parallel implementation of the nested dissection algorithms includes the papers [GHLN] and [ZG], which give a parallel time bound of $O(\sqrt{n})$ for grid graphs.

In the proceedings version of our paper [PR1], nested dissection was applied for the first time to yield a numerically stable and processor efficient parallel algorithm for sparse LINEAR-SOLVE with poly-log time bounds, thus reaching (within poly-log factors) the *optimum* bounds for both time and the number of processors. Furthermore, our nested dissection parallel algorithm has been applied to a much larger class of graphs than grid graphs, including planar graphs and $s(n)$-separatable graphs (see Definition 3.1 below) with $s(n) = o(n)$, whereas in the previous literature the parallel nested dissection was restricted to grid graphs. Such an enhanced generality required us to exploit the intricate construction of [LRT] (rather than the simpler constructions of the earlier nested dissection papers, more familiar to the numerical analysis audience); to devise the desired processor efficient version of this approach, we had to elaborate the construction of [LRT] by including the recursive factorization of the input matrix and by proving several properties of the associated graphs. This paper assumes the reader has some exposure to graph techniques. These generalizations of the nested dissection algorithms, including the recursive factorization techniques, are required in several important applications, particularly path-algebra computation in graphs (see [T1], [T2], [PR1], [PR4], and §3 below).

Remark 1.1. Some readers may agree to sacrifice the generality of the results in order to simplify the graph techniques involved. Such readers may replace our Definition 3.1 of separators in graphs by the definition from [GT]. The difference between these approaches is that our definition requires the inclusion of the separator subgraph S into both subgraphs G_1 and G_2, otherwise separated from each other by S in the original graph G, whereas the definition of [GT] requires the elimination of all of the vertices of S and all of the edges adjacent to them from both subgraphs G_1 and G_2. The resulting construction of [GT] is a little simpler than ours, and its application decreases by a constant factor the complexity of the performance of the algorithm in the case of planar graphs G, but the results are applied to a class of graphs that is strictly more narrow than the class we address. As in our proceedings paper, [PR1], we extend to parallel computation the more general algorithm of [LRT], rather than one of [GT], but we demonstrate the vertex elimination construction of [GT] for a 7×7 grid graph in our Figs. 1–5 below (in this case, using the version of [GT], rather than ours, made our display simpler and more compact).

[Li2] and [OR] describe two recent implementations of the parallel nested dissection algorithm on massively parallel SIMD machines. The first implementation is very general and applies to any $s(n)$-separatable graph; it runs on the CONNECTION MACHINE, which is a hypercube-connected parallel machine with 65,536 processors; the second implementation is

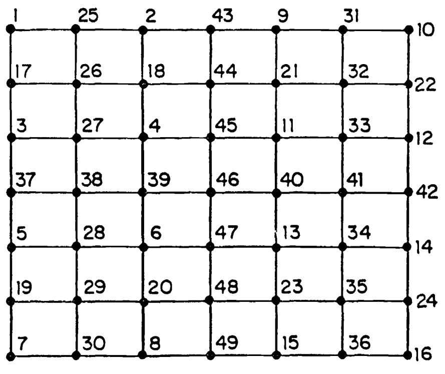


FIG. 1(a). *7 × 7 grid graph G_0 with elimination numbering.*

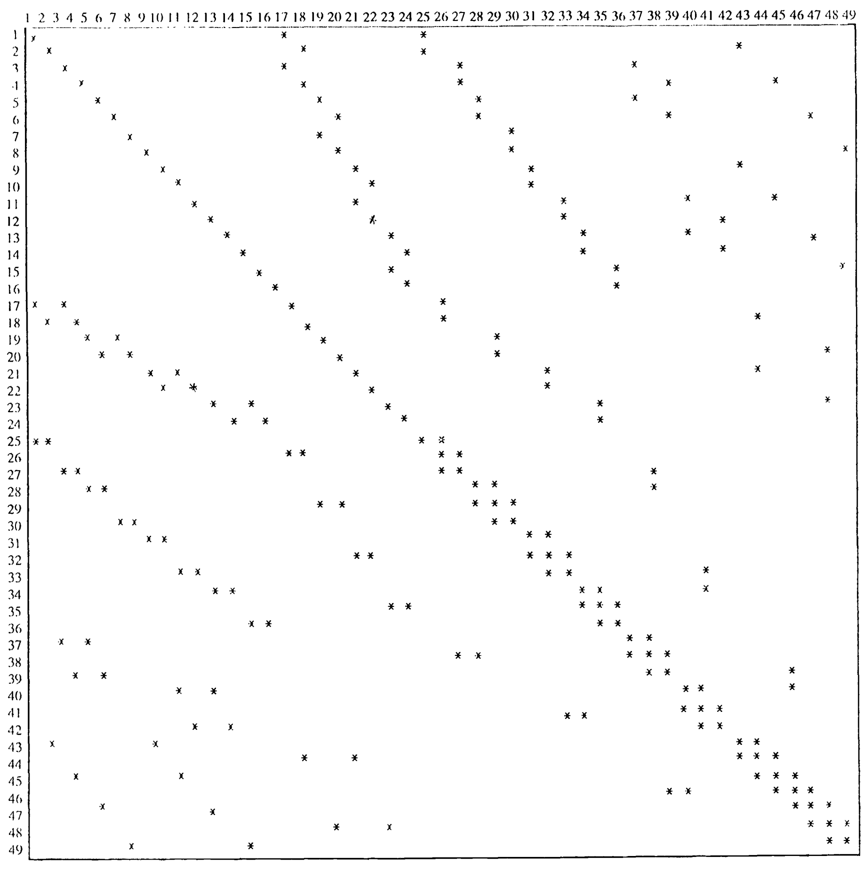


FIG. 1(b). *The matrix A_0. * denotes a nonzero entry. G_0 is the sparsity graph of the matrix A_0.*

restricted to grid graphs and runs on the MPP, which is a grid-connected parallel machine with 16,384 processors. Both implementations represent a version of the parallel nested dissection algorithm using $O(s(n))$ time and $s(n)^2$ processors (see the end of §3).

In the papers [PR2]–[PR4], [PR6], [PR7] we extend our parallel nested dissection algorithm to the linear least-squares problem, to the linear programming problem, and to path-

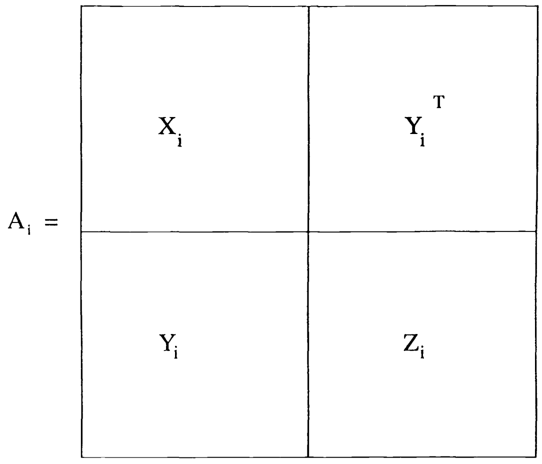


FIG. 1(c). *Recursive factorization of the matrix A_i.*

algebra computation in graphs; in all these papers the resulting algorithms are ultimately reduced to application of the algorithm of the present paper.

The rest of the paper is organized as follows: In §2 we briefly recall the known parallel algorithms for MULT and INVERT, and we review their complexity estimates. In §3 we recall some definitions and then state our estimates for the complexity of LINEAR-SOLVE. In §4 we present the parallel nested dissection algorithm for LINEAR-SOLVE for the case of sparse symmetric positive definite systems. In §5, we state our main theorem, which provides bounds on the complexity of the nested dissection algorithm. In §§6–8 we prove these bounds. In Remark 6.1 in §6 we comment on the extension of our results to the nonsymmetric sparse linear systems associated with directed graphs.

2. Auxiliary results on matrix multiplication and inversion. Our algorithm for LINEAR-SOLVE recursively reduces the original problem of large size to a sequence of problems of MULT and INVERT of smaller sizes. Let us recall the complexity of the solution of the two latter problems.

Let $M(n)$ denote an upper bound on the number of processors that suffice to multiply a pair of $n \times n$ matrices in $O(\log n)$ time. Here and hereafter the numbers of processors are defined within a constant factor (we assume Brent's (slowdown) scheduling principle of parallel computations, according to which we may decrease the number of processors from P to $\lceil P/s \rceil$ by using s times as many parallel steps for any natural $s \le P$). By the upper bound of [Ch], obtained by the straightforward parallelization of the algorithm of [Stra1], we may chose $M(n) \le n^{2.81}$. In [PR1] and [Pan10] we show that if $k \times k$ matrices can be multiplied in $O(k^\beta)$ arithmetic operations and $\beta < \gamma$ for some γ, then we may choose $M(n) \le n^\omega$ for some $\omega < \gamma$ and for all n. The current best upper bound on β and ω is 2.375 [CW1]; for surveys of the exciting history of the asymptotic acceleration of matrix multiplications, see also [Pan6], [Pan7], or the original works [Stra1] (the first and justly celebrated breakthrough in this area), [Pan1]–[Pan5], [BCLR], [Bi], [Schö], [CW2], [Stra2]. In practice, however, even for matrices of reasonably large sizes, we should only count on $M(n) = n^3/\log n$ or, at best, on $M(n) = O(n^{2.78})$ because of the considerable overhead of the known asymptotically faster algorithms for matrix multiplication (see [Pan7]).

Let us next estimate the complexity of INVERT.

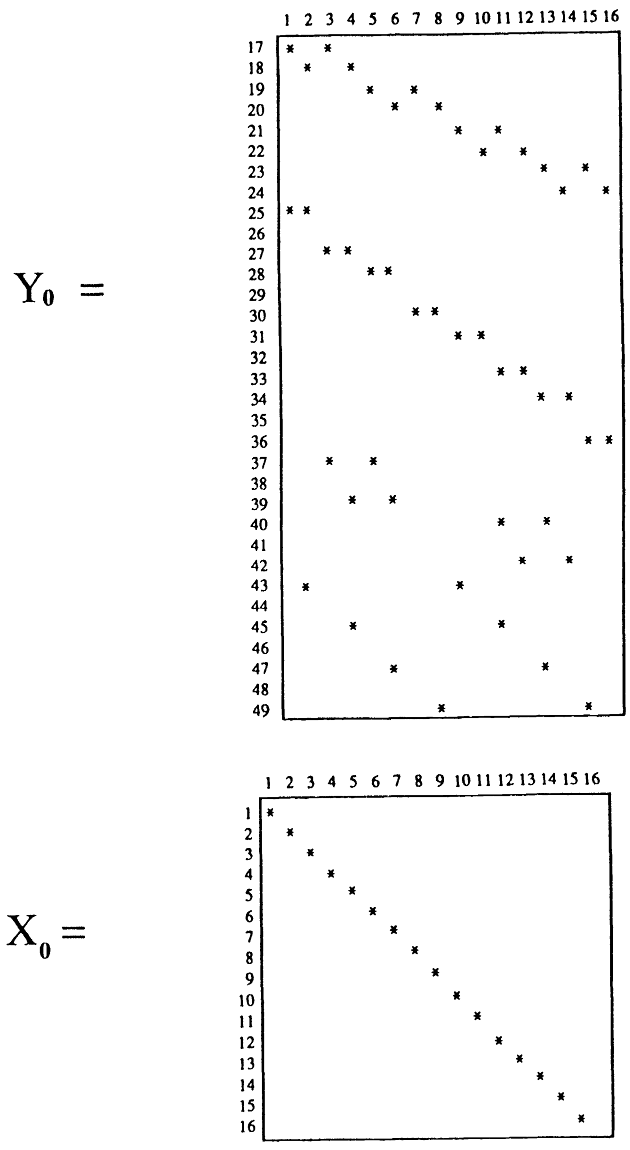


FIG. 1(d). *Submatrices Y_0 and X_0 of the matrix A_0.*

DEFINITION 2.1. We call an $n \times n$ matrix W well conditioned if $\log \operatorname{cond} W = O(\log n)$, where $\operatorname{cond} W = \|W\|\ \|W^{-1}\|$ for a fixed matrix norm (this definition is invariant in l for all l-norms of matrices).

Now we may recall the following estimate from [PR5] based on the algorithm of [Be] (compare [PaS]):

Fact 2.1. The problem INVERT for an $n \times n$ well-conditioned matrix A and for a positive $\varepsilon < 1$ such that $\log\log(1/\epsilon) = O(\log n)$ can be solved within error bound ε by using $O(\log^2 n)$ parallel time and $M(n)$ processors.

Furthermore, matrix multiplication can be reduced to matrix inversion (see [BM, p. 51] or [Pan7]), so that the processor bound, as well as the parallel and sequential time bounds

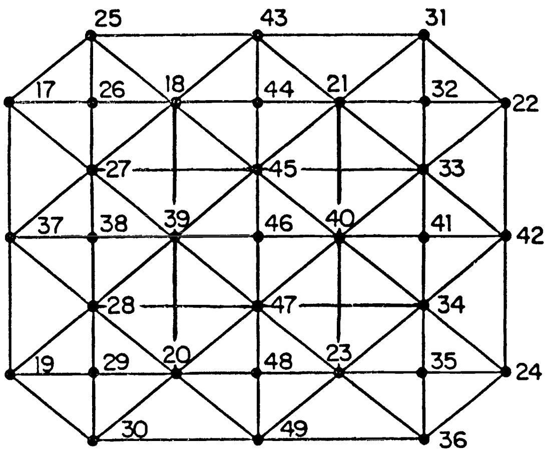


FIG. 2(a). *Graph G_1 derived from G_0 by simultaneous elimination of $R_0 = \{1, 2, \dots, 16\}$.*

$A_1 =$

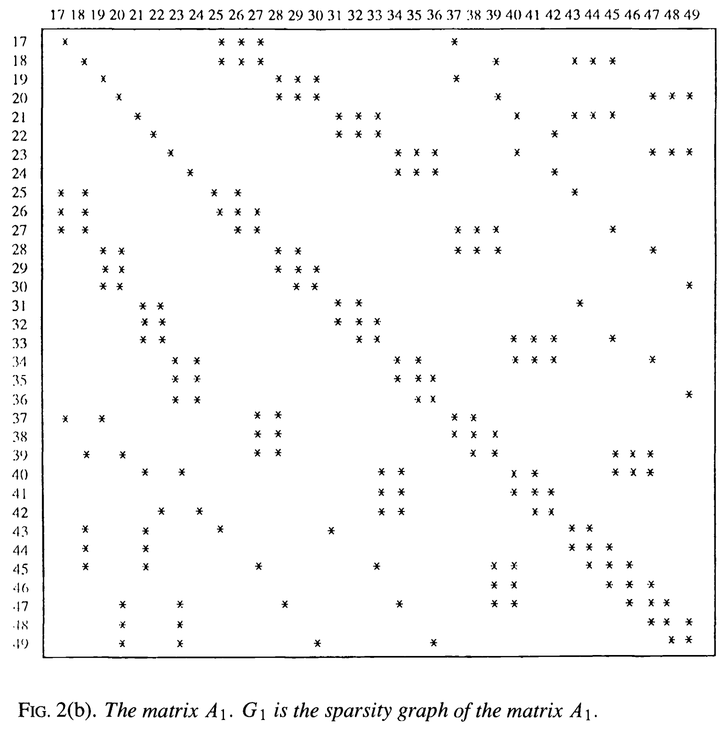


FIG. 2(b). *The matrix A_1. G_1 is the sparsity graph of the matrix A_1.*

attained this way, are optimal or nearly optimal (to within a factor of $O(\log n)$). In [Pan8] the above results for dense matrices are extended to the exact evaluation of the inverse of A, of the determinant of A, and of all the coefficients of the characteristic polynomial of A in $O(\log^2 n)$ steps by using $M(n)$ processors in the case for which A is an arbitrary matrix filled with integers and such that $\log \|A\| = n^{O(1)}$ (see proceedings papers [GP1, Part 1] and [Pan9], which cite and (partly) reproduce [Pan8], and see also its extensions in [Pan10], [Pan11], and [KS]).

In §3 we state our estimates for the complexity of sparse LINEAR-SOLVE by using the above estimates for the complexity of MULT and INVERT.

$$Y_1 = \begin{array}{c|cccccccc|}
 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
\hline
25 & * & * & & & & & & \\
26 & * & * & & & & & & \\
27 & * & * & & & & & & \\
28 & & & * & * & & & & \\
29 & & & * & * & & & & \\
30 & & & * & * & & & & \\
31 & & & & & * & * & & \\
32 & & & & & * & * & & \\
33 & & & & & * & * & & \\
34 & & & & & & & * & * \\
35 & & & & & & & * & * \\
36 & & & & & & & * & * \\
37 & * & & * & & & & & \\
38 & & & & & & & & \\
39 & & * & & * & & & & \\
40 & & & & & * & & * & \\
41 & & & & & & & & \\
42 & & & & & & * & & * \\
43 & & * & & & * & & & \\
44 & & * & & & * & & & \\
45 & & * & & & * & & & \\
46 & & & & & & & & \\
47 & & & & * & & & * & \\
48 & & & & * & & & * & \\
49 & & & & * & & & * & \\
\hline
\end{array}$$

$$X_1 = \begin{array}{c|cccccccc|}
 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\
\hline
17 & * & & & & & & & \\
18 & & * & & & & & & \\
19 & & & * & & & & & \\
20 & & & & * & & & & \\
21 & & & & & * & & & \\
22 & & & & & & * & & \\
23 & & & & & & & * & \\
24 & & & & & & & & * \\
\hline
\end{array}$$

FIG. 2(c). *Submatrices Y_1 and X_1 of the matrix A_1.*

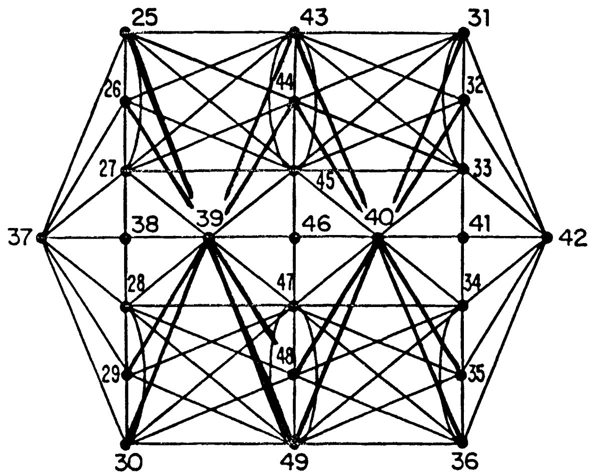


FIG. 3(a). *Graph G_2 derived from G_1 by simultaneous elimination of $R_1 = \{17, 18, \ldots, 24\}$.*

Let us point out two alternatives. In the current applications of our algorithm (see the end of §3) we apply Gaussian elimination for matrix inversion, which for an $n \times n$ matrix means $O(n)$ steps and n^2 processors. On the other hand, theoretically, we may rely on the exact evaluation of the inverse of an $n \times n$ matrix over rationals. This problem has interesting combinatorial applications (see [Lo], [GP1], [GP2], [MVV]). The known parallel algorithms

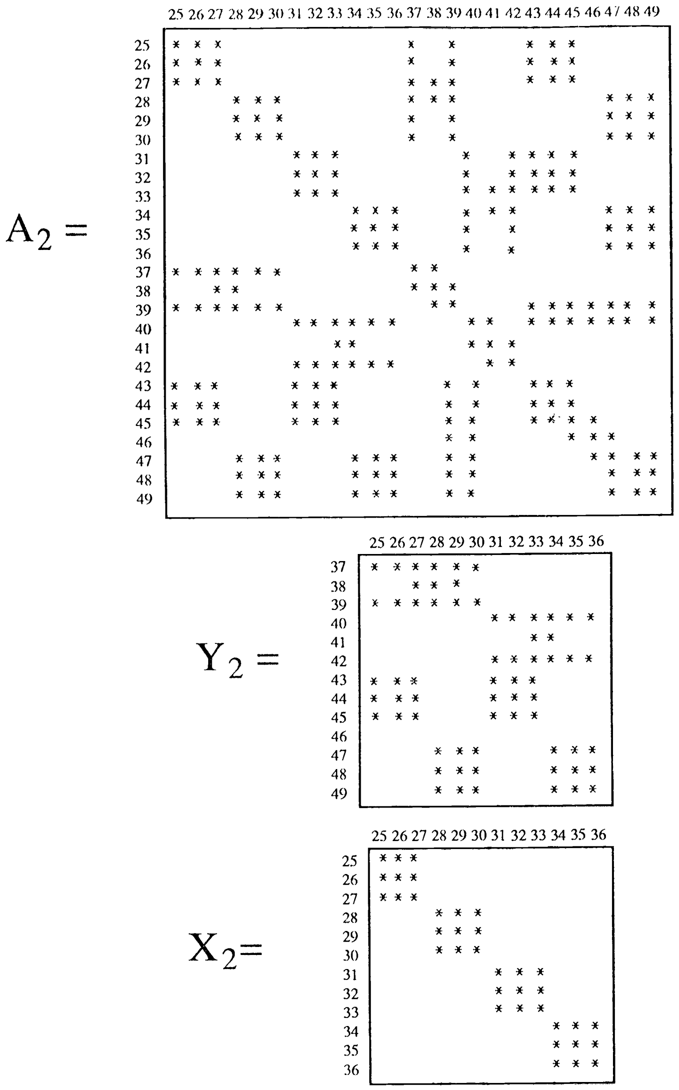


FIG. 3(b). *The matrix A_2 and its submatrices Y_2 and X_2. G_2 is the sparsity graph of the matrix A_2.*

for its solution use $O(\log^2 n)$ steps and $n^\alpha M(n)$ processors, where α varies from 1 in [Cs] to $\frac{1}{2}$ in [PrS] and to slightly less than $\frac{1}{2}$ in [GP3]; furthermore, $\alpha = 0$ even for INVERT over the real matrices if we allow randomized Las Vegas algorithms, because of combining [KS] and [Pan11] (see also [KP], [BP]), although the problem of numerical stability arises with all of these matrix inversion algorithms. The parallel cost of solving sparse linear systems varies, respectively, with the change of matrix inversion algorithms.

3. Some definitions and the complexity of sparse LINEAR-SOLVE. To characterize the linear systems $A\mathbf{x} = \mathbf{b}$ that our algorithm solves, we will need some definitions.

DEFINITION 3.1. A graph $G = (V, E)$ is said to have an *$s(n)$-separator family* (with respect to two constants, $\alpha < 1$ and n_0) if either $|V| \leq n_0$ or, by deleting some separator set S of vertices such that $|S| \leq s(|V|)$, we may partition G into two disconnected subgraphs with

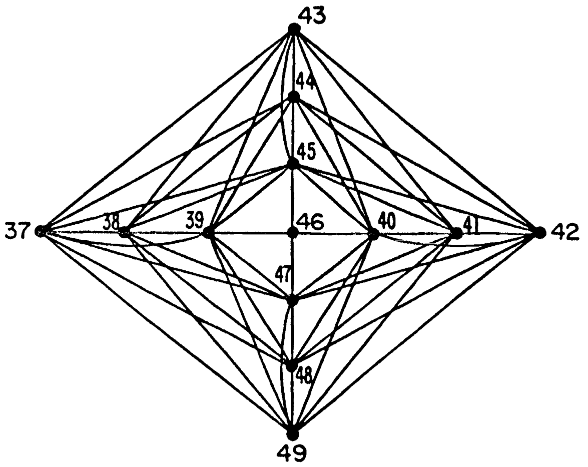


FIG. 4(a). *Graph G_3 derived from G_2 by simultaneous elimination of $R_2 = \{25, 26, \dots, 36\}$.*

$$A_3 = \begin{array}{c|ccccccccccccc|}
 & 37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 & 49 \\
\hline
37 & * & * & * & & & & * & * & * & & * & * & * \\
38 & * & * & * & & & & * & * & * & & * & * & * \\
39 & * & * & * & & & & * & * & * & * & * & * & * \\
40 & & & & * & * & * & * & * & * & * & * & * & * \\
41 & & & & * & * & * & * & * & * & & * & * & * \\
42 & & & & * & * & * & * & * & * & & * & * & * \\
43 & * & * & * & * & * & * & * & * & * & & & & \\
44 & * & * & * & * & * & * & * & * & * & & & & \\
45 & * & * & * & * & * & * & * & * & * & * & & & \\
46 & & & * & * & & & & & * & * & * & & \\
47 & * & * & * & * & * & * & & & & * & * & * & * \\
48 & * & * & * & * & * & * & & & & & * & * & * \\
49 & * & * & * & * & * & * & & & & & * & * & * \\
\hline
\end{array}$$

$$Y_3 = \begin{array}{c|cccccc|}
 & 37 & 38 & 39 & 40 & 41 & 42 \\
\hline
43 & * & * & * & * & * & * \\
44 & * & * & * & * & * & * \\
45 & * & * & * & * & * & * \\
46 & & & * & * & & \\
47 & * & * & * & * & * & * \\
48 & * & * & * & * & * & * \\
49 & * & * & * & * & * & * \\
\hline
\end{array}$$

$$X_3 = \begin{array}{c|cccccc|}
 & 37 & 38 & 39 & 40 & 41 & 42 \\
\hline
37 & * & * & * & & & \\
38 & * & * & * & & & \\
39 & * & * & * & & & \\
40 & & & & * & * & * \\
41 & & & & * & * & * \\
42 & & & & * & * & * \\
\hline
\end{array}$$

FIG. 4(b). *The matrix A_3 and its submatrices Y_3 and X_3. G_3 is the sparsity graph of the matrix A_3.*

the vertex sets V_1 and V_2 such that $|V_i| \leq \alpha|V|$, $i = 1, 2$, and if, furthermore, each of the two subgraphs of G defined by the vertex sets $S \cup V_i$, $i = 1, 2$, also has an $s(n)$-separator family (with respect to the same constants α and n_0). The resulting recursive decomposition of G is known as the *$s(n)$-separator tree*, so that each partition of the subgraph G defines its children in the tree. The vertices of the tree can thus be interpreted as subgraphs of G or as their vertex sets (we will assume the latter interpretation), and the edges of the tree can be interpreted as the

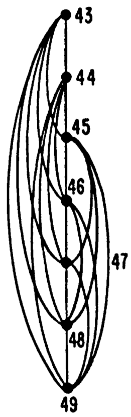


FIG. 5(a). *Graph G_4 derived from G_3 by simultaneous elimination of $R_3 = \{37, 38, \ldots, 42\}$.*

$$A_4 = \begin{array}{c|cccccc|} & 44 & 45 & 46 & 47 & 48 & 49 \\ \hline 44 & * & * & * & * & * & * \\ 45 & * & * & * & * & * & * \\ 46 & * & * & * & * & * & * \\ 47 & * & * & * & * & * & * \\ 48 & * & * & * & * & * & * \\ 49 & * & * & * & * & * & * \\ \hline \end{array}$$

FIG. 5(b). *The matrix A_4. G_4 is the sparsity graph of the matrix A_4.*

separator sets. Then the vertex set V equals the union of all the vertex subsets in V associated with the edges of the $s(n)$-separator tree and with its leaves. We call a graph *$s(n)$-separatable* if it has an $s(n)$-separator family and if its $s(n)$-separator tree is available.

The above definition of a separator tree follows [LT] and includes a separator in each induced subgraph, unlike the definition of [GT] (see Remark 1.1).

Binary trees obviously have a 1-separator family. A d-dimensional grid (of a uniform size in each dimension) has an $n^{1-(1/d)}$-separator family. [LRT] shows that the planar graphs have a $\sqrt{8n}$-separator family and that every n-vertex finite element graph with at most k boundary vertices in every element has a $4\lfloor k/2 \rfloor \sqrt{n}$-separator family. An improved construction due to [D] gives a $\sqrt{6n}$-separator family for planar graphs. (Similar small separator bounds have also been derived by Djidjev for bounded genus graphs and for several other classes of graphs.)

DEFINITION 3.2. Given an $n \times n$ symmetric matrix $A = (a_{ij})$, define $G(A) = (V, E)$ to be the undirected graph with the vertex set $V = \{1, \ldots, n\}$ and the edge set $E = \{\{i, j\} | a_{ij} \neq 0\}$. (We may say that A is *sparse* if $|E| = o(n^2)$.)

The very large linear systems $A\mathbf{x} = \mathbf{b}$ that arise in practice are often sparse and, furthermore, have graphs $G(A)$ with small separators. Important examples of such systems can be found in circuit analysis (e.g., in the analysis of the electrical properties of a VLSI circuit), in structural mechanics (e.g., in the stress analysis of large structures), and in fluid mechanics (e.g., in the design of airplane wings and in weather prediction). These problems require the solution of (nonlinear) partial differential equations, which are then closely approximated by very large linear differential equations whose graphs are planar graphs or three-dimensional grids. Certain weather prediction models, for example, consist of a three-dimensional grid of size $H_1 \times H_2 \times H_3$ with a very large number $n = H_1 H_2 H_3$ of grid points, but this grid has only a constant height H_3, and hence it has an $s(n)$-separator family for which $s(n) \leq \sqrt{H_3 n}$.

Our algorithm for LINEAR-SOLVE is effective for the systems whose associated graphs have $s(n)$-separator families for which $s(n) = o(n)$ and for which $s(n)$-separator trees are readily available. Thus our result can be viewed as a reduction of sparse LINEAR-SOLVE to the problems of (1) computing an $s(n)$-separator tree in parallel and (2) solving dense linear systems of $s(n)$ equations with $s(n)$ unknowns.

Efficient parallel computation of $s(n)$-separator trees is not simple in general, but it is rather straightforward in the practically important cases of grid graphs (see Figs. 1–5); similarly, such computation is simple for many finite element graphs (see [Ge]). The recent $O(\log^2 n)$-time, $n^{1+\varepsilon}$ processor (for any $\varepsilon > 0$) randomized parallel algorithm of [GM1] gives $O(\sqrt{n})$-separator trees for all the planar graphs.

Many very large sparse linear systems of algebraic equations found in practice, such as linear systems arising in the solution of two-dimensional linear partial differential equations with variable coefficients, have associated graphs that are not grid graphs but that have $s(n)$-separators for $s(n) = o(n)$. The correctness of the generalized parallel nested dissection algorithm applied in the case of graphs with $s(n)$-separators (and thus already in the important case of planar graphs) requires a considerably more complex substantiation and a more advanced proof technique than does the case of grid graphs; in particular, a sophisticated inductive argument is required (see §§7 and 8). This occurs because grid graphs are more regular than are general graphs with $s(n)$-separators.

Let us state the complexity estimates for our solution of sparse LINEAR-SOLVE. Our main result is the decrease of the previous processor bound (supporting the poly-log parallel time) from $M(n)$ to $(|E| + M(s))/\log n$, whereas the time bound increases from $O(\log^2 n)$ to $O(\log^3 n)$. (Since in all the known applications of interest $s(n)$ exceeds cn^δ for some positive constants c and δ, we will write $O(\log n)$ rather than $O(\log s(n))$ to simplify the notation. Also note that $2|E|$ is roughly the number of nonzero input entries, so that we cannot generally count on decreasing the sequential time (and therefore also the total work, that is, parallel time times the processor bound) below $|E|$.) It follows, for example, that our improvement of the previous processor bound is by a factor of n if $s(n) = n^{1/2}$, $|E| = O(n^{3/2})$. Because practical implementations of the algorithms would be slowed down to satisfy processor limitations of the actual computers (see discussion at the end of this section), we will decrease the processor bound of $M(n)\log^2 n/T(n)$ to $(|E| + M(s(n)))\log^2 n/T(n)$ in our algorithm, provided that it runs in $T(n)$ time, where $T(n) > c\log^3 n$, $c = O(1)$.

Let us comment further on how we arrive at our estimates. In general, the inverse A^{-1} of a sparse matrix A (even of one with small separators) is dense, and, in fact, if $G(A)$ is connected, A^{-1} may have no zero entries. Therefore, it is common to avoid computing the inverse matrix and instead to factorize it. Our algorithm for LINEAR-SOLVE follows this custom: It computes a special recursive factorization of A. For sparse matrices with small separators, our poly-log-time algorithm yields processor bounds that are of an order of magnitude lower than the bounds attained by means of other poly-log-time parallel algorithms, which compute the inverse matrix. Specifically, let an $n \times n$ positive definite symmetric well-conditioned matrix A be given, such that $G(A)$ has an $s(n)$-separator family, its $s(n)$-separator tree is known, and $s(n)$ is of the form αn^σ for two constants $\sigma < 1$ and α. Then we first compute a special recursive factorization of A (within the error bound 2^{-n^c} for a positive constant c) in $O(\log^3 n)$ time by using $M(s(n))/\log n$ processors (see Theorem 5.1 below), and finally we compute the desired solution vector $\mathbf{x} = A^{-1}\mathbf{b}$. The complexity of this final stage is lower than the complexity of computing the recursive factorization.

For comparison the inversion of an $s(n) \times s(n)$ dense matrix is one of the steps of computing the recursive factorization of A, and the current parallel cost of this step alone is at best $O(\log^2 n)$ time and $M(s(n))$ processors (by using the parallel algorithm of [Be], [PR1].) When our special recursive factorization has been computed, the solution of $A\mathbf{x} = \mathbf{b}$ (for any given $\mathbf{b}$) requires only $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors. It is interesting that by multiplying our parallel time and processor bounds we arrive at the sequential complexity estimate of $O(M(s(n))\log^2 n) = o(s(n)^{2.4})$ arithmetic operations, which matches the theoretical upper bound of [LRT].

Let us demonstrate some consequences of the complexity bounds of our algorithm. We will first assume the weak bound $M(n) = n^3/\log n$ for matrix multiplication. It is significant that already under this assumption our parallel nested dissection algorithm, for poly-log time bounds, has processor bounds that substantially improve the previously known bounds. Let G be a fixed planar graph with n vertices given with its $O(\sqrt{n})$-separator tree. (For example, G might be a graph with a $\sqrt{n} \times \sqrt{n}$ grid.) Then, for any $n \times n$ matrix A such that $G = G(A)$, our parallel nested dissection algorithm takes $O(\log^3 n)$ time and $n^{1.5}/\log^2 n$ processors to compute the special recursive factorization of A and then $O(\log^2 n)$ time and n processors to solve any linear system $A\mathbf{x} = \mathbf{b}$ with A fixed. We have the time bounds $O(\log^3(kn))$ and $O(\log^2(kn))$ and the processor bounds $k^3n^{1.5}/\log^2(kn)$ and kn, respectively, if $G(A)$ is an n-vertex finite element graph with at most k vertices on the boundary of each face. In yet another example, $G(A)$ is a three-dimensional grid, so that it has an $n^{2/3}$-separator family. In this case we have the same asymptotic time bounds as for planar graphs and our processor bounds are $n^2/\log^2 n$ and $n^{1.33}$, respectively. Furthermore, if we use the theoretical bounds for matrix multiplication, say $M(n) = n^{2.4}$, then our processor bounds for computing the special recursive factorization are further decreased to $n^{1.2}$ in the planar case, to $k^{2.4}n^{1.2}$ in the case of the n-vertex finite elements graphs with at most k vertices per face, and to $n^{1.6}$ for the three-dimensional grid.

In the current practical implementations of our algorithm ([LMNOR] and [OR]) we have not reached the poly-log-time bounds because we have simply used the Gaussian elimination rather than Ben-Israel's algorithm at the stages of matrix inversions, and so we achieve time $s(n)$ with $s(n)^2$ processors. The reason for this choice is the limitation on the number of processors that we could efficiently use on the available computers. It is certain, however, that the future parallel computers will have significantly more processors, and then the application of Ben-Israel's algorithm may be preferred; we cannot exactly estimate the threshold number of processors that would in practice give the edge to Ben-Israel's algorithm over Gaussian elimination, but according to our theoretical estimates the former algorithm improves the parallel time bounds of the latter one if more than $s(n)^2$ processors are available.

4. Outline of the parallel generalized nested dissection algorithm. In this section we fix an undirected graph G having an $s(n)$-separator family (with respect to constants n_0 and α) (see Definition 3.1). Let A be an $n \times n$ real symmetric positive definite matrix with graph $G = G(A)$ (see Definition 3.2). We will describe an efficient parallel algorithm that computes a special recursive factorization of A. With such a factorization available it will become very simple to solve the system of linear equations $A\mathbf{x} = \mathbf{b}$ for any given vector $\mathbf{b}$ (see the last part of Theorem 5.1 below).

DEFINITION 4.1. A *recursive $s(n)$-factorization* of a symmetric matrix A (associated with a graph $G = G(a)$ having an $s(n)$-separator family with respect to two constants α, $\alpha < 1$, and n_0) is a sequence of matrices, $A_0, A_1, \dots, A_d$, such that $A_0 = PAP^T$, where P is an $n \times n$ permutation matrix, A_h has size $n_{d-h} \times n_{d-h}$,

$$(4.1)\quad A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix}, \qquad Z_h = A_{h+1} + Y_h X_h^{-1} Y_h^T, \qquad h = 0, 1, \dots, d-1,$$

and X_h is a symmetric block diagonal matrix corresponding to the separators or, for $h = d$, to the vertex sets in V associated with the leaves of the $s(n)$-separator tree and consisting of square blocks of sizes at most $s(n_{d-h}) \times s(n_{d-h})$, where

$$n_d = n, \qquad n_{h-1} \le \alpha n_h + s(n_h), \qquad h = 1, \ldots, d. \tag{4.2}$$

Here and hereafter, W^T denotes the transpose of a matrix or vector W. (Note that the constant n_0 used in Definition 3.1 for the separator family is also the order of the diagonal blocks of the matrix X_d.) This recursive factorization is said to be *numerically computed* if the computerized approximants of the induced matrices $A_1, \ldots, A_d$ satisfy (4.1) within the error norm 2^{-n^c} (relative to $\|A\|$) for a positive constant c.

Our definition of a recursive $s(n)$-factorization relies on the matrix identities

$$A_h = \begin{bmatrix} I & O \\ Y_h X_h^{-1} & I \end{bmatrix} \begin{bmatrix} X_h & O \\ O & A_{h+1} \end{bmatrix} \begin{bmatrix} I & X_h^{-1} Y_h^T \\ O & I \end{bmatrix} \tag{4.3}$$

and

$$A_h^{-1} = \begin{bmatrix} I & -X_h^{-1} Y_h^T \\ O & I \end{bmatrix} \begin{bmatrix} X_h^{-1} & O \\ O & A_{h+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_h X_h^{-1} & I \end{bmatrix}. \tag{4.4}$$

Here the matrix A_{h+1} defined by (4.1) is known in linear algebra as Schur's complement of X_h, h ranges from 0 to $d-1$, I denotes the identity matrices, and O denotes the null matrices of appropriate sizes. We show in §7 that A_{h+1} also has certain sparsity properties.

The recursive decomposition (4.1)–(4.4) is intimately related to the recursive decomposition of the associated graph $G = G(A)$ defined by (and defining) the $s(n)$-separator tree. Specifically, we will see that the matrices X_h are block diagonal matrices whose blocks for $h = 0, 1, \ldots, d-1$ are associated with the separator sets of level h from the root of the tree, whereas the blocks of X_d are associated with the leaves of the tree.

Given a symmetric $n \times n$ matrix A associated with an $s(n)$-separable graph $G(A)$, we may compute the recursive $s(n)$-factorization (4.1)–(4.4) by performing the following stages:

Stage 0. Compute an appropriate permutation matrix P, matrix $A_0 = PAP^T$, and the decreasing sequence of positive integers $n = n_d, n_{d-1}, \ldots, n_0$ satisfying (4.2) and defined by the sizes of the separators in the $s(n)$-separator family of $G = G(A)$ (as specified below in §7). The permutation matrix P and the integers $n_d, n_{d-1}, \ldots, n_0$ completely define the order of the elimination of the variables (vertices), so that first we eliminate the vertices of G corresponding to the leaves of the $s(n)$-separator tree, then we eliminate the vertices of G corresponding to the edges adjacent to the leaves of the tree (that is, the vertices of the separators used at the final partition step), then we eliminate the vertices of G corresponding to the next edge level of the tree (separators of the previous partition step), and so on; we formally analyze this in §§7 and 8.

Stage $h+1$ ($h = 0, \ldots, d-1$). Compute the matrices X_h^{-1}, $-Y_h X_h^{-1}$ (which also gives us the matrix $-X_h^{-1} Y_h^T = (-Y_h X_h^{-1})^T$) and $A_{h+1} = Z_h - Y_h X_h^{-1} Y_h^T$ satisfying (4.1), (4.3), (4.4) and such that A_{h+1} has size $n_{d-h-1} \times n_{d-h-1}$. (Each of these stages amounts to inversion, two multiplications, and subtraction of some matrices.)

When the recursive factorization (4.4) has been computed, it will remain to compute the vector $\mathbf{x} = A^{-1}\mathbf{b} = P^T A_0^{-1}(P\mathbf{b})$ for a column vector $\mathbf{b}$. This can be done by means of recursive premultiplications of some subvectors of the vector $P\mathbf{b}$ by the matrices

$$\begin{bmatrix} I & O \\ -Y_h X_h^{-1} & I \end{bmatrix}, \quad \begin{bmatrix} X_h^{-1} & O \\ O & A_{h+1}^{-1} \end{bmatrix}, \quad \begin{bmatrix} I & -X_h^{-1} Y_h^T \\ O & I \end{bmatrix}$$

for $h = 0, 1, \ldots, d$. At the stage of the premultiplication by the second matrix above, the premultiplication by X_h^{-1} is done explicitly and the premultiplication by A_{h+1}^{-1} is performed by means of recursive application of (4.4), so that (4.4) defines a simple recursive algorithm for computing $A^{-1}\mathbf{b}$ for any column vector $\mathbf{b}$ of length n, provided that a recursive $s(n)$-factorization (4.1) is given.

It is instructive to compare the recursive $s(n)$-factorization (4.1)–(4.4) with the Cholesky factorization of A_h used in [LRT]. The notations in [LRT] are distinct from ours, but for the sake of making the comparison we assume that the notation is adjusted to the same format. Then we may say that both factorizations rely on the matrix identities (4.3), (4.4) which, in fact, just represent the block Jordan elimination algorithm for a 2×2 block matrix A_h of (4.1). The Cholesky factorization $PAP^T = LDL^T$ is obtained in [LRT] by the application of the Jordan elimination to the matrix PAP^T, which is equivalent to the recursive application of (4.3) to *both* submatrices X_h and A_{h+1}. (This defines L and L^T in factorized form, but the entries of the factors do not interfere with each other, so that all the entries of $Y_h X_h^{-1}$ coincide with the respective entries of L.) Efficient parallelization of this recursive algorithm (yielding $O(\log^3 n)$ parallel time) is straightforward, except for the stage of the factorization of the matrices X_h (which, by Lemma 7.2 below, are block diagonal with dense diagonal blocks of sizes of the order of $s(n_h) \times s(n_h)$). However, for the purpose of solving the systems $A\mathbf{x} = \mathbf{b}$, we do not have to factorize the matrices X_h. It suffices to invert them, and this can be efficiently done by using the techniques of [Be], provided that A is a well-conditioned matrix. Thus we arrive at the recursive $s(n)$-factorization (4.1)–(4.4), where we recursively factorize only the matrices A_{h+1} in (4.3) and A_{h+1}^{-1} in (4.4) but not the matrices X_h and X_h^{-1}. This modification of the factorization scheme is crucial in some important combinatorial computations (see [PR4], [PR6]).

5. Parallel generalized nested dissection: The main theorem. Hereafter, we will assume that c and σ are constants such that

$$s(n) = cn^{\sigma}, \qquad \tfrac{1}{2} \le \sigma < 1. \tag{5.1}$$

Equation (5.1) holds in all the interesting applications (such as planar graphs, grid graphs, and finite element graphs) for which $s(n)$-separator families are defined.

For simplicity, we will also assume hereafter that

$$M(n) = n^{\omega^*} \quad \text{for some constant } \omega^* > 2 \ge \tfrac{1}{\sigma} \tag{5.2}$$

and consequently that

$$M(ab) = M(a)M(b). \tag{5.3}$$

THEOREM 5.1. *Let $G = (V, E)$ be an $s(n)$-separatable graph for $s(n)$ satisfying* (5.1). *Then, given an $n \times n$ symmetric positive definite matrix A such that* cond $A = n^{O(1)}$ *and $G = G(A)$, we can numerically compute a recursive $s(n)$-factorization of A in time $O(\log^3 n)$ with $M(s(n))/\log n$ processors* (*provided that $M(s(n))$ processors suffice to multiply a pair of $s(n) \times s(n)$ matrices in time $O(\log n)$ and that $M(n)$ satisfies* (5.2)). *Whenever such a recursive $s(n)$-factorization of A is available, $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors suffice to solve a system of linear equations $A\mathbf{x} = \mathbf{b}$ for any given vector $\mathbf{b}$ of dimension n.*

Remark 5.1. It is possible to extend Theorem 5.1 to the case for which (5.1) does not hold by using [LRT, Thms. 7–9]. On the other hand, the restriction to the class of symmetric positive definite input matrices A in the statements of Theorem 5.1 is needed only to support numerical stability of the factorization (4.3), (4.4).

Remark 5.2. The product of our parallel time and processor bounds is the same, TP = (PARALLEL TIME) $*$ PROCESSOR = $O(M(s(n)) \log^2 n)$, both for computing the whole recursive factorization (4.3), (4.4) and for its proper stage of inverting X_{d-1}.

Remark 5.3. As was noted by Gazit and Miller [GM2], the recursive $s(n)$-factorization can be computed by using $O(\log^2 n \log\log n)$ time and $M(s(n))$ processors. Moreover, their approach can be extended to reach the bounds of $O(\log^2 n \log(1/\epsilon))$ time and simultaneously $O(M(s(n))^{1+\epsilon})$ processors for a positive parameter $\in$ [AR]. One may try to improve the processor efficiency of the latter estimates by applying a relatively minor super effective slowdown of the computations [PP].

6. Outline of the proof of the main theorem. We will show that the parallel algorithm uses only $d = O(\log n)$ stages. Let $\delta = \delta(n)$ denote $cn^{\sigma-1} = s(n)/n$, and let n_0 be large enough, so that

$$(6.1) \qquad \alpha k + s(k) = (\alpha+\delta)k = \beta k, \qquad \beta = \alpha + \delta < 1 \quad \text{if } k > n_0.$$

Equations (4.2) and (6.1) together imply that

$$(6.2) \qquad n_h \le (\alpha+\delta)^{d-h} n = \beta^{d-h} n, \qquad h = 0, 1, \dots, d.$$

LEMMA 6.1. *$d = O(\log n)$ for fixed n_0 and $\alpha < 1$.*

Proof. Relation (6.2) for $h = 0$ implies that $d \le \log(n/n_0)/\log(1/(\alpha+\delta)) = O(\log n)$. $\square$

The next lemma shows that for all h the auxiliary matrices A_h and X_h are positive definite (and therefore nonsingular) and have condition numbers not exceeding the condition number of A. This implies that our parallel algorithm is numerically stable. The lemma follows from the observation that A_{h+1}^{-1} is a principal submatrix of A_h^{-1} and from the interlacing property of the eigenvalues of a symmetric matrix (see [GoL] or [Par]).

LEMMA 6.2. *The matrices A_{h+1} and X_h are symmetric positive definite if the matrix A_h is symmetric positive definite, and, furthermore,* $\max\{(\text{cond } A_h)_2, (\text{cond } X_h)_2\} \le (\text{cond } A)_2$ *for all h.*

In the remainder of this paper we further specify our parallel nested dissection algorithm and estimate its complexity. We observe that all its arithmetic operations (except those needed in order to invert X_h for all h) are also involved in the sequential algorithm of [LRT]. (As in the latter paper, we ignore the arithmetic operations for which at least one operand is zero; we assume that no random cancellations of nonzero entries takes place, for if there are such cancellations, we would only arrive at more optimistic bounds; we will treat both X_h and X_h^{-1} as block diagonal matrices having nonzero blocks in the same places.)

For each h we group all the arithmetic operations involved to reduce these operations to a pair of matrix multiplications $U_h = Y_h X_h^{-1}$ (which also gives $X_h^{-1} Y_h^T = U_h^T$) and $W_h = U_h Y_h^T$ and to a (low-cost) matrix subtraction $A_{h+1} = Z_h - W_h$ (it is assumed here that the matrix X_h^{-1} has been precomputed). Below, Theorem 7.1 provides a bound on the complexity of numerically computing the inverse of the auxiliary matrices $X_0, \dots, X_d$, and Theorem 8.1 provides a bound on the cost of parallel multiplication of the auxiliary matrices and implies the time bound of $O(\log^2 n)$ for the entire computation, excluding the stage of the inversion of the matrices X_h. The number of processors is bounded above by the number of arithmetic operations used in the algorithm of [LRT], that is, by $O(s(n)^3)$ (see [LRT, Thm. 3] and Remark 5.2).

The estimates of Theorem 5.1 for the cost of computing the recursive factorization (4.1)–(4.4) immediately follow from Theorems 7.1 and 8.1 below.

Next we discuss the parallel complexity of back solving the linear system $A\mathbf{x} = \mathbf{b}$, given the recursive factorization. As we have already pointed out, when the recursive factorization (4.1)–(4.4) has been computed, we evaluate $\mathbf{x} = A^{-1}\mathbf{b}$ by means of successive premultiplications of some subvectors of $\mathbf{b}$ by the matrices $Y_h X_h^{-1}$, X_h^{-1}, and $X_h^{-1} Y_h^T$ for h ranging between

0 and d. The parallel time bounds are $O(\log n)$ for each h and $O(\log^2 n)$ for all h. The obvious processor bound is $(|E| + |F|)/\log n$, where $|E| + |F|$ denotes the number of entries of an $n \times n$ array corresponding to the nonzeros of at least one of the submatrices $X_h^{-1}Y_h^T$, $Y_h X_h^{-1}$, and X_h^{-1} of (4.4) for $h = 0, 1, \dots, d-1$ (for each h, $X_h^{-1}Y_h^T$, $Y_h X_h^{-1}$, and X_h^{-1} occupy the upper-right, lower-left, and upper-left corners of the array, respectively). The nonzeros of A_0 form the set E of the edges of $G(A)$; other nonzeros form the set F called *fill-in* (associated with the nonzeros introduced in the process of computing the $s(n)$-factorization (4.1)–(4.4)).

Finally, we must discuss the space bounds of the algorithm. By [LRT, Thm. 2], $|F| = O(n+s(n)^2 \log n)$, and this bound can be applied to our algorithm as well. The proofs in [LRT] are under the assumption that $s(n) = O(\sqrt{n})$, but the extension to any $s(n)$ satisfying (5.1) is immediate. Likewise, Lipton, Rose, and Tarjan estimated only the number of multiplications involved, but including the additions and subtractions would increase the upper estimates yielded in their and our algorithms by only a constant factor.

Remark 6.1. The algorithms and the complexity estimates of this paper will be immediately extended to the case of nonsymmetric linear systems with directed graphs if for all h we replace the matrices Y_h^T by matrices W_h (which are not generally the transposes of Y_h) and remove the assumption that the matrices X_h are symmetric. Of all the results and proofs, only Lemma 6.2 and the numerical stability of our $s(n)$-recursive factorization (4.1)–(4.4) are not extended. This lack of extension of Lemma 6.2 surely devalues the resulting numerical algorithm, but the algorithm remains powerful for the computations over the semirings (dioids), with interesting combinatorial applications (see [PR4], [PR6], [PR7]).

7. Cost of parallel inversion of the auxiliary matrices X_h. In this section we specify Stage 0 of computing the recursive factorization (4.1)–(4.4) (see §4) and prove the following result:

THEOREM 7.1. *Let A be an $n \times n$ well-conditioned symmetric positive definite matrix having a recursive $s(n)$-factorization. Then $O(\log^3 n)$ parallel time and $M(s(n))/\log n$ processors suffice to numerically invert the auxiliary matrices $X_0, \dots, X_d$ that appear in the recursive $s(n)$-factorization* (4.1)–(4.4).

Proof. We first reexamine the well-known correlations between the elimination of the variables and of the associated vertices of $G = G(A)$, which we will derive from the previous analysis of nested dissection in [R] and [GeL]. We observe that the elimination of a vertex (variable) v is associated with the replacement of the edges in the graph G as follows: (1) First, for *every* pair of edges $\{u_1, v\}$ and $\{v, u_2\}$, the fill-in edge $\{u_1, u_2\}$ is to be added to the set of edges (unless $\{u_1, u_2\}$ is already in the graph); (2) then every edge with an end point v is deleted.

Adding an edge such as $\{u_1, u_2\}$ to the edge set corresponds to four arithmetic operations of the form $z - y_1 x^{-1} y_2$, where x, y_1, y_2, z represent the edges $\{v, v\}$, $\{u_1, v\}$, $\{v, u_2\}$, $\{u_1, u_2\}$, respectively (see Figs. 1–5 and the end of Remark 1.1). If a block of variables is eliminated, then a set S, representing this block, should replace a vertex in the above description, so that, at first, for every pair of edges $\{u_1, s_1\}$, $\{u_2, s_2\}$ with the end points s_1 and s_2 in S, the edge $\{u_1, u_2\}$ is added to the set of edges, and then, when all such pairs of edges have been scanned, all the edges with one or two end points in S are deleted. This corresponds to the matrix operations of the form $Z - Y_1 X^{-1} Y_2^T$, where X, Y_1, Y_2^T, Z represent the blocks of edges of the form $\{s_1, s_2\}$, $\{u_1, s_1\}$, $\{s_2, u_2\}$, $\{u_1, u_2\}$, respectively, where $s_1, s_2 \in S$ and where u_1, u_2 denote two vertices connected by edges with S. For symmetric matrices we may assume that $Y_1 = Y_2 = Y$. Of course, the objective is to arrange the elimination so as to decrease the fill-in and the (sequential and parallel) arithmetic cost. This objective is achieved in the nested dissection algorithm, in which the elimination is ordered so that every eliminated block of vertices is connected by edges with only relatively few vertices (confined to an $s(n_h)$-separator, separating the vertices of the eliminated block from all other vertices).

To ensure the latter property we exploit the existence of an $s(n)$-separator family for the graph $G = G(A)$ and order the elimination by using a separator tree T_G defined for a graph G as follows:

DEFINITION 7.1. See Fig. 6. Suppose that the graph $G = (V, E)$ has n vertices. If $n \leq n_0$ (see Definition 3.1), let T_G be the trivial tree with no edges and with the single leaf (V, S), where $S = V$. If $n > n_0$, we know an $s(n)$-separator S of G, so that we can find a partition $V(1)$, $V(2)$, S of V such that there exists no edge in E between the sets $V(1)$ and $V(2)$ and, furthermore, $|V(1)| \leq \alpha n$, $|V(2)| \leq \alpha n$, and $|S| \leq s(n)$. Then T_G is defined to be the binary tree with the root (V, S) having exactly two children that are the roots of the subtrees T_{G_1}, T_{G_2} *of* T_G, where G_j is the subgraph of G induced by the vertex set $S \cup V(j)$ for $j = 1, 2$. (Note that T_G is *not* equivalent to the elimination trees of [Schr], [Li2], and [GHLN] or to the separator trees of [GT], since the latter trees do not include the separator in the induced subgraphs.)

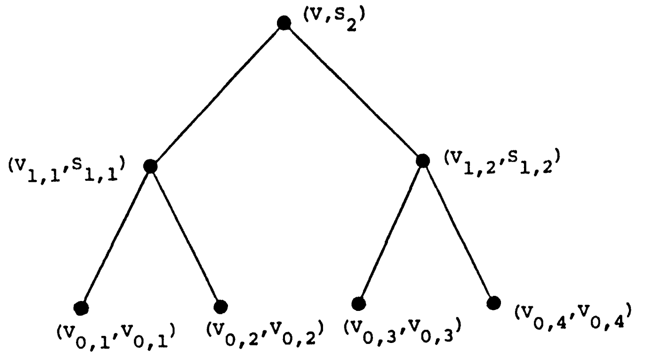


FIG. 6. *Tree* T_G *for the case* $d = 2$.

The following definitions are equivalent to the usual ones, as, for example, given in [LRT].

DEFINITION 7.2. Let the *height* of a node v in T_G equal d minus the length of the path from the root to v, where d, the height of the root, is the maximum length of a path from the root to a leaf. Let N_h be the number of nodes of height h in T_G. Since T_G is a binary tree, $N_h \leq 2^{d-h}$. Let $(V_{h,1}, S_{h,1}), \ldots, (V_{h,N_h}, S_{h,N_h})$ be a list of all the nodes of height h in T_G, and let $S_h = \bigcup_{k=1}^{N_h} S_{h,k}$.

Let n be the number of vertices of G. Since G has an $s(n)$-separator family, $|V_{h,k}| \leq n_h$ and $|S_{h,k}| \leq s(n_h)$ for each $h \geq 1$ and for $k = 1, \ldots, N_h$ (see (4.2) for the definition of n_h); furthermore, $|V_{0,k}| \leq n_0$ and $S_{0,k} = V_{0,k}$ for $k = 1, \ldots, N_0$ by the definition of the tree T_G.

DEFINITION 7.3. For each $k = 1, \ldots, N_h$ let $R_{h,k}$ denote the set of all the elements of $S_{h,k}$ that are not in S_{h^*} for $h^* > h$, so that $R_{h,k} = S_{h,k} - \cup S_{h^*,k^*}$, where the union is over all the ancestors $(V_{h^*,k^*}, S_{h^*,k^*})$ of $(V_{h,k}, S_{h,k})$ in T_G. Let $R_h = \bigcup_{k=1}^{N_h} R_{h,k}$.

Observe that, by the definition of the sets $R_{h,k}$ and R_h and of an $s(n)$-separator family, $R_{h,k_1} \cap R_{h,k_2} = \emptyset$ if $k_1 \neq k_2$, $R_h \cap R_{h^*} = \emptyset$ if $h \neq h^*$, and $V = \bigcup_{h=0}^{d} R_h$. Also observe that for distinct k the subsets $R_{h,k}$ of R_h are not connected by edges with each other; moreover, the vertices of each set $R_{h,k}$ can be connected by edges only with the vertices of the set $R_{h,k}$ itself and of the separator sets $S_{h+g,q}$ in the ancestor nodes of $(V_{h,k}, S_{h,k})$ of the tree T_G. Now we are ready to describe the order of elimination of vertices and of the associated variables. We will eliminate the vertices in the following order: first the vertices of R_0, then the vertices of R_1, then the vertices of R_2, and so on. (For each h we will eliminate the vertices of R_h in parallel for all the disjoint subsets $R_{h,1}, R_{h,2}, \ldots, R_{h,N_h}$.) This way all the vertices of $V = \bigcup_h R_h$ will be processed. In particular, the rows and columns of A_h associated with the vertices of $R_{h,k}$ form an $|R_{h,k}| \times |R_{h,k}|$ diagonal block of X_h for $k = 1, 2, \ldots, N_h$; X_h is the block diagonal submatrix of A_h with these N_h diagonal blocks, and $n_{h+1} = n_h - |R_h|$.

Let us now formally define the desired permutation matrix P and set of integers $n_d, n_{d-1}, \ldots, n_0$, which we need in Stage 0 (see §4). Let $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be any enumeration of the n vertices of G such that $\pi(v) < \pi(v^*)$ if $v \in R_h$, $v^* \in R_{h^*}$, $h^* > h$, and, furthermore, π consecutively orders the vertices of $R_{h,k}$ for each h and k. Thus the elements of $\pi(R_h)$ are in the range $\delta_h + 1, \ldots, \delta_{h+1}$, where $\delta_h = \sum_{g<h} |R_g|$. Such an enumeration can be easily computed directly from the separator tree in parallel time $O(\log^2 n)$ with $n/\log n$ processors by first numbering the vertices of $R_d = S_d$ as $n, n-1, \ldots$ and then numbering (also in the decreasing order) all the previously unnumbered vertices of R_h of height h for each h, where $h = d-1, d-2, \ldots, 0$.

We define the permutation matrix $P = [p_{ij}]$ such that $p_{ij} = 1$ if $j = \pi(i)$ and $p_{ij} = 0$ otherwise. This gives us the initial matrix $A_0 = PAP^T$. Recursively, for $h = 0, 1, \ldots, d-1$, let $n_h = n - \delta_h$, and this completes Stage 0.

Now we define

$$A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix},$$

the $(n - \delta_h) \times (n - \delta_h)$ symmetric matrix, where X_h is the $|R_h| \times |R_h|$ upper-left submatrix of A_h, Y_h is the $(n - \delta_h - |R_h| \times |R_h|)$ lower-left submatrix of A_h, and Z_h is the $(n - \delta_h - |R_h|) \times (n - \delta_h - |R_h|)$ lower-right submatrix of A_h. We then define $A_{h+1} = Z_h - Y_h X_h^{-1} Y_h^T$. Thus in Stage $h+1$ of computing the recursive factorization (see §4) we have eliminated the variables (vertices) associated with R_h.

We now claim that for a fixed h we can compute A_{h+1} from X_h, Z_h, and Y_h in time $O(\log^2 s(n))$ with at most $M(s(n))$ processors. To prove this we will investigate the sparsity of A_h and the connectivity of G_h.

Let $A_h = (a_{ij}^{(h)})$. We define the associated graph $G_h = (V_h, E_h)$ with the vertex set $V_h = \{\delta_h + 1, \delta_h + 2, \ldots, n\}$ and the edge set $E_h = \{\{i + \delta_h, j + \delta_h\} | a_{ij}^{(h)} \neq 0\}$; that is, G_h is derived from $G(A_h)$ by adding δ_h to each vertex number (see Figs. 1–5). Note that $i, j \in V_h$ if the edge $\{i, j\}$ belongs to E_h. (The fill-in in stage h is the set of edges that are in E_h but not in E_{h-1}.)

Now we are ready to establish a lemma that provides some useful information about the fill-in, about the connectivity of G_h, and, consequently, about the sparsity of X_h. By usual arguments (see [GeL], [Li2], [GT]) we arrive at the following lemma:

LEMMA 7.1. *Let $h \geq 0$. Then the following hold*:

(a) *If p is a path in G_h between two vertices $i \notin V_{h^*,k}$ and $j \in R_{h^*,k}$ for some $h^* \geq h$ and some k, then p visits some vertex v such that $\pi(v) > \delta_{h^*+1}$, that is, $v \notin R_q$ for $q \leq h^*$.*

(b)

$$E_{h+1} = E_h^- \cup F_h,$$

$$E_h^- = \{\{i, j\} \in E_h | i, j \notin R_h\},$$

$$F_h = \cup\{\{i, j\} | \exists k \exists \{i, j_1\}, \{j_1, j_2\}, \ldots \{j_{l-1}, j_l\}, \{j_l, j\} \in E_h\}$$

provided that $j_1, \ldots, j_l \in R_{h,k}$ and that $\pi(i) > \delta_{h+1}$, $\pi(j) > \delta_{h+1}$ in the definition of F_h.

Lemma 7.1 defines the desired restriction on the edge connections in $G(A_h)$. In particular, part (a) of Lemma 7.1 implies that E_h contains no edge between $R_{h,k}$ and R_{h,k^*} for $k \neq k^*$.

Since π groups the vertices of $R_{h,k}$ together and since $\max_k |R_{h,k}| \leq \max_k s(|V_{h,k}|) \leq s(n_h)$, we immediately arrive at the following lemma:

LEMMA 7.2. *X_h is a block diagonal matrix consisting of $N_h \leq 2^{d-h}$ square blocks of sizes $|R_{h,k}| \times |R_{h,k}|$, so that each block is of size at most $s(n_h) \times s(n_h)$.*

Lemma 7.2 implies that for $h > 0$ the numerical inversion of X_h can be reduced to $N_h \leq 2^{d-h}$ parallel numerical inversions of generally dense matrices, each of size at most $s(n_h) \times s(n_h)$ (that is, one dense matrix is associated with each $R_{h,k}$, so that its size is at most $|R_{h,k}| \times |R_{h,k}|$). By Fact 2.1 such inversions can be performed in $O(\log^2 n)$ time with $N_h M(s(n_h)) \leq 2^{d-h} M(s(n_h))$ processors.

The next lemma is from [LT]. Its proof is simplified in [PR1] for the case in which $\alpha^* > \frac{1}{2}$, and we need only this case. Both proofs also show simple transformations of the respective $s(n)$-separator tree into $s^*(n)$-separator trees.

LEMMA 7.3. *For any triple of constants α, α^*, and n_0 such that $\frac{1}{2} \leq \alpha^* < \alpha < 1$ and $n_0 > 0$, if a graph has an $s(n)$-separator family with respect to α and n_0 (see Definition 3.1), then this graph has an $s^*(n)$-separator family with respect to α^* and n_0, where $s^*(n) \leq \sum_{h=0}^{d} s(n_h)$; in particular, $s^*(n) \leq cn^\sigma/(1-\beta^\sigma)$ if $s(n) \leq cn^\sigma$ for some positive constants c, β, and σ, where $\beta < 1$ (see (5.1), (6.1), (6.2)).*

LEMMA 7.4. *$2^{d-h} M(s(n_h)) \leq M(s(n))\eta^{d-h}$ for some $\eta < 1$.*

Proof. Equation (5.1) and relation (6.2) imply that $s(n_h) \leq c(\alpha+\delta)^{\sigma(d-h)} n^\sigma$, so that $M(s(n_h)) \leq c^{\omega^*}(\alpha+\delta)^{\sigma\omega^*(d-h)} n^{\sigma\omega^*}$ (see (5.2)). We may choose n_0 sufficiently large so as to make δ sufficiently small and then apply Lemma 7.3 to make sure that $\alpha+\delta$ lies as close to $\frac{1}{2}$ as we like. Since $\sigma\omega^* > 1$ (see (5.2)), we may assume that $(\alpha+\delta)^{\sigma\omega^*} < \frac{1}{2}$, so that $\eta = 2(\alpha+\delta)^{\sigma\omega^*} < 1$. Then

$$2^{d-h} M(s(n_h)) \leq \eta^{d-h} c^{\omega^*} n^{\sigma\omega^*} = \eta^{d-h} M(s(n)). \qquad \square$$

From Fact 2.1 and Brent's slowdown principle of parallel computation, we may invert X_h by using $O(k \log^2 n)$ steps and $\lceil 2^{d-h} M(s(n_h))/k \rceil \leq \lceil M(s(n))\eta^{d-h}/k \rceil$ processors for some $\eta < 1$ and for any k such that $1 \leq k = k(h)$. Choosing the minimum $k = k(h) \geq 1$ such that $M(s(n))\eta^{d-h}/k(h) \leq M(s(n))/\log n$ (so that $k(h) = \eta^{d-h}\log n$ if $h > d + \log\log n/\log \eta$ and $k(h) = 1$ otherwise), we simultaneously obtain the time bound $O(\log^3 n)$ (see Lemma 6.1) and the processor bound $M(s(n))/\log n$, required in Theorem 7.1. $\square$

8. Estimating the cost of parallel multiplication of auxiliary matrices.

THEOREM 8.1. *All the $2d$ matrix multiplications*

$$U_h = Y_h X_h^{-1}, \qquad W_h = U_h Y_h^T, \qquad h = 0, 1, \ldots, d-1 \tag{8.1}$$

involved in the recursive $s(n)$-factorization (4.1)–(4.4) *can be performed by using $O(\log^2 n)$ parallel time and $M(s(n))$ processors or (if we slow down the computations by a factor of $\log n$) by using $O(\log^3 n)$ parallel time and $M(s(n))/\log n$ processors.*

Proof. We will prove Theorem 8.1 by estimating the cost of parallel evaluation of the matrix products of (8.1) (given Y_h and X_h^{-1}) for $h = 0, 1, \ldots, d-1$. First we will arrange the matrix multiplications of (8.1) by reducing them to several matrix multiplications of the form

$$U_{h,k} = Y_{h,k} X_{h,k}^{-1}, \qquad W_{h,k} = U_{h,k} Y_{h,k}^T, \qquad k = 1, 2, \ldots, N_h, \quad h = 0, 1, \ldots, d-1. \tag{8.2}$$

To arrive at such a rearrangement, partition Y_h into N_h submatrices $Y_{h,k}$ having columns associated with the row sets $R_{h,k}$ and having the sizes $m_{h,k} \times |R_{h,k}|$, where $m_{h,k} \leq n - \delta_h$

for $k = 1, \ldots, N_h$. The dense diagonal blocks of X_h^{-1} are denoted $X_{h,k}^{-1}$, respectively. By the definition of G_h and T_G and by virtue of Lemma 7.1, the matrix $Y_{h,k}$ may have nonzero entries only in rows i such that i lies in one of the sets $R_{h+g,q}$ (for $1 \le g \le d-h$, $q = q(g,h,k)$) corresponding to an ancestor $(V_{h+g,q}, S_{h+g,q})$ of the node $(V_{h,k}, S_{h,k})$ in T_G.

To deduce the desired complexity estimates, examine the cost of all the latter matrix multiplications (8.2), grouping them not in the above *horizontal* order (where k ranges from 1 to N_h for a fixed h) but in the *vertical* order of Definition 3.1, that is, going from the root of the tree T_G to its leaves.

By slightly abusing the notation, denote $n = |R_{h,k}|$, $m = m_{h,k}$ for a fixed pair h and k, and consider the matrix multiplications of (8.2) associated with the node $(V_{h,k}, S_{h,k})$ and with its descendents in the tree T_G. These matrix multiplications can be performed in $O(\log^2 n)$ time (this is required in Theorem 8.1); let $P(n,m)$ denote the associated processor bound. For the two children of the node $(V_{h,k}, S_{h,k})$ the two associated numbers of processors will be denoted by $P(n_1, m_1)$ and $P(n_2, m_2)$, where, by virtue of Lemma 7.2 and Definition 3.1 (see also [LRT]),

$$\begin{gathered} m_1 + m_2 \le m + 2s(n), \\ n \le n_1 + n_2 \le n + s(n), \\ (1-\alpha)n \le n_i \le \alpha n + s(n) \quad \text{for } i = 1, 2. \end{gathered} \tag{8.3}$$

Let $M(p,q,r)$ hereafter denote the number of processors required in order to multiply $p \times q$ by $q \times r$ matrices in $O(\log(pqr))$ parallel steps, so that $M(p,q,r) \le M(q)\lceil p/q \rceil \lceil r/q \rceil$ (all the processor bounds have been defined up to within constant factors). For fixed h and k (and, therefore, for a fixed separator $S_{h,k}$) the matrix multiplications (8.2) can be performed by using $O(\log n)$ parallel steps and $M(s(n)+m, s(n), s(n)+m) \le \lceil (1 + m/s(n)) \rceil^2 M(s(n))$ processors. Therefore, recursively,

$$P(n,m) \le \left(1 + \left(1 + \frac{m}{s(n)}\right)^2\right) M(s(n)) + P(n_1, m_1) + P(n_2, m_2) \tag{8.4}$$

for some n_1, n_2, m_1, m_2 satisfying (8.3).

Using (8.4), we will prove the following claim, which in its special case for $m = 0$ amounts to Theorem 8.1 (recall that we already have the parallel time bound $O(\log^2 n)$ of this theorem):

CLAIM. $P(n,m) \le (c_0 + c_1(m/s(n)) + c_2(m/s(n))^2)M(s(n))$ *for all m and n and for some constants* c_0, c_1, c_2.

Proof. If $n \le n_0$, then $P(n,m) \le M(n) \le c_0$ provided that $c_0 \ge M(n_0)$. Thus let $n \ge n_0$ and prove the claim by induction on n. We may assume that n_0 is large enough, so that (8.3) implies that $n_i < n$ for $i = 1, 2$. Then by the induction hypothesis the claim holds if n is replaced by n_i for $i = 1, 2$, so that

$$P(n_1, m_1) + P(n_2, m_2) \le \sum_{i=1}^{2} \left(c_0 + c_1 \frac{m_i}{s(n_i)} + c_2 \left(\frac{m_i}{s(n_i)}\right)^2\right) M(s(n_i)).$$

Therefore,

$$\sum_i P(n_i, m_i) \le c_0 \sum_i M(s(n_i)) + c_1 \sum_i m_i \frac{M(s(n_i))}{s(n_i)} + c_2 \sum_i m_i^2 \frac{M(s(n_i))}{s(n_i)^2}. \tag{8.5}$$

Next we deduce from (8.3) that for $g = 1$ and $g = 2$

$$\begin{aligned}\sum_i m_i^g \frac{M(s(n_i))}{s(n_i)^g} &\le \left(\sum_i m_i^g\right) \max_i \left(\frac{M(s(n_i))}{s(n_i)^g}\right)\\ &\le (m + 2s(n))^g \frac{M(s(\alpha n + s(n)))}{s(\alpha n + s(n))^g}\\ &\le (m + 2s(n))^g \frac{M(s(\beta n))}{s(\beta n)^g} \quad \text{for } \beta < 1\end{aligned}$$

(apply (6.1) to deduce the last inequality). Applying here (5.1) and (5.2), we obtain that

$$\sum_i m_i^g \frac{M(s(n_i))}{s(n_i)^g} \le \gamma (m + 2s(n))^g \frac{M(s(n))}{s(n)^g}, \tag{8.6}$$

where $\gamma = \beta^{(\omega^* - g)\sigma}$ is a constant, $\gamma < 1$, $g = 1, 2$, $g < \omega^*$.

Furthermore, (5.1) and (8.3) imply that the sum $M(s(n_1)) + M(s(n_2))$ takes on its maximum value where one of n_1, n_2 is as large as possible (that is, equal to $\alpha n + s(n)$), which makes the other as small as possible (that is, equal to $(1-\alpha)n$). Therefore,

$$\begin{aligned}M(s(n_1)) + M(s(n_2)) &\le M(s(\alpha n + s(n))) + M(s((1-\alpha)n))\\ &\le M(s((\alpha + \delta)n)) + M(s((1-\alpha)n))\\ &= M(cn^\sigma(\alpha+\delta)^\sigma) + M(cn^\delta(1-\alpha)^\sigma)\end{aligned}$$

(see (5.1) and (6.1)). Applying here (5.3) and then (5.1) and (5.2), we deduce that

$$\begin{aligned}M(s(n_1)) + M(s(n_2)) &\le (M((\alpha+\delta)^\sigma) + M((1-\alpha)^\sigma))M(cn^\sigma)\\ &\le ((\alpha+\delta)^{\omega^*\sigma} + (1-\alpha)^{\omega^*\sigma})M(s(n)),\end{aligned}$$

where $\omega^*\sigma > 1$. The positive δ can be assumed to be arbitrarily close to 0, and so we deduce that

$$M(s(n_1)) + M(s(n_2)) \le vM(s(n)) \tag{8.7}$$

for a constant $v < 1$.

Combining (8.4)–(8.7), we obtain that

$$\begin{aligned}P(n,m) \le\ & (2 + vc_0 + 2\gamma c_1 + 4\gamma c_2)M(s(n))\\ &+(2 + \gamma c_1 + 4\gamma c_2)M(s(n))\frac{m}{s(n)}\\ &+(1 + \gamma c_2)M(s(n))\left(\frac{m}{s(n)}\right)^2\end{aligned} \tag{8.8}$$

for two constants $\gamma < 1$, $v < 1$. We choose c_2 large enough, so that $1 + \gamma c_2 \le c_2$, we then choose c_1 large enough so that $2 + \gamma c_1 + 4\gamma c_2 \le c_1$, and, finally, we chose c_0 large enough, so that $2 + vc_0 + 2\gamma c_1 + 4\gamma c_2 \le c_0$. Then (8.8) implies the claim and, consequently, Theorem 8.1. □

Acknowledgments. The authors thank the referees for numerous helpful suggestions. The paper was typed by Sally Goodall (with assistance from Joan Bentley, Bettye Kirkland, and Chris Lane).

REFERENCES

[AR] D. Armon and J. Reif, *Space and time efficient implementation of a parallel nested dissection*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1992, pp. 344–352.

[BCLR] D. Bini, M. Capovani, G. Lotti, and F. Romani, $O(n^{2.7799})$ *complexity for matrix multiplication*, Inform. Process. Lett., 8 (1979), pp. 234–235.

[Be] A. Ben-Israel, *A note on iterative methods for generalized inversion of matrices*, Math. Comput., 20 (1966), pp. 439–440.

[BGH] A. Borodin, J. von zur Gathen, and J. Hopcroft, *Fast parallel matrix and* GCD *computation*, Inform. and Control, 52 (1982), pp. 241–256.

[Bi] D. Bini, *Relations between* EC-*algorithms and* APA-*algorithms*: *Applications*, Calcolo, 17 (1980), pp. 87–97.

[BM] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[BP] D. Bini and V. Pan, *Numerical and Algebraic Computations with Matrices and Polynomials*, Birkhäuser-Verlag, Boston, 1993.

[Ca] D. A. Calahan, *Parallel solution of sparse simultaneous linear equations*, in Proc. 11th Allerton Conference, 1973, pp. 729–738.

[Ch] A. K. Chandra, *Maximal Parallelism in Matrix Multiplication*, Report RC-6193, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1976.

[Cs] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[CW1] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in Proc. 19th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 1–6; J. Symbolic Comput., 9 (1990), pp. 251–280.

[CW2] ———, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput., 11 (1982), pp. 472–492.

[D] H. N. Djidjev, *On the problem of partitioning planar graphs*, SIAM J. Alg. Discrete Meth., 3 (1982), pp. 229–240.

[EG] D. Eppstein and Z. Galil, *Parallel algorithmic techniques for combinatorial computation*, Annual Rev. Comput. Sci., 3 (1988), pp. 233–283.

[Ga] D. A. Gannon, *A note on pipelining mesh-connected multiprocessor for finite element problems by nested dissection*, in Proc. International Conference on Parallel Processing, 1980, pp. 197–204.

[GeL] J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Ge] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–367.

[GHLN] A. George, M. T. Heath, J. W. H. Liu, and E. G. Y. Ng, *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 327–340.

[GM1] H. Gazit and G. L. Miller, *A parallel algorithm for finding a separator in planar graphs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1987, pp. 238–248.

[GM2] ———, private communication, 1992.

[GoL] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.

[GP1] Z. Galil and V. Pan, *Improving processor bounds for algebraic and combinatorial problems in* RNC, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1985, pp. 490–495.

[GP2] ———, *Improved processor bounds for combinatorial problems in* RNC, Combinatorica, 8 (1988), pp. 189–200.

[GP3] ———, *Parallel evaluation of the determinant and of the inverse of a matrix*, Inform. Process. Lett., 30 (1989), pp. 41–45.

[GT] J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numer. Math., 50 (1987), pp. 377–404.

[KP] E. Kaltofen and V. Pan, *Processor efficient solution of linear systems over an abstract field*, in Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architecture, Association for Computing Machinery, New York, 1991, pp. 180–191,

[KR] R. Karp and V. Ramachandran, *A Survey of Parallel Algorithms for Shared Memory Machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[KS] E. Kaltofen and M. Singer, *Size Efficient Parallel Algebraic Circuits for Partial Derivatives*, Tech. Report 90-32, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1990.

[Li1] J. W. H. Liu, *The Solution of Mesh Equations on a Parallel Computer*, Tech. Report CS-78-19, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1978.

[Li2] ———, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. Math. Software, 12 (1986), pp. 127–148.

[LMNOR] C. E. Leiserson, J. P. Mesirov, L. Nekludova, S. Omahundro, and J. Reif, *Solving sparse systems of linear equations on the connection machine*, in Proc. Annual SIAM Conference, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986, p. A51.

[Lo] L. Lovász, *Connectivity algorithms using rubber-bands*, in Proc. 6th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 241, Springer-Verlag, Berlin, 1986, pp. 394–412.

[LRT] R. J. Lipton, D. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[LT] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[MVV] K. Mulmuley, U. Vazirani, and V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–114.

[OR] T. Opsahl and J. Reif, *Solving sparse systems of linear equations on the massive parallel machine*, in Proc. 1st Symposium on Frontiers of Scientific Computing, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, MD, 1986, pp. 2241–2248.

[OV] J. M. Ortega and R. G. Voight, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149–240.

[Pan1] V. Pan, *On schemes for the evaluation of products and inverses of matrices*, Uspekhi Mat. Nauk, 27 (1972), pp. 249–250.

[Pan2] ———, *Strassen's algorithm is not optimal. Trilinear technique of aggregating, uniting, and canceling for constructing fast algorithms for matrix multiplication*, in Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1978, pp. 166–176.

[Pan3] ———, *Fields extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplication*, in Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1979, pp. 28–38.

[Pan4] ———, *New fast algorithms for matrix operations*, SIAM J. Comput., 9 (1980), pp. 321–342.

[Pan5] ———, *New combinations of methods for the acceleration of matrix multiplications*, Comput. Mach. Appl., 7 (1981), pp. 73–125.

[Pan6] ———, *How can we speed up matrix multiplication?* SIAM Rev., 26 (1984), pp. 393–415.

[Pan7] ———, *How to Multiply Matrices Faster*, Lecture Notes in Computer Science 179, Springer-Verlag, Berlin, 1984.

[Pan8] ———, *Fast and Efficient Parallel Algorithms for the Exact Inversion of Integer Matrices*, Tech. Report 85-2, Department of Computer Science, State University of New York, Albany, NY, 1985.

[Pan9] ———, *Fast and efficient parallel algorithms for the exact inversion of integer matrices*, in Proc. 5th Conference on the Foundation of Software Technology, Lecture Notes in Computer Science 206, Springer-Verlag, Berlin, 1985, pp. 504–521.

[Pan10] ———, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[Pan11] ———, *Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications*, Tech. Report CUCS-032-90, Department of Computer Science, Columbia University, New York, 1990; Comput. Math. Appl., 24 (1992), pp. 61–75.

[Par] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[PaS] V. Pan and R. Schreiber, *An improved Newton iteration for the generalized inverse of a matrix, with applications*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1109–1131.

[PP] V. Pan and F. P. Preparata, *Supereffective slowdown of parallel computations*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1992, pp. 402–409.

[PR1] V. Pan and J. Reif, *Efficient parallel solution of linear systems*, in Proc. 17th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1985, pp. 143–152.

[PR2] ———, *Efficient parallel linear programming*, Oper. Res. Lett., 5 (1986), pp. 127–135.

[PR3] V. Pan and J. Reif, *Fast and efficient algorithms for linear programming and for the linear least squares problem*, Comput. Math. Appl., 12A (1986), pp. 1217–1227.

[PR4] ———, *Parallel nested dissection for path algebra computations*, Oper. Res. Lett., 5 (1986), pp. 177–184.

[PR5] ———, *Fast and efficient parallel solution of dense linear systems*, Comput. Math. Appl., 17 (1989), pp. 1481–1491; preliminary version in Proc. 17th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1985, pp. 143–152.

[PR6] ———, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.

[PR7] ———, *The parallel computation of minimum cost paths in graphs by stream contraction*, Inform. Process. Lett., 49 (1991), pp. 79–83.

[PrS] E. P. Preparata and D. V. Sarwate, *An improved parallel processor bound in fast matrix inversion*, Inform. Process. Lett., 7 (1978), pp. 148–149.

[R] D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, pp. 183–217.

[Schö] A. Schönhage, *Partial and total matrix multiplication*, SIAM J. Comput., 19 (1981), pp. 434–456.

[Schr] R. Schreiber, *A new implementation of sparse Gaussian elimination*, Trans. Math. Software, 8 (1982), pp. 256–276.

[Stra1] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[Stra2] ———, *Relative bilinear complexity and matrix multiplication*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1986, pp. 49–54.

[T1] R. E. Tarjan, *Fast algorithms for solving path problems*, J. Assoc. Comput. Mach., 28 (1981), pp. 594–614.

[T2] ———, *A unified approach to path problems*, J. Assoc. Comput. Mach., 28 (1981), pp. 577–593.

[ZG] E. Zmijewski and J. R. Gilbert, *A parallel algorithm for sparse Cholesky factorization on a multiprocessor*, Parallel Comput., 7 (1988), pp. 199–210.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1251–1256, December 1993

ON-LINE BIN PACKING OF ITEMS OF RANDOM SIZES, II*

WANSOO T. RHEE† AND MICHEL TALAGRAND‡

Abstract. This paper describes the construction of an on-line algorithm with the following property. There exist universal constants K, α such that given any probability measure μ on [0, 1] and a sequence $X_1, \ldots, X_n, \ldots$ of items independently and identically distributed according to μ, the algorithm packs $X_1, \ldots, X_n$ into at most $T_n(X_1, \ldots, X_n) + Kn^{1/2}(\log n)^{3/4}$ unit-size bins, where $T_n(X_1, \ldots, X_n)$ denotes the minimum number of bins needed to pack $X_1, \ldots, X_n$, and does this with probability greater than or equal to $1 - K\exp(-\alpha(\log n)^{3/2})$. In contrast with the authors' previous work on this problem, the algorithm is now independent of μ.

Key words. stochastic, bin packing, on-line algorithm

AMS subject classifications. 90B05, 05B430

1. Introduction.

The bin-packing problem requires finding the minimum number $T_n(x_1, \ldots, x_n)$ of unit-size bins needed to pack items of sizes $x_1, \ldots, x_n$, subject to the condition that the sum of the sizes of the items attributed to any given bin does not exceed unity. The present work is concerned with on-line methods of packing; that is, the items $x_1, x_2, \ldots$ are inspected one at a time. (For simplicity, we denote item names and item sizes by the same letter.) The decision concerning which bin to attribute x_p is taken at the time x_p is inspected and cannot be modified later. Celebrated on-line algorithms include Next Fit, First Fit, and Best Fit. In this paper we focus on the model in which the items $X_1, \ldots, X_n$ are identically distributed according to a given distribution μ. This model has been investigated in detail for general distributions [5], [7], [8]. It is known that

$$c(\mu) = \lim_{n\to\infty} n^{-1} T_n(X_1, \ldots, X_n)$$

exists almost everywhere and that

$$P(T_n(X_1, \ldots, X_n) \geq nc(\mu) + tn^{1/2}) \leq K\exp(-\alpha t^2) \tag{1.1}$$

for some universal constants K, α. However, to pack $X_1, \ldots, X_n$ in $nc(\mu) + O(n^{1/2})$ bins one needs to know the whole sequence $X_1, \ldots, X_n$, and the case of on-line procedures is rather different from most. Shor [11] showed that when μ is uniform on [0, 1], any on-line procedure must use $n/2 + \Omega((n\log n)^{1/2})$ bins and that the heuristic Best Fit uses $n/2 + O(n^{1/2}(\log n)^{3/4})$ bins. In a previous work [9] the authors showed that, given any distribution μ, one can (in principle) find an on-line algorithm (depending on μ) that, with probability $1 - \exp(-\alpha(\log n)^{3/2})$, packs the items $X_1, \ldots, X_n$ into at most $nc(\mu) + Kn^{1/2}(\log n)^{3/4}$ bins, where K, α are universal constants. The contribution of the present work is to make this algorithm independent of μ. Moreover, in contrast to the result of [9], which was an existence theorem rather than an explicit algorithm, the method presented here is simple and explicit.

THEOREM 1.1. *There exist universal constants K, α and an on-line algorithm with the following property. Given a sequence $X_1, \ldots, X_n, \ldots$ of items that are independently distributed according to a given distribution μ, the algorithm packs the items $X_1, \ldots, X_n$ into at most $nc(\mu) + Kn^{1/2}(\log n)^{3/4}$ bins with probability greater than or equal to $1 - K\exp(-\alpha(\log n)^{3/4})$.*

*Received by the editors May 26, 1992; accepted for publication June 24, 1992. This work was partially supported by National Science Foundation grants CCR-9000611 and DMS-9101452.

†Department of Management Sciences, Ohio State University, 1775 College Road, Columbus, Ohio 43210.

‡Equipe d'Analyse – Tour 48, Universite Paris VI, European Research Associates, Centre National de la Recherche Scientifique, No. 294, 4 Place Jussieu, 75230 Paris Cedex 05, France, and Department of Mathematics, Ohio State University, 231 West 18th Avenue, Columbus, Ohio 43210.

2. Description of the algorithm. The construction is based on an auxiliary algorithm (Algorithm A) whose performance is described as follows.

THEOREM 2.1. *Consider items $x_1, \dots, x_q$, and consider an independent sequence of integer-valued random variables $(r(i))_{i\geq 1}$ distributed uniformly over $1, \dots, q$. From the knowledge of a packing of $x_1, \dots, x_q$ into m bins, one can construct an on-line algorithm that packs the sequence $(x_{r(i)})_{i\leq p}$ into at most $\frac{m}{q}p + K\sqrt{p}(\log p)^{3/4}$ bins with probability greater than or equal to $1 - K\exp(-\alpha(\log p)^{3/2})$, where K, α are universal constants.*

We now describe the algorithm of Theorem 1.1. We put X_1 in the first bin. Upon packing X_{2^k} $(k \geq 1)$, we forget everything that was done up to this point. We construct a packing of $X_1, \dots, X_{2^k}$ into m_k bins, where $m_k \leq T_{2^k}(X_1, \dots, X_{2^k}) + 2^{k/2}k^{3/4}$ bins (that this is possible in practice is shown in [3]). Using this packing, we then construct the auxiliary algorithm provided by Theorem 2.1. We generate a sequence $(r(i))_{i\leq 2^k}$ that is independently and uniformly distributed over $\{1, \dots, 2^k\}$. We now proceed to the packing of the items $(X_i)_{2^k<i\leq 2^{k+1}}$. These are packed in two new sequences of bins, and we make no attempt to use the wasted spaces left in the bins used in the packing of $X_1, \dots, X_{2^k}$. For the purpose of this packing we keep track of a subset S_i of $\{1, \dots, 2^k\}$. We set $S_0 = \emptyset$. Upon inspection of X_i, we first pack $X_{r(i)}$ according to the auxiliary algorithm in the first new sequence of bins. We determine whether there is $j \in S_{i-1} \cup \{i\}$ such that $X_i \leq X_{r(j)}$. If this is not the case, we put X_i alone in a bin of the second new sequence, we say that X_i is unmatched, and we set $S_i = S_{i-1} \cup \{i\}$. Otherwise, we select j such that $X_{r(j)} \geq X$ is minimum, and we set $S_i = S_{i-1} \cup \{i\}\backslash\{j\}$. We then remove $X_{r(j)}$ from its bin, and we put X_i in its place.

3. Proof of Theorem 1.1. Consider $n \geq 1$, and denote by k_1 the largest integer such that $2^{k_1} \leq n$. In the packing of the items $X_1, \dots, X_n$ by our algorithm, we call the kth *stage of the packing* the packing of items $X_{2^k+1}, \dots, X_{\min(n,2^{k+1})}$. We denote by A_k the number of bins used by the auxiliary algorithm during the kth stage and by M_k the number of unmatched items during that stage. The total number of bins used by the algorithm is $\sum_{k\leq k_1}(A_k + M_k)$.

We observe that obviously

$$\sum_{k\leq k_1/2}(A_k + M_k) \leq \sum_{k\leq k_1/2} 2\cdot 2^k \leq 4\cdot 2^{k_1/2}. \tag{3.1}$$

We set $m_k = 2^k$ if $k < k_1$, $m_{k_1} = n - 2^{k_1}$. For definiteness we assume now that $m_{k_1} \geq 2^{k_1/2}$ (we will explain later what modification to make when $m_{k_1} \leq 2^{k_1/2}$). By Theorem 2.1 we have for $k_1/2 \leq k \leq k_1$ that

$$A_k \leq \frac{m_k}{2^k}(T_{2^k}(X_1, \dots, X_{2^k}) + 2^{k/2}k^{3/4}) + K\sqrt{m_k}(\log n)^{3/4}$$

with probability greater than or equal to $1 - K\exp(-\alpha(\log 2^{k_1/2})^{3/2})$.

By (1.1) we have for $k_1/2 \leq k \leq k_1$ that

$$P(T_{2^k}(X_1, \dots, X_{2^k}) \leq 2^k c(\mu) + k^{3/4}2^{k/2}) \geq 1 - K\exp\left(-\alpha k_1^{3/2}\right).$$

(Here and in the following the numbers K and α denote constants that are > 0 and may change at each occurrence.) Thus with probability greater than or equal to $1-Kk_1\exp(-\alpha(\log 2^{k_1/2})^{3/2})$ we have

$$k_1/2 \leq k \leq k_1 \Rightarrow A_k \leq m_k(c(\mu) + 2\cdot 2^{-k/2}k^{3/4}) + K\sqrt{m_k}(\log n)^{3/4}.$$

Thus with the same probability we obtain that

$$\sum_{k\leq k_1} A_k \leq nc(\mu) + K\sqrt{n}(\log n)^{3/4}.$$

It remains to study $\sum_{k_1/2\le k\le k_1} M_k$. This is certainly the nontrivial part of the argument. We first turn toward the explanation of the basic concepts. Consider two subsets I, J of $\mathbb{N}$, and numbers $(x_i)_{i\in I}$, $(y_i)_{i\in J}$. A *matching* of these lists is a one-to-one map from a subset B of I to J such that $\varphi(i) \le i$ and $y_{\varphi(i)} \ge x_i$. We say that x_i is unmatched if $i \notin B$. A matching is called *maximal* if Card B is as large as possible.

Consider the following procedure. The elements $(x_i)_{i\in I}$ are examined in turn (as ranked by their indices). If there exists $j \in J$, $j \le i$, such that $y_j \ge x_i$, and if y_j has not yet been matched, we consider such an index j for which y_j is as small as possible and we set $j = \varphi(i)$; otherwise, we leave x_i unmatched. It is shown in [11] that this procedure always constructs a maximum matching.

LEMMA 3.1. *Consider two probability measures μ, ν on* [0, 1]. *Consider a sequence $(X_i)_{i\le n}$ (respectively, $(Z_i)_{i\le n}$) that is independently distributed according to μ (respectively, ν). Set*

$$a = \sup(\mu([t, 1]) - \nu([t, 1]) : 0 \le t \le 1).$$

Then, with probability greater than or equal to $1 - K\exp -\alpha(\log n)^{3/2}$, a maximal matching of the sequences $(Y_i)_{i\le n}$ and $(Z_i)_{i\le n}$ will leave at most $na + Kn^{1/2}(\log n)^{3/4}$ unmatched items.

Proof. Step 1. We reduce the problem to the case $a = 0$. We can assume that $Y_i = F(U_i)$, where the sequence $(U_i)_{i\le n}$ is independently and uniformly distributed over [0, 1] and where F is nondecreasing. We set

$$Y_i' = 0 \quad \text{if } U_i \ge 1 - a,$$

$$Y_i' = Y_i \quad \text{if } U_i < 1 - a.$$

The sequence $(Y_i')_{i\le n}$ is independently and identically distributed. The distribution μ' of this sequence satisfies, for $t > 0$,

$$\mu'([t, 1]) = \max(\mu([t, 1]) - a, 0).$$

It follows that

$$\mu'([t, 1]) \le \nu([t, 1])$$

for $0 \le t \le 1$.

Let $I = \{i \le n : Y_i = Y_i'\}$. Consider a maximal matching of the sequences $(Y_i')_{i\le n}$ and $(Z_i)_{i\le n}$, and denote by m the number of unmatched points. The restriction of this matching to $(Y_i)_{i\in I}$ gives a matching of the sequences $(Y_i)_{i\in I}$ and $(Z_i)_{i\le n}$ such that the number of unmatched points is at most

$$m + n - \text{Card } I = m + \text{Card}\{i \le n : U_i \ge 1 - a\}.$$

It follows from the inequality of Hoeffding [2] that we have

$$\text{Card}\{i \le n : U_i \ge 1 - a\} \le na + Kn^{1/2}(\log n)^{3/4}$$

with probability greater than or equal to $1 - K\exp(-\alpha(\log n)^{3/2})$. This completes this step.

Step 2. Consider now two sequences $(U_i)_{i\le n}$, $(V_i)_{i\le n}$ that are independently and uniformly distributed on [0, 1]. We can assume that $Y_i = F(U_i)$, $Z_i = G(V_i)$, where F, G are given by

$$F(u) = \inf\{t : \mu([t, 1]) \ge 1 - u\},$$

$$G(u) = \inf\{t : \nu([t, 1]) \ge 1 - u\}.$$

Thus since $a = 0$, we have $G(u) \geq F(u)$. Thus a matching of the sequences $(U_i)_{i\leq n}$ and $(V_i)_{i\leq n}$ induces in a natural way a matching of the sequences $(Y_i)_{i\leq n}$ and $(Z_i)_{i\leq n}$.

Step 3. We have reduced the problem to the case in which $(Y_i)_{i\leq n}$ and $(Z_i)_{i\leq n}$ are uniformly distributed on $[0, 1]$. In that case the problem is equivalent to the "upright matching problem" solved in [10] and, independently, in [6].

We set

$$a_k = \sup_{0\leq t\leq 1} (\mu([t, 1]) - 2^{-k}\mathrm{Card}\{i \leq 2^k : X_i \geq t\}).$$

It follows from Lemma 3.1 that conditionally on $X_1, \dots, X_{2^k}$ the number of unmatched items in the kth stage is less than or equal to $Km_k^{1/2}k^{3/4} + m_k a_k$, with probability greater than or equal to $1 - K\exp(-\alpha k_1^{3/2})$ (at this point we use the fact that $m_1 \to 2^{k_1/2}$). On the other hand, by the Kolmogorov–Smirnov statistics, we have $P(a_k \geq 2^{-k/2}k^{3/4}) \leq 2\exp -\alpha k^{3/2}$. Thus with probability greater than or equal to $1 - K\exp(-\alpha k_1^{3/2})$ we have $M_k \leq K2^{k/2}k^{3/4}$. Thus

$$\sum_{k_1/2\leq k\leq k_1} M_k \leq K\sqrt{n}(\log n)^{3/4}$$

with probability greater than or equal to $1 - Kk_1\exp(-\alpha k_1^{3/2})$.

Finally, we see that with probability greater than or equal to $1 - Kk_1\exp(-\alpha k_1^{3/2})$ the number of bins used is less than or equal to $nc(\mu) + K\sqrt{n}(\log n)^{3/4}$.

In the case $m_{k_1} \leq 2^{k_1/2}$, the same conclusion still holds if the estimate $M_{k_1} + A_{k_1} \leq 2m_{k_1}$ is used.

To conclude the proof it suffices to observe that

$$k_1 \exp -\alpha k_1^{3/2} \leq K\exp\left(\frac{-\alpha k_1^{3/2}}{2}\right). \qquad \square$$

4. Auxiliary algorithm. In this section we construct the algorithm described in Theorem 2.1. This algorithm follows the ideas of [9], but considerable simplification is obtained from the fact that we model our strategy from a packing of $x_1, \dots, x_q$ instead of from the measure-theoretic decomposition of [5], so that all measure-theoretic difficulties disappear.

We consider a packing of $x_1, \dots, x_q$ into m bins. This packing will be referred to as the *model packing* (since it will be used as our model). The packing of the items $(x_{r(i)})$ will be done in two sequences of bins. The first is called the *main sequence*, and by *bin s* we will always mean *of the main sequence*. The second sequence is called the *overflow sequence*. Its purpose is to get rid of those few items that will not fit in the main packing scheme. Items sent to the overflow sequence are packed, say, by using Next Fit, so that if S denotes the sum of their sizes, they will use at most $1 + 2S$ bins.

During the packing of $(x_{r(i)})_{i\leq p}$ we keep track of an integer N that is the number of bins of the main sequence that contain at least one item. We keep track of lists $(L_j)_{2\leq j\leq q}$. An element of L_j is a couple (x, s), where $s \leq N$, $0 \leq x \leq 1$. It indicates that in bin s (of the main sequence) there is a space of size x that is reserved for a certain type of item.

Before packing $x_{r(1)}$ we have $N = 0$ and all the lists are empty. On inspection of $x_{r(i)}$, we proceed as follows.

Case 1. $x_{r(i)}$ is the largest element in the bin of the model packing that contains it (ties are broken once and for all arbitrarily). We set $N =: N + 1$, and we put $x_{r(i)}$ into the bin $N + 1$ of the main sequence. We denote by $z_2 \geq \cdots \geq z_h$ the other elements of the model bin that contains $x_{r(i)}$. For $2 \leq j \leq h$ we add the couple $(z_j, N + 1)$ to the list L_j.

Case 2. $x_{r(i)}$ is the j-largest element in the model bin that contains it, for $j \geq 2$. We check whether the list L_j contains a couple (x, s) for which $x \geq x_{r(i)}$. If this is not the case, we send $x_{r(i)}$ to the overflow sequence. If this is the case, we select such a couple (x, s) for which $x \geq x_{r(i)}$ is as small as possible. We put $x_{r(i)}$ in bin s, and we remove (x, s) from the list L_j.

5. Proof of Theorem 2.1. It should be clear that the sum of the sizes of the items attributed to any given bin of the main sequence is less than or equal to 1. Upon the packing of $x_{r(p)}$, the number of bins used by the auxiliary algorithm is $\alpha \leq N + 2S + 1$, where $N = N(p)$ is the current value of N and S is the sum of the sizes of the overflow items up to that point.

There is a probability $\frac{m}{q}$ that $x_{r(i)}$ is the largest element in the bin of the model packing that contains it. These probabilities are independent, and $N(p)$ is the number of indexes $i \leq p$ for which this event occurred. Thus, by the inequality of Hoeffding we have

$$P\left(N(p) \geq p\frac{m}{q} + p^{1/2}(\log p)^{3/2}\right) \leq K\exp\left(-\alpha(\log p)^{3/2}\right).$$

It now remains to evaluate S. This will rely on a slightly different version of the matching principles used in §3. Consider a subset I of $\{1, \ldots, p\}$ and a sequence $(C_i)_{i\in I}$, where C_i is a couple (w_i, ε_i), $w_i \in [0, 1]$, $\varepsilon_i = \pm 1$. Set $I^+ = \{i \in I : \varepsilon_i = +1\}$, $I^- = \{i \in I : \varepsilon_i = -1\}$. We define a matching of the sequence $(C_i)_{i\in I}$ as a matching of the lists $(w_i)_{i\in I^+}$ and $(w_i)_{i\in I^-}$. We fix $j \leq q$. We define the set I_j and the sequence $(C_i^j)_{i\in I_j}$ as follows. If $x_{r(i)}$ is the largest element of the bin of the model packing that contains it and if this bin contains at least j items, we put i in I_j, we set $\varepsilon_i = -1$, and we define w_i as the size of the j-largest item of this bin. If $x_{r(i)}$ is the j-largest item in the bin of the model packing that contains it ($j \geq 2$), we put i in I_j, we set $\varepsilon_i = +1$, and we set $w_i = x_{r(i)}$. We observe that $w_i \leq 1/j$.

We also observe that, conditionally on $i \in I_j$, we have $P(\varepsilon_i = 1) = P(\varepsilon_i = -1) = 1/2$. Also, j being fixed, the events $i \in I_j$ are independent and each has probability $2m_j/q$, where m_j is the number of bins of the model packing that contain at least j items.

Among the items $X_{r(i)}$, $i \in I_j$, those that are sent to the overflow sequence are exactly those for which $i \in I_j^+$ and that are left unmatched in the matching of the lists $(C_i^j)_{i\in I_j^+}$ and $(C_i^j)_{i\in I_j^-}$. As mentioned, this is the number of unmatched items in a maximal matching of these sequences. Conditionally on I_j, the items $(w_i)_{i\in I_j}$ are independently and identically distributed. It thus follows from the results of [11] (and the argument of Step 2 of the proof of Lemma 3.1) that conditionally on I_j the number of unmatched items is less than or equal to $K\sqrt{p_j}(\log p_j)^{3/4}$ with probability greater than or equal to $1 - K\exp(\alpha(\log p_j)^{3/2})$, where $p_j =$ Card I_j. By distinguishing whether $p_j \leq p^{1/3}$ or $p_j \geq p^{1/3}$ we see that, conditionally on I_j, the number n_j of unmatched items in the list $(C_i^j)_{i\in I_j^+}$ is less than or equal to $p^{1/3} + K\sqrt{p_j}\,(\log p)^{3/4}$ with probability greater than or equal to $1 - K\exp(-\alpha(\log p)^{3/2})$. The inequality of Hoeffding shows that

$$p_j \leq \frac{2m_j}{q}p + K\sqrt{p}(\log p)^{3/4}$$

with probability greater than or equal to $1 - K\exp(-\alpha(\log p)^{3/2})$. Thus, with probability greater than or equal to $1 - Kp\exp(-\alpha(\log p)^{3/2})$, for all $j \leq q$ we have

$$p_j \leq \frac{2m_j}{q}p + K\sqrt{p}(\log p)^{3/4},$$

so that

$$\sqrt{p_j} \leq K\left(\sqrt{\frac{p}{q}}\sqrt{m_j} + p^{1/4}(\log p)^{3/8}\right)$$

and hence $n_j \le h_j$, where

$$h_j = p^{1/3} + K\left(\sqrt{\frac{p}{q}}\sqrt{m_j}(\log p)^{3/4} + p^{1/4}(\log p)^{9/8}\right).$$

The sum of the sizes of the unmatched items in the lists (C^j) for $j \le q$ is at most $\sum_{j\le q} n_j/j$. Since $n_j \le h_j$ and $\sum_{j\le q} n_j \le p$, it should be clear that this quantity is bounded by $\sum_{j\le s} h_j/j$, where $s = q$ if $\sum_{j\le q} h_j \le p$, or else s is the smallest integer for which $\sum_{j\le s} h_j > p$. Since $h_j \ge p^{1/3}$, we have $s \le p$, so that $\sum_{j\le s} 1/j \le K \log p$.

Since by Cauchy–Schwarz we have

$$\sum_{j\le q} \frac{\sqrt{m_j}}{j} \le \left(\sum m_j\right)^{1/2} \left(\sum \frac{1}{j^2}\right)^{1/2} \le K\sqrt{q},$$

we see that $\sum_{j\le s} h_j/j \le K\sqrt{p}\,(\log p)^{3/4}$. □

REFERENCES

[1] M. Ajtai, J. Komlos, and G. Tusnady, *On optimal matchings*, Combinatorica, 4 (1984), pp. 259–264.

[2] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc. 58 (1963), pp. 13–30.

[3] N. Karmakar and R. Karp, *An efficient approximation scheme for the one-dimensional bin packing problem*, Proc. 23rd Annual IEEE Symposium on Foundation of Computer Science, IEEE Computer Society, Washington, DC, 1982, pp. 312–320.

[4] T. Leighton and P. Shor, *Tight bounds for minimax grid matching with applications to the average case analysis of algorithms*, in Proc. 18th Annual Symposium on Theory of Computing, Association for Computing Machinery, New York, 1986, pp. 91–103.

[5] W. Rhee, *Optimal bin packing with items of random sizes*, Math. Oper. Res. 13 (1988), pp. 140–151.

[6] W. Rhee and M. Talagrand, *Exact bounds for the stochastic upward matching problem*, Trans. Amer. Math. Soc. 307(1988), pp. 109–125.

[7] ———, *Optimal bin packing with items of random sizes* II, SIAM J. Comput. 18 (1989), pp. 139–151.

[8] ———, *Optimal bin packing with items of random sizes* III, SIAM J. Comput. 18 (1989), pp. 473–486.

[9] ———, *On line bin packing of items of random sizes*, Math. Oper. Res. to appear.

[10] P. Shor, *Random Planar Matching and Bin Packing*, Ph.D. thesis, Mathematics Department, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[11] ———, *The average case analysis of some on-line algorithms for bin packing*, Combinatorica, 6 (1986), pp. 179–200.

[12] M. Talagrand, *Matching theorems and empirical discrepancy computations using majorizing measures*, unpublished manuscript, 1991.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1257–1275, December 1993

ON THE COMPUTATIONAL COMPLEXITY OF SMALL DESCRIPTIONS*

RICARD GAVALDÀ[†] AND OSAMU WATANABE[‡]

Abstract. For a set L that is polynomial-time reducible (in short, $\leq_T^P$-reducible) to some sparse set, the authors investigate the computational complexity of such sparse sets relative to L and prove the first lower bounds on the complexity of recognizing such sets. Sets A and B are constructed such that both of them are $\leq_T^P$-reducible to *some* sparse set, but A (respectively, B) is $\leq_T^P$-reducible to *no* sparse set in P^A (respectively, $NP^B \cap \text{co-}NP^B$); that is, the complexity of sparse sets to which A (respectively, B) is $\leq_T^P$-reducible is more than P^A (respectively, $NP^B \cap \text{co-}NP^B$). From these results or application of the proof technique the authors obtain (1) lower bounds for the relative complexity of generating polynomial-size circuits for some sets in P/poly and (2) separations of the classes of sets equivalent or reducible to sparse sets under various polynomial-time reducibilities.

Key words. computational complexity, sparse sets, tally sets, polynomial-time reducibility, polynomial-time equivalence, Kolmogorov complexity

AMS subject classifications. 68Q05, 68Q15, 68Q30

1. Introduction.

In computational complexity theory or, more in general, in the theory of computation, sets with a small description have often been the subject of investigation. In this paper we study the computational complexity of (recognizing or generating) such small descriptions relative to an original set. More specifically, for a set L that is polynomial-time reducible to some sparse set, we investigate the complexity of such sparse sets relative to L.

The relative complexity of small descriptions is important in several contexts. Let us take, for example, "concept learning" as studied in [4], [5]. Roughly speaking, the goal of concept learning is to obtain a (small) description of an unknown set X. In particular, "query learning," the type of learning discussed in [4], [5], achieves this task by asking queries on X to a teacher; thus one can formalize query learning as a type of oracle computation, where an unknown set is used as an oracle [24]. Therefore, the relative complexity of finding a description of an unknown set is a key factor in order to discuss whether a given concept class is, say, polynomial-time learnable.

Now we explain how the problem is formalized in this paper. $R_T(\text{SPARSE})$[1] is the class of sets that are polynomial-time reducible (in short, $\leq_T^P$-reducible) to some sparse set. Recall that a set S is *sparse* if for some polynomial p and for all $n \geq 0$, S has at most $p(n)$ elements of length $\leq n$. Hence one can encode a sparse set up to a given length n in a string of length polynomial in n. Thus $R_T(\text{SPARSE})$ sets can be regarded as sets with small descriptions. There is another interpretation of sets in $R_T(\text{SPARSE})$. It is well known (see, e.g., [7]) that a set belongs to $R_T(\text{SPARSE})$ if and only if it has polynomial-size circuits. Since a family of polynomial-size circuits can be considered as a small description, $R_T(\text{SPARSE})$ sets are, also in this sense, sets with small descriptions.

*Received by the editors September 25, 1991; accepted for publication (in revised form) July 14, 1992. The authors were supported in part by ESPRIT II Basic Research Actions Program of the European Community under contract 3075 (project ALCOM).

[†]Department of Software (LSI), Universitat Politècnica de Catalunya. Pau Gargallo 5, 08028 Barcelona, Spain (gavalda@lsi.upc.es). This paper was prepared while the author was visiting the University of California at Santa Barbara, supported by National Science Foundation grant CCR-89-13584.

[‡]Department of Computer Science, Tokyo Institute of Technology, Meguro-ku, Ookayama 1-12-1, Tokyo 152, Japan (watanabe@cs.titech.ac.jp). This research was performed while this author was visiting the Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. This author was supported by Grant-in-Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Cooperative Research (A) 02302047 (1990).

[1]This class was denoted as $P_T(\text{SPARSE})$ in [8], [22]. We are following more recent notation [2].

Let L be any $\mathrm{R_T}$(SPARSE) set. A sparse set S to which L is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible is called a *sparse-set description* for L. We investigate the computational complexity of sparse-set descriptions for L. First, notice that L can be nonrecursive. (For example, there is clearly some sparse set that is not recursive, although all sparse sets are by definition in $\mathrm{R_T}$(SPARSE).) Then all sparse-set descriptions for such L should be nonrecursive, and so it does not make sense to discuss the *absolute* complexity of sparse-set descriptions for L. Thus we consider the *relative* complexity; that is, we investigate the complexity of sparse-set descriptions for L relative to L. Second, notice that one can easily construct a sparse-set description for L that is not recursive relative to L. That is, the problem of interest is the complexity of an easiest sparse-set description for L. We study this type of description complexity for sets in $\mathrm{R_T}$(SPARSE).

In order to state our main results, let us introduce some notions and notation. For any complexity class $\mathcal{C}$ and any set $L \in \mathrm{R_T}$(SPARSE), we say that L has a *$\mathcal{C}$-self-recognizable sparse-set description* if L is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some sparse set that is in $\mathcal{C}^L$. Define $\mathcal{C}$-SELF to be the class of $\mathrm{R_T}$(SPARSE) sets with a $\mathcal{C}$-self-recognizable sparse-set description. Intuitively, $\mathcal{C}$ represents an upper bound for the relative complexity of sparse-set descriptions.

There is a general upper bound for the complexity of sparse-set descriptions for $\mathrm{R_T}$(SPARSE) sets. By extending [21, Lem. 5.6], one can easily show that every set L in $\mathrm{R_T}$(SPARSE) has a sparse set description in $\Delta_3^{\mathrm{P},L}$ ($= \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}^L}}$). In other words, $\mathrm{R_T}(\mathrm{SPARSE}) \subseteq \Delta_3^{\mathrm{P}}$-SELF. (Indeed, $\mathrm{R_T}(\mathrm{SPARSE}) = \Delta_3^{\mathrm{P}}$-SELF since by definition we have $\Delta_3^{\mathrm{P}}\text{-SELF} \subseteq \mathrm{R_T}(\mathrm{SPARSE})$.) However, we do not know whether this is the optimal upper bound for some $\mathrm{R_T}$(SPARSE) set. In fact, no nontrivial lower bound has been known for any $\mathrm{R_T}$(SPARSE) set; for example, it has been asked [23] whether there exists an $\mathrm{R_T}$(SPARSE) set that is not in P-SELF. In our main results we construct such a set A, i.e., $A \in \mathrm{R_T}(\mathrm{SPARSE}) - \text{P-SELF}$, and also another set B in $\mathrm{R_T}(\mathrm{SPARSE}) - (\mathrm{NP} \cap \text{co-NP})$-SELF. Furthermore, we show that $A \in (\mathrm{NP} \cap \text{co-NP})$-SELF and $B \in$ NP-SELF. Thus our results yield a $\mathcal{C}$-SELF hierarchy in $\mathrm{R_T}$(SPARSE); that is, $\text{P-SELF} \subsetneq (\mathrm{NP} \cap \text{co-NP})\text{-SELF} \subsetneq \text{NP-SELF} \subseteq \Delta_3^{\mathrm{P}}\text{-SELF}$ ($= \mathrm{R_T}(\mathrm{SPARSE})$).

Note that every sparse set S has a trivial sparse-set description, namely, S itself, that is in P^S. Thus a set in $\mathrm{R_T}(\mathrm{SPARSE}) - \text{P-SELF}$ must be one that is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some sparse set but that looks quite different from sparse sets. Also note that although one can easily think of a pair of L and S such that S is sparse, $L \leq_{\mathrm{T}}^{\mathrm{P}} S$, and $S \notin \mathrm{P}^L$, this does not necessarily mean that $L \in \mathrm{R_T}(\mathrm{SPARSE}) - \text{P-SELF}$; L might have another sparse-set description in P^L. That is, a set L in $\mathrm{R_T}(\mathrm{SPARSE}) - \text{P-SELF}$ must be such that *no* sparse-set description for L belongs to P^L.

There are some topics closely related to our problem; our main results and our proof technique provide interesting observations in these topics.

Recall that a set is in $\mathrm{R_T}$(SPARSE) if and only if it has polynomial-size circuits. Furthermore, from the proof of this fact it is clear that there should be a relation between the complexity of sparse-set descriptions and that of finding polynomial-size circuits. Indeed, these two measures are closely related, but they are not exactly the same. (The difference is mainly that between recognition and generation; see §4.1 for more explanation.) Hence our main results yield similar (but not the same) lower-bound results for the problem of finding polynomial-size circuits. We show that the set A constructed in the first main result has no polynomial-size circuits that are polynomial-time generable relative to A and that the set B constructed in the second main result has no polynomial-size circuits that are NPSV generable relative to B.

In order to study the difference between various reduction types, Tang and Book [22] proposed investigating classes E_r(TALLY) and E_r(SPARSE), where for any reduction type

r, E_r(TALLY) (respectively, E_r(SPARSE)) is the class of sets that are equivalent to some tally set (respectively, some sparse set) under reducibility r. (A *tally set* is any subset of 0^*.) Although the structure of E_r(TALLY) classes was studied well in [22] and later in [3], many important questions concerning E_r(SPARSE) classes have been left open. Here we answer all such questions. It has been asked [23] whether P/poly has a set not in $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(SPARSE), which is the same as asking whether P-SELF $\subsetneq$ R_{T}(SPARSE); thus the first main result answers this question affirmatively. We also prove various separations between E_r(SPARSE) classes that have been asked in [22]. From our main results we immediately obtain $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(SPARSE) $\subsetneq$ $\mathrm{E}^{\mathrm{SN}}_{\mathrm{T}}$(SPARSE) $\subsetneq$ $\mathrm{E}^{\mathrm{NP}}_{\mathrm{T}}$(SPARSE). By applying the proof technique we used to obtain our main results, we show that $\mathrm{E}^{\mathrm{P}}_{\mathrm{tt}}$(SPARSE) $\subsetneq$ $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(SPARSE).

With similar motivations, Book and Ko [8] and Ko [15], [16] studied the inclusion relations between classes of sets *reducible* to sparse sets under several polynomial-time reducibilities r, $\mathrm{R}^{\mathrm{P}}_r$(SPARSE). Most of the inclusion relations between these classes were well determined, except for a few ones that were formulated as conjectures in [15]. We apply again the technique used to prove our main results and show that two of the conjectures in [15] are true. More precisely, we show that not every set $\leq^{\mathrm{P}}_{\mathrm{c}}$-reducible to a sparse set is $\leq^{\mathrm{P}}_{\mathrm{d}}$-reducible to a sparse set and therefore that $\mathrm{R}^{\mathrm{P}}_{\mathrm{d}}$(SPARSE) $\subsetneq$ R_{T}(SPARSE).

2. Preliminaries.

In this paper we follow standard definitions and notations in computational complexity theory (see [7], [21]).

We use the alphabet $\Sigma = \{0, 1\}$. By *string* or *word* we mean an element of Σ^*. Let x and y be any strings, and let X be any set of strings. We denote by $x \cdot y$ the concatenation of strings x and y, by $|x|$ the length of a string x, and by $\|X\|$ the cardinality of X. We use $\overline{X}$ and $X^{\leq n}$ to denote the complement of X, i.e., $\Sigma^* - X$, and the set $\{x \in X : |x| \leq n\}$, respectively.

A pairing (or tupling) function on a domain $\mathcal{D}$ is a one-to-one function from $\mathcal{D} \times \mathcal{D}$ (or $\mathcal{D} \times \cdots \times \mathcal{D}$) to $\mathcal{D}$. We use various pairing (or tupling) functions, depending on our purpose. Let $\langle \cdot, \cdot \rangle$ be a standard pairing function on Σ^*; we assume that this pairing function is polynomial-time computable and invertible. Let $\langle \cdot, \cdot \rangle_{\mathrm{N}}$ denote a pairing function from $\mathbf{N} \times \mathbf{N}$ *onto* $\mathbf{N}$; for example, we can define $\langle n, m \rangle_{\mathrm{N}} = (n + m)(n + m + 1)/2 + n$. Let $\langle \cdot, \cdot \rangle_{\mathrm{T}}$ denote a pairing function from $0^* \times 0^*$ onto 0^*, which can be defined, for example, as $\langle 0^n, 0^m \rangle_{\mathrm{T}} = 0^{\langle n,m \rangle_{\mathrm{N}}}$. We generalize these pairing functions to k-tupling functions by applying them $k - 1$ times; thus $\langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle$.

For any string x, a *self-delimiting code* of x, denoted as $\overline{x}$, is the string $u10x$, where u is obtained by doubling each bit of the binary representation of $|x|$; for example, $\overline{0} = 11 \cdot 10 \cdot 0$, $\overline{0110} = 110000 \cdot 10 \cdot 0110$, etc. Note that from $\overline{x} \cdot y$ one can compute both x and y in linear time. For any strings $x_1, \ldots, x_n$ we use $x_1\# \cdots \# x_n$ to denote the string $\overline{x_1} \cdots \overline{x_n}$. With our convention we have that $|x_1\# \cdots \# x_n| \leq |x_1| + \cdots + |x_n| + 2n(\lceil \log(\max\{|x_1|, \ldots, |x_n|\}) \rceil + 1)$. The quantity $2n(\lceil \log(\max\{|x_1|, \ldots, |x_n|\}) \rceil + 1)$ will be referred as a *tupling overhead.*

Our computation model is the standard multiple-tape (oracle) Turing machine. We consider deterministic and nondeterministic machines, all of which are either acceptors or transducers. Note that nondeterministic transducers compute, in general, multivalued functions. For any (oracle) machine M, the execution of M on an input x (relative to an oracle X) is denoted as $M(x)$ (respectively, $M^X(x)$). Let $L(M)$ (respectively, $L(M; X)$) denote the set of strings accepted by M (relative to X). For any complexity class $\mathcal{C}$ let $\mathcal{C}^X$ denote the complexity class $\mathcal{C}$ defined relative to X.

We use several polynomial-time reducibilities. For deterministic ones we consider $\leq^{\mathrm{P}}_{f(n)\text{-tt}}$, $\leq^{\mathrm{P}}_{\mathrm{c}}$, $\leq^{\mathrm{P}}_{\mathrm{d}}$, $\leq^{\mathrm{P}}_{\mathrm{tt}}$, and $\leq^{\mathrm{P}}_{\mathrm{T}}$. See [17] for the definitions of $\leq^{\mathrm{P}}_{\mathrm{tt}}$ and $\leq^{\mathrm{P}}_{\mathrm{T}}$. $\leq^{\mathrm{P}}_{\mathrm{c}}$ and $\leq^{\mathrm{P}}_{\mathrm{d}}$ are conjunctive and disjunctive polynomial-time reducibilities; again see [17]. Note that truth-table and Turing versions of, say, $\leq^{\mathrm{P}}_{\mathrm{c}}$ are equivalent in power. Therefore, we make no distinction between truth-table and Turing versions when talking about conjunctive or disjunctive reducibilities. $\leq^{\mathrm{P}}_{f(n)\text{-tt}}$ is a special case of $\leq^{\mathrm{P}}_{\mathrm{tt}}$-reducibility that is defined as follows: for any function $f(n)$, $X \leq^{\mathrm{P}}_{f(n)\text{-tt}} Y$ if X is reducible to Y via a $\leq^{\mathrm{P}}_{\mathrm{tt}}$-reduction that makes at most $f(n)$ queries on an input of length n. We also consider two nondeterministic polynomial-time reducibilities, $\leq^{\mathrm{SN}}_{\mathrm{T}}$ and $\leq^{\mathrm{NP}}_{\mathrm{T}}$. For any sets A and B, $A \leq^{\mathrm{SN}}_{\mathrm{T}} B$ (respectively, $A \leq^{\mathrm{NP}}_{\mathrm{T}} B$) if A is in $\mathrm{NP}^B \cap \text{co-NP}^B$ (respectively, NP^B).

In our discussions we use the notion of "resource-bounded Kolmogorov complexity." (See [18] for a survey on this notion and related topics.) Here we prepare some useful notation concerning this complexity measure.

We first fix one *universal Turing transducer* U. Roughly speaking, U is a deterministic transducer that expects as input a pair of a transducer T and a string x and simulates T on input x. More precisely, for every transducer T (whose program is encoded as d) and every string x, U on input $\overline{d} \cdot x$ executes in the following way:

(i) $U(\overline{d} \cdot x)$ halts and outputs y if and only if $T(x)$ halts and outputs y;

(ii) if $T(x)$ halts within t steps, then $U(\overline{d} \cdot x)$ halts within $|d|^2 \cdot t^2$ steps.

The existence of such a universal machine is shown in [11].

Now we define the following sets: For any string x and any $n, m \geq 0$, $\mathrm{KT}[n, m \mid x]$ is the set of strings that are generated by U from some input $y \cdot x$ within m steps, where $|y| \leq n$. Define $\mathrm{KT}[n, m]$ to be $\mathrm{KT}[n, m \mid \epsilon]$, where ϵ denotes the empty string. Notice that for each n, m, and x, $\mathrm{KT}[n, m \mid x]$ contains only finitely many strings.

3. Main results. In this section we construct two sets A and B in $\mathrm{R_T}$(SPARSE) such that A is not in P-SELF and B is not in (NP $\cap$ co-NP)-SELF. Furthermore, we show that A is in (NP $\cap$ co-NP)-SELF and B is in NP-SELF. In other words, A is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *some* sparse set in $\mathrm{NP}^A \cap \text{co-NP}^A$ but $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *no* sparse set in P^A, and B is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *some* sparse set in NP^B but $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *no* sparse set in $\mathrm{NP}^B \cap \text{co-NP}^B$. Therefore, we have that P-SELF $\subsetneq$ (NP $\cap$ co-NP)-SELF $\subsetneq$ NP-SELF.

THEOREM 3.1. *There is a set A in* $\mathrm{R_T}$(SPARSE) *that is not in* P-SELF.

Proof. We construct A that meets the following two requirements:

(I) For some sparse set S_A, $A \leq^{\mathrm{P}}_{\mathrm{T}} S_A$.

(II) For every sparse set S, if $A \leq^{\mathrm{P}}_{\mathrm{T}} S$, then $S \notin \mathrm{P}^A$.

Note that the second requirement is equivalent to the following: for every set X, if $A \leq^{\mathrm{P}}_{\mathrm{T}} X \leq^{\mathrm{P}}_{\mathrm{T}} A$, then X is not sparse.

We first explain the rough idea of our construction. The set A is built to be a nonsparse collection of Kolmogorov complex strings such that every string in A is hard to describe relative to any other string in A (Lemma 3.2; we can define such A so that it is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to some sparse set S_A). Using this property, we are able to show that every $\leq^{\mathrm{P}}_{\mathrm{T}}$ reduction from A to A must be very similar to the identity function (in a way to be made precise in Lemma 3.3). Now consider any set X that is $\leq^{\mathrm{P}}_{\mathrm{T}}$-equivalent to A, that is, $A \leq^{\mathrm{P}}_{\mathrm{T}} X \leq^{\mathrm{P}}_{\mathrm{T}} A$. Then the composition of the $\leq^{\mathrm{P}}_{\mathrm{T}}$ reduction from A to X and that from X to A is regarded as a $\leq^{\mathrm{P}}_{\mathrm{T}}$ reduction from A to A; thus such a reduction is very similar to the identity function. Hence X must have distinguished information about most strings in A (which is nonsparse) and thus cannot be sparse.

Now we define the set A precisely. Let $\{p_l\}_{l \geq 1}$ denote an enumeration of polynomials and $\{M_i\}_{i \geq 1}$ denote an enumeration of polynomial-time deterministic Turing reductions. We can assume without loss of generality that the running time of each M_i is bounded by p_i. Also, we say that a set X is p_l-sparse if $\|X^{\leq n}\| \leq p_l(n)$ for every n. In the construction we use two fixed functions $t(n)$ and $m(n)$. Function $t(n)$ must grow faster than any polynomial; choosing $t(n) = 2^n$ is enough for this proof. For $m(n)$ we can use any unbounded function that can be computed in time polynomial in n and for which $m(n) \cdot \log n = o(n)$.

We build our set A in stages. (Recall that the function $\lambda i, j, l.\langle i, j, l\rangle_{\mathrm{N}}$ is a one-to-one and onto function from $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$.) At each stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ we define $A^{(k)}$ so that one of the following conditions holds:

(a) $A^{(k)} \neq L(M_i; L(M_j; A^{(k)}))$.

(b) $L(M_j; A^{(k)})$ is not p_l-sparse.

Then define $A = \cup_{k\geq 1} A^{(k)}$. We also build $S^{(k)}$ at stage k and define $S_A = \cup_{k\geq 1} S^{(k)}$ so that $A \leq_{\mathrm{T}}^{\mathrm{P}} S_A$; thereby A satisfies requirement (I).

Note that for each stage k, conditions (a) and (b) have the following meaning: (a) guarantees that no X exists such that $A^{(k)} \leq_{\mathrm{T}}^{\mathrm{P}} X \leq_{\mathrm{T}}^{\mathrm{P}} A^{(k)}$ via M_i and M_j, and (b) guarantees that if $X \leq_{\mathrm{T}}^{\mathrm{P}} A^{(k)}$ via M_j, then X is not p_l-sparse. Furthermore, we will construct $A^{(k)}$ in such a way that the construction does not affect the previously established conditions. Thus the defined set A satisfies requirement (II).

What follows is how we define the sets $A^{(k)}$ and $S^{(k)}$ at stage $k = \langle i, j, l\rangle_{\mathrm{N}}$.

(1) Choose $c > 0$ that is larger than the constants and the size of programs that appear during the following construction. These constants and program sizes depend only on M_i, M_j, and p_l; thus c can be determined at this point of stage k.

(2) Define $n > 0$ to be the smallest integer such that

(i) $c \log n + (m(n) + 1)\log(2n) + 2c < n/2$;

(ii) $2(5 + m(n))\log 2n^2 < n/4$;

(iii) $c \cdot (t(n) + 2n^2)^2 < t(2n^2)$;

(iv) $\binom{n}{m(n)} > p_l(p_i(n \cdot m(n)))$;

(v) $n^c + c < t(n)$;

(vi) n is sufficiently large that no string of length $\geq n$ is used or queried up to stage $k-1$, and this construction up to stage $k-1$ can be done in time n.

With the conditions placed on $m(n)$, clearly we can choose such n.

(3) Let τ be the lexicographically first string of length $2n^2$ such that $\tau \notin \mathrm{KT}[2n^2 - 1, t(2n^2)]$. Divide τ into $2n$ blocks of n bits, $w_1, \ldots, w_{2n}$, and define $W_1 = \{w_1, \ldots, w_n\}$ and $W_2 = \{w_{n+1}, \ldots, w_{2n}\}$. Let m stand for the value of $m(n)$ during this stage. Then define L_1 and L_2 as follows:

$$L_1 = \{\ w_{i_1} \cdots w_{i_m} : \text{ all } w_{i_j} \text{ are different, sorted, and in } W_1\ \},$$

$$L_2 = \{\ w_{i_1} \cdots w_{i_m} : \text{ all } w_{i_j} \text{ are different, sorted, and in } W_2\ \},$$

(where "sorted" means in lexicographical order). It is easy to show that all blocks $w_1, \ldots, w_{2n}$ must be distinct unless we contradict the definition of τ. Hence L_1 and L_2 are disjoint, and each of them contains $\binom{n}{m}$ strings of length $n \cdot m$. We should notice that L_1 (respectively, L_2) is reducible to W_1 (respectively, W_2) by a simple and uniform reduction that makes m queries; that is, to decide whether a given string $u_1 \cdots u_m$ is in L_1 (where all u_i are of length n, different, and sorted), it is enough to check whether every u_i is in W_1.

(4) Finally, define $A^{(k)}$ and $S^{(k)}$ as follows. Let $A_1 = A^{(k-1)} \cup L_1$ and $A_2 = A^{(k-1)} \cup L_2$.

Case (a). Either (i) $A_1 \neq L(M_i; L(M_j; A_1))^{\leq nm}$ or (ii) $A_2 \neq L(M_i; L(M_j; A_2))^{\leq nm}$ occurs. In this case, if (i) occurs, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

Case (b). Either (i) $L(M_j; A_1)^{\leq p_i(nm)}$ is not p_l-sparse or (ii) $L(M_j; A_2)^{\leq p_i(nm)}$ is not p_l-sparse. In this case, if (i) occurs, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

If this construction is completed, then the defined sets A $(= \cup_{k\geq 1} A^{(k)})$ and S_A $(= \cup_{k\geq 1} S^{(k)})$ clearly satisfy the requirements. Thus it suffices to show that the construction can be completed, which is proved in the following lemmas. □

We show that the construction can be completed. For this purpose it is enough to show that either Case (a) or Case (b) holds at every stage; in other words, at every stage, if Case (a) fails, then Case (b) certainly holds. We will prove this in Lemma 3.4.

The following lemmas refer to any stage k of the construction and use all the symbols defined in that stage. For a word $x = w_{i_1} \cdots w_{i_m}$ in $L_1 \cup L_2$, we say that the words $w_{i_1}, \ldots, w_{i_m}$ are the *blocks* of x.

We first show the following property.

LEMMA 3.2. *For every x and y in $L_1 \cup L_2$, if $x \neq y$, then $y \notin \mathrm{KT}[c \log n + c, t(n) \mid x]$.*

Proof. For the sake of contradiction, assume that there is a description d_1 of size at most $c \log n + c$ such that our universal Turing machine U generates y from $d_1 \cdot x$ in time $t(n)$. We will show that this assumption implies $\tau \in \mathrm{KT}[2n^2 - 1, t(2n^2)]$, contradicting the choice of τ.

Let w be the first block in y that is not a block in x, and let $\tau - x - w$ be the string that is obtained by deleting w and all blocks of x from the word τ. Also, let ℓ be the list $\ell_0\#\ell_1\# \cdots \#\ell_m$ such that

—w is the ℓ_0th block in τ;

—for every $i \in \{1, \ldots, m\}$ the ith block in x is the ℓ_ith block in τ.

That is, ℓ gives the information needed to insert w and x again into $\tau - x - w$.

Finally, let d_0 be the description of some program such that U generates τ from $\overline{d_0} \cdot (d_1\#x\#\ell\#(\tau - x - w))$. Intuitively, d_0, on input $d_1\#x\#\ell\#(\tau - x - w)$, computes y from $d_1 \cdot x$, compares x and y in order to extract w, and inserts x and w into $\tau - x - w$ from the information in ℓ, thereby obtaining τ. Note that such a program can be chosen independently of n, τ, x, y, and d_1; thus we can regard $|d_0|$ as a fixed constant. Furthermore, the program can do this task in time $\leq b \cdot (t(n) + 2n^2)$ for some constant b that depends on d_0 only. Hence U with $\overline{d_0} \cdot (d_1\#x\#\ell\#(\tau - x - w))$ as input halts within $|d_0|^2 \cdot b^2 \cdot (t(n) + 2n^2)^2$ steps. Choose $c \geq |d_0|^2 \cdot b^2$. By condition (iii) on n this is less than $t(2n^2)$.

We now estimate the size of $\overline{d_0} \cdot (d_1\#x\#\ell\#(\tau - x - w)) = d_0\#d_1\#x\#\ell\#(\tau - x - w)$. Note first that the size of each ℓ_i is at most $\log(2n)$. Let e be $|d_0| + |d_1| + |x| + |\ell_0| + \cdots + |\ell_m| + |\tau - x - w|$. Adding up the size of each string, we have that $e < c + (c \log n + c) + mn + (m+1)\log(2n) + (2n^2 - mn - n)$; hence, by condition (i) on n, $e < 2n^2 - n/2$. The tupling overhead for $d_0\#d_1\#x\#\ell\#(\tau - x - w)$ is smaller than $2(5+m)(\log e + 1) \leq 2(5+m)\log 2n^2 < n/4$ (by condition (ii)). Thus the total size of this string is less than $2n^2 - n/2 + n/4 < 2n^2$, giving the desired contradiction. □

Now we define a new oracle machine M_s by composing machines M_i and M_j. On input x ($|x| \leq n \cdot m$) and with oracle Y, M_s executes as follows (if $|x| > n \cdot m$, then M_s rejects x): (1) It first builds a table t that encodes $A^{(k-1)}$. (2) Then M_s simulates $M_i(x)$, where each query y asked by $M_i(x)$ is solved by simulating $M_j(y)$, and each query asked by $M_j(y)$ is solved by asking oracle $t \cup Y$. (3) M_s accepts x if and only if M_i accepts x.

By recalling condition (vi) on n, it is clear that there exists some polynomial p_s (depending on M_i and M_j only) that bounds the running time of M_s. Furthermore, if Case (a) does not hold, we have that $L(M_s; A_1)^{\leq nm} = A_1$ and $L(M_s; A_2)^{\leq nm} = A_2$. The next lemma says that, in the performance of these reductions M_s can expect very little help from oracles A_1 and A_2.

LEMMA 3.3. *For all strings x and y in $L_1 \cup L_2$, if either $M_s^{A_1}(x)$ or $M_s^{A_2}(x)$ query y, then $x = y$.*

Proof. We will proceed by contradiction. Suppose for example that x is in L_1 and that the first $\ell - 1$ queries of $M_s^{A_1}(x)$ are either x or not in $L_1 \cup L_2$ but that the ℓth one is a word $y \in L_1 \cup L_2$ such that $y \neq x$.

Define d_2 as the description of a program that, being given $\overline{\ell} \cdot x$ as an input, simulates $M_s(x)$ with oracle $\{x\}$ and prints its ℓth query. Then, on input $\overline{d_2} \cdot (\overline{\ell} \cdot x)$ $(= (d_2\#\ell) \cdot x)$, our universal machine prints a word in $L_1 \cup L_2$ other than x.

Since the program d_2 depends only on M_s, we can regard $|d_2|$ as a constant in this stage. Because M_s is p_s-time-bounded, there must be a constant c_1, depending on M_i and M_j only, such that $|\ell| \leq c_1 \cdot \log n$. If we choose $c \geq 3 \cdot (|d_2| + c_1)$, some routine calculations show that $|d_2\#\ell| \leq c \cdot \log n + c$. Furthermore, there must be another constant c_2, depending on M_i, M_j, and $|d_2|$ only, such that U on input $\overline{d_2} \cdot (\overline{\ell} \cdot x)$ runs in time $n^{c_2} + c_2$. If we choose $c \geq c_2$, this is less than $t(n)$ by condition (v) on n. This contradicts Lemma 3.2.

If $x \in L_2$ instead, we use oracle $\emptyset$ in the simulation of M_s. The other cases are symmetric. $\square$

Lemma 3.3 roughly states that the composition of reductions M_i and M_j must have a very simple structure. We can use this to show that intermediate sets in these reductions cannot be sparse.

LEMMA 3.4. *If Case* (a) *fails, then Case* (b) *holds.*

Proof. Suppose that Case (a) fails, and let X_1 and X_2 be two sets such that $A_1 \leq_T^P X_1 \leq_T^P A_1$ and $A_2 \leq_T^P X_2 \leq_T^P A_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)}$ and $X_2^{\leq p_i(nm)}$ is not p_l-sparse.

Define function C as follows. For any $x \in L_1 \cup L_2$, $C(x)$ is the first query made by $M_i^{X_1}(x)$ that has different answers from oracles X_1 and X_2. Note that $C(x)$ is always defined for every x in $L_1 \cup L_2$. Otherwise, $M_i(x)$ gives the same answer with oracles X_1 and X_2; but this answer cannot be NO because x is in one of L_1 and L_2, and it cannot be YES because L_1 and L_2 are disjoint.

Since $C(x)$ is always an element of either X_1 or X_2, we can regard C as a mapping from $L_1 \cup L_2$ into $X_1 \cup X_2$. Furthermore, we show in the following that C yields a different element in $X_1 \cup X_2$ for each one in $L_1 \cup L_2$.

CLAIM. *The function C is one-to-one on the domain* $L_1 \cup L_2$.

Proof of the claim. Take any $x \in L_1 \cup L_2$, and name $y = C(x)$. Let us show first the following fact: both $M_j^{A_1}(y)$ and $M_j^{A_2}(y)$ query x.

We know that y must have different answers with respect to oracles X_1 and X_2. With our assumptions, this means that $M_j(y)$ must answer differently with oracles A_1 and A_2. But these oracles differ only in the words in L_1 and L_2, and neither $M_j^{A_1}(y)$ nor $M_j^{A_2}(y)$ can query any of these words except for x; otherwise, so does $M_s(x)$ and we have a contradiction to Lemma 3.3. Therefore, to give a different answer both $M_j^{A_1}(y)$ and $M_j^{A_2}(y)$ must query x.

To prove that C is one-to-one, assume, for example, that also $C(x') = y$ and that $M_i^{X_1}(x')$ queries y, for some $x' \in L_1 \cup L_2$ different from x. We have now that $M_s^{A_1}(x')$ queries x when simulating $M_j(y)$. This again contradicts Lemma 3.3. This proves the claim. $\square$

Define now $C_1 = \{ x \in L_1 \cup L_2 : C(x) \in X_1 \}$ and $C_2 = \{ x \in L_1 \cup L_2 : C(x) \in X_2 \}$. Since C is one-to-one on $L_1 \cup L_2$, we have

$$\|X_1^{\leq p_i(nm)}\| + \|X_2^{\leq p_i(nm)}\| \geq \|C_1\| + \|C_2\| = \|C_1 \cup C_2\|.$$

But C_1 and C_2 cover all of $L_1 \cup L_2$, so that

$$\|C_1 \cup C_2\| = \|L_1 \cup L_2\| = 2\binom{n}{m} > 2p_l(p_i(nm))$$

(by condition (iv) on n).

Thus $\|X_1^{\leq p_i(nm)}\| + \|X_2^{\leq p_i(nm)}\| > 2 \cdot p_l(p_i(nm))$. Therefore, at least one of $\|X_1^{\leq p_i(nm)}\|$ and $\|X_2^{\leq p_i(nm)}\|$ must be greater than $p_l(p_i(nm))$, and hence either $X_1^{\leq p_i(nm)}$ or $X_2^{\leq p_i(nm)}$ is not p_l-sparse. This proves Lemma 3.4. $\square$

From Theorem 3.1 we know that A has no sparse-set description that is P^A-recognizable. Here we give an upper bound; that is, we show that A has a sparse-set description in $\mathrm{NP}^A \cap$ co-NP^A.

Indeed, the sparse set S_A constructed in Theorem 3.1 is in $\mathrm{NP}^A \cap \text{co-NP}^A$. It is easy to see that S_A is in NP^A by considering the following procedure. To check that a word w is in S_A, guess a word x that has w as a block and verify that $x \in A$. On the other hand, because of the fact that S_A has an easy census function, we can prove that $S_A \in \text{co-NP}^A$.

For any set L the following function cens_L is called the *census function* of L: For each $n \geq 0$, $\mathrm{cens}_L(0^n) = \|L^{\leq n}\|$. Note that S_A has n elements at all the lengths n used in its construction and is empty at all others. Together with the following proposition, this shows that $S_A \in \mathrm{NP}^A \cap \text{co-NP}^A$.

PROPOSITION 3.5. *For every pair of sets X and Y, if X is sparse and in* NP^Y *and its census function is polynomial-time computable, then X is in* $\mathrm{NP}^Y \cap \text{co-NP}^Y$.

Proof. It is enough to show that $\overline{X} \in \mathrm{NP}^Y$. Since $X \in \mathrm{NP}^Y$, there is a polynomial-time nondeterministic machine M that recognizes X relative to Y. For a given u we can check that u is not in X by the following procedure:

(1) Compute $c = \mathrm{cens}_X(0^{|u|})$.

(2) Guess c different words v such that $v \neq u$, $|v| \leq |u|$, and $v \in X$. (The last condition is verified by M relative to Y.)

(3) Conclude that u is not in X if and only if c different words are found (and verified) at (2). Clearly, this procedure witnesses that $\overline{X} \in \mathrm{NP}^Y$. □

COROLLARY 3.6. *The set A defined in Theorem* 3.1 *is in* (NP ∩ co-NP)-SELF − P-SELF.

Next we construct a set having only sparse-set descriptions of higher complexity; we will show a set B that is in $\mathrm{R_T}$(SPARSE) but not in (NP ∩ co-NP)-SELF. To simplify the following discussion, let us introduce the following notions.

A *three-output machine* is a nondeterministic Turing machine with three final states: ACC, REJ, and ? (standing for "accept," "reject," and "don't know"). Let M be any three-output oracle machine, and let Y be any oracle set. M is called *strong under oracle Y* if for all x, $M^Y(x)$ has either at least one computation path ending in ACC or at least one ending in REJ, but not both. When M is strong under oracle Y, we define $L(M; Y)$ to be the set of strings x for which $M^Y(x)$ has at least one computation path ending in ACC. $L(M; Y)$ can be regarded as the set recognized by M relative to Y.

These are taken from the original definition of $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reducibility in [19], where the following fact is proved.

PROPOSITION 3.7. *For any sets X and Y,* $X \leq_{\mathrm{T}}^{\mathrm{SN}} Y$ *if and only if there is a polynomial-time three-output machine M that is strong under oracle Y and that recognizes X relative to Y, i.e.,* $X = L(M; Y)$.

Our second main result is as follows.

THEOREM 3.8. *There is a set B in* $\mathrm{R_T}$(SPARSE) *such that for every sparse set S, if* $B \in \mathrm{NP}^S \cap \text{co-NP}^S$, *then* $S \notin \mathrm{NP}^B \cap \text{co-NP}^B$. *Thus B is not in* (NP ∩ co-NP)-SELF.

Remark. For proving $B \notin$ (NP ∩ co-NP)-SELF, it suffices to show that for any sparse set S, if $B \leq_{\mathrm{T}}^{\mathrm{P}} S$, then $S \notin \mathrm{NP}^B \cap \text{co-NP}^B$. The theorem is slightly stronger than what we need for the moment. We will use this stronger version in §4.2.

Proof. We construct B that meets the following requirements:

(I) For some sparse set S_B, $B \leq_{\mathrm{T}}^{\mathrm{P}} S_B$.

(II) For every set X, if $B \leq_{\mathrm{T}}^{\mathrm{SN}} X \leq_{\mathrm{T}}^{\mathrm{SN}} B$, then X is not sparse.

The construction is quite similar to that of Theorem 3.1, but now we must attack nondeterministic machines that may have many computation paths. Note that our construction technique does not seem to yield a set L in $\mathrm{R_T}$(SPARSE) − NP-SELF; that is, a set L constructed by our technique always has a NP^L-recognizable sparse-set description. Then, from Proposition 3.5, if such a sparse-set description has an easy census function, its complexity goes down to $\mathrm{NP}^L \cap \text{co-NP}^L$. Thus the key to constructing the desired set B is to make sure that it is reducible to no sparse set with an easy census function.

In order to explain how we use this point, let us consider any pair of machines, that may reduce $B \leq_{\mathrm{T}}^{\mathrm{SN}} X \leq_{\mathrm{T}}^{\mathrm{SN}} B$ for some set X. Similarly to the previous proof, define M_s to be a machine obtained by composing these two machines, i.e., M_s is a $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reduction from B to B. The requirement we consider first is that M_s must be strong under B. But during the construction at stage k we can try M_s with various extensions of $B^{(k-1)}$ that have different census values. If M_s fails to be strong under some extension, we can define the set $B^{(k)}$ so that the pair of machines does not make a correct $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reduction from B to B. Otherwise, M_s is strong under any extension; furthermore, the extension $B' = B^{(k)}$ is such that $M_s^{B'}$ always has a path ending in REJ state for every sufficiently large input. We choose one such path for every input, thereby fixing a nondeterministic path of M_s. Then we regard M_s (on such a path) as a deterministic machine, and, using an argument similar to the previous proof, we can prove that the intermediate set X must be nonsparse.

To define the sets B and S_B, let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial-time nondeterministic three-output machines. We can assume that the running time of each M_i is bounded by p_i. Choose $m(n)$ as before, but choose the time bound $t(n) = 2^{2^n}$, much larger than before.

We build our set B in stages. At each stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ we define $B^{(k)}$ so that one of the following conditions holds:

(a) Either M_j is not strong under oracle $B^{(k)}$ or M_i is not strong under oracle $L(M_j;\, B^{(k)})$.

(b) $B^{(k)} \neq L(M_i;\, L(M_j;\, B^{(k)}))$.

(c) $L(M_j;\, B^{(k)})$ is not p_l-sparse.

Then define $B = \cup_{k\geq 1} B^{(k)}$. We also build $S^{(k)}$ at stage k and define $S_B = \cup_{k\geq 1} S^{(k)}$ so that $B \leq_{\mathrm{T}}^{\mathrm{P}} S_B$; thereby, B satisfies requirement (I). Also, if we always achieve (a), (b), or (c), then B clearly satisfies requirement (II).

To define $B^{(k)}$ and $S^{(k)}$ at stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ we follow steps (1)–(3) in the proof of Theorem 3.1; that is, choose c, n, m, τ, and define W_1, W_2, L_1, and L_2. Then do step (4) below.

(4) Define $B^{(k)}$ and $S^{(k)}$ as follows. Let $B_1 = B^{(k-1)} \cup L_1$ and $B_2 = B^{(k-1)} \cup L_2$.

Case (a). One of the following holds: (i) M_j is not strong under $B^{(k-1)}$ on some input of length at most $p_i(nm)$, (ii) M_i is not strong under $L(M_j;\, B^{(k-1)})$ on some input of length at most nm, or (iii) $L(M_i;\, L(M_j;\, B^{(k-1)}))^{\leq nm} \neq B^{(k-1)}$. Then set $B^{(k)}$ to $B^{(k-1)}$ and $S^{(k)}$ to $\emptyset$.

Case (b). Either (i) $L(M_i;\, L(M_j;\, B_1))^{\leq nm} \neq B_1$ or (ii) $L(M_i;\, L(M_j;\, B_2))^{\leq nm} \neq B_2$.[2] In this case, if (i) occurs, then set $B^{(k)}$ to B_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $B^{(k)}$ to B_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

Case (c) Either (i) $L(M_j;\, B_1)^{\leq p_i(nm)}$ is not p_l-sparse or (ii) $L(M_j;\, B_2)^{\leq p_i(nm)}$ is not p_l-sparse. In this case, if (i) occurs, then set $B^{(k)}$ to B_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $B^{(k)}$ to B_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

If this construction is completed, then the defined sets B $(= \cup_{k\geq 1} B^{(k)})$ and S_B $(= \cup_{k\geq 1} S^{(k)})$ clearly satisfy the requirements. The following lemmas show that it can always be completed. □

Consider again any stage k. Lemma 3.2 is still true here. M_s is also defined similarly by using M_i and M_j. Now, for each word x of length $n \cdot m$, define r_x to be the lexicographically least computation path of $M_s(x)$ relative to $\emptyset$ that ends in state REJ (if such a path exists).

It is important to notice that if r_x exists, it can be found recursively from x by brute-force search of the computation tree of $M_s(x)$. More precisely, r_x can be found in time $2^{p(n)}$ for some polynomial p that depends on M_i and M_j only. The following property of r_x is crucial in our proof.

LEMMA 3.9. *If Case* (a) *does not hold, then r_x exists for every $x \in L_1 \cup L_2$.*

[2]Case (b) also holds if any of the computations involved fails to be strong, as in Case (a). We omit the details.

Proof. If Case (a) does not hold, then $M_i(x)$ is strong under $L(M_j;\ B^{(k-1)})$ because x has length nm. Furthermore, $M_j(z)$ is strong under $B^{(k-1)}$ for every possible query z of $M_i(x)$ because z must have length at most $p_i(nm)$. Suppose that no path of $M_s^{\emptyset}(x)$ ends in state REJ. Because M_s is the composition of M_i and M_j and the composition of strong computations is strong, some path of $M_s^{\emptyset}(x)$ must end in ACC. That is, M_i accepts x with oracle $L(M_j;\ B^{(k-1)})$. But this means that $L(M_i;\ L(M_j;\ B^{(k-1)}))^{\leq nm} \neq B^{(k-1)}$ because $x \notin B^{(k-1)}$, a contradiction. □

The next lemma corresponds to Lemma 3.3.

LEMMA 3.10. *Assume that Case* (a) *does not hold. For all strings x and y in $L_1 \cup L_2$, if y is queried in r_x, then $x = y$.*

Proof. If Case (a) does not hold, then, by Lemma 3.9, r_x exists for every $x \in L_1 \cup L_2$ and, moreover, can be found in time smaller than $t(n)$. Hence one can easily modify the proof of Lemma 3.3; the detail is omitted. □

The following lemma is the counterpart of Lemma 3.4. Its proof is almost the same when only the path r_x for every $x \in L_1 \cup L_2$ is considered.

LEMMA 3.11. *If both Case* (a) *and Case* (b) *fail, then Case* (c) *holds.*

Proof. Suppose that Case (a) and Case (b) fail, and let X_1 and X_2 be two sets such that $B_1 \leq_{\mathrm{T}}^{\mathrm{SN}} X_1 \leq_{\mathrm{T}}^{\mathrm{SN}} B_1$ and $B_2 \leq_{\mathrm{T}}^{\mathrm{SN}} X_2 \leq_{\mathrm{T}}^{\mathrm{SN}} B_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)}$ and $X_2^{\leq p_i(nm)}$ is not p_l-sparse.

For any $x \in L_1 \cup L_2$, let t_x be the computation path of $M_i(x)$ that $M_s^{\emptyset}(x)$ is simulating along path r_x. It is clear that t_x is uniquely determined by r_x. Now define function C as follows. For each x in $L_1 \cup L_2$, $C(x)$ is the first query along t_x that has different answers from oracles X_1 and X_2. In a way similar to Lemma 3.4 we can show the following:

(i) function C is defined on $L_1 \cup L_2$;

(ii) it is one-to-one on $L_1 \cup L_2$.

To prove (i) assume that $x \in L_1$. Because L_1 and L_2 are disjoint, $x \notin L_2$. By Lemma 3.10 no query along r_x is in L_2, and so all queries in r_x have the same answer from oracles $\emptyset$ and L_2. Hence t_x is a correct rejecting path of $M_i(x)$ relative to X_2. On the other hand, because $x \in L_1$, $M_i(x)$ has no rejecting path relative to X_1, some query in t_x must have different answers with oracles X_1 and X_2.

Part (ii) is proved as was the claim in Lemma 3.4, by repeatedly using Lemma 3.10. Then the same counting argument of Lemma 3.4 shows that at least one of $\|X_1^{\leq p_i(nm)}\|$ and $\|X_2^{\leq p_i(nm)}\|$ must be greater than $p_l(p_i(nm))$. □

The separation of NP-SELF from (NP ∩ co-NP)-SELF follows easily.

COROLLARY 3.12. *The set B defined in Theorem* 3.8 *is in* NP-SELF − (NP∩co-NP)-SELF.

Proof. Recall the algorithm that is given before Proposition 3.5 for showing $S_A \in \mathrm{NP}^A$. Clearly, a similar idea proves that $S_B \in \mathrm{NP}^B$. Thus S_B witnesses that B is in NP-SELF. □

4. Related topics.

This section presents some consequences of our results and applications of our proof techniques to other, but closely related, topics.

4.1. Complexity of finding polynomial-size circuits.

In §3 we investigated the problem of recognizing sparse-set descriptions. Recall that sets having sparse-set descriptions are exactly those accepted by polynomial-size circuits. (Because the term "polynomial-size" is always assumed in the following discussion, it is often omitted.) However, the relative complexity of a sparse set S such that $L \leq_{\mathrm{T}}^{\mathrm{P}} S$ is not exactly representing the relative complexity of finding circuits for L. Here we will first make such difference clear and then use the results from §3 to show some lower bounds for the complexity of finding circuits.

First we review the notion of "having polynomial-size circuits." For any set L we say that L has *polynomial-size circuits* if for some polynomial p and for every $n \geq 0$ there exists a boolean circuit consisting of $p(n)$ (or fewer) gates that determines whether $x \in L$ for every $x \in \Sigma^n$. Notice that this definition does not guarantee that a set L with polynomial-size circuits is recognized by some single machinery; we just know that L is recognized by a collection of circuits. Thus the class of sets with polynomial-size circuits does not fit in conventional complexity classes. Karp and Lipton [13] introduced a general framework for such nonuniform complexity classes, which is useful for studying the class of sets having polynomial-size circuits. By using one of their nonuniform complexity classes — namely, P/poly — one can completely characterize the class of sets with polynomial-size circuits. (This fact is essentially proved by Pippenger [20]. A good and complete proof can be found in [7].) Thus we can consider that P/poly is identical to the class of sets with polynomial-size circuits.

Let us review the definition of P/poly. Define a class of functions, poly, by

$$\text{poly} = \{ g : 0^* \to \Sigma^* \ : \ \exists p : \text{polynomial}, \forall n \geq 0 \, [\, |g(0^n)| \leq p(n) \,] \}.$$

Then the class P/poly is defined as follows.

$$\begin{aligned} L \in \text{P/poly} \ \leftrightarrow \quad & \text{there exist } g \in \text{poly and } E \in \text{P such that} \\ & \forall n \geq 0, \forall x \in \Sigma^n \, [\, x \in L \ \leftrightarrow \ \langle g(0^n), x \rangle \in E \,]. \end{aligned}$$

From the proof showing the equivalence between P/poly and the class of sets with polynomial-size circuits, one can easily fix some set E_0 so that for every L a function g satisfying the preceding with E_0 is regarded as a *circuit generator*; that is, $g(0^n)$ denotes the circuit for $L^{=n}$ and the computation of E_0 on $\langle g(0^n), d_1 \cdots d_n \rangle$ (where $d_i \in \{0, 1\}$) is the evaluation of the circuit (denoted as) $g(0^n)$ with $d_1, \ldots, d_n$ on its input gates.

Now we are ready to discuss the complexity of finding circuits formally. For any $L \in$ P/poly we regard the relative complexity of a circuit generator for L as the complexity of finding circuits for L. For example, suppose that some set $L \in$ P/poly has a generator g that is polynomial-time computable relative to L, i.e., $g \in \mathrm{PF}^L$. Then we consider that L has circuits that are easy to find, i.e., that are polynomial-time computable, relative to L. On the other hand, a set $H \in$ P/poly such that no circuit generator for H is in $\mathrm{PF}^{\mathrm{NP}^H}$ is considered as a set with no easy-to-find circuits. Let us introduce the following notion: For any complexity class $\mathcal{CF}$ of functions and any set $L \in$ P/poly, we say that a set L has *$\mathcal{CF}$-self-generable circuits* if there is a circuit generator for L that is in $\mathcal{CF}^L$. The class $\mathcal{CF}$-SELF$_{\text{gen}}$ is defined to be the class of sets with $\mathcal{CF}$-self-generable circuits. Ko [14] introduced the notion of "having self-p-producible circuits," which is the same as "having PF-self-generable circuits." Thus ours is a generalization of his notion.

It is well known (see, e.g., [7]) that P/poly equals $\mathrm{R_T}$(SPARSE), that is, L has polynomial-size circuits if and only if L is reducible to some sparse set. Thus the complexity of finding circuits for L seems to correspond to the complexity of a sparse-set description for L. Indeed, the two notions "having a $\mathcal{C}$-self-recognizable sparse-set description" and "having $\mathcal{CF}$-self-generable circuits" are closely related (where $\mathcal{CF}$ is a function class corresponding to $\mathcal{C}$). However, they are not the same; in other words, classes $\mathcal{C}$-SELF and $\mathcal{CF}$-SELF$_{\text{gen}}$ are not the same in general. This difference is mainly the one between recognition and generation, which is often seen in computational complexity theory. Usually recognition is easier than generation. For example, whereas every PF-printable set [12] is clearly P-recognizable (and sparse), we do not know whether every sparse set in P is PF-printable. (As a matter of fact, there is some evidence for the existence of a sparse set in P that is not PF-printable [3].) The best-known relation is that every sparse set in P is $\mathrm{PF}^{\mathrm{NP}}$-printable. Here we have similar relationships.

In order to discuss the relation between $\mathcal{C}$-SELF and $\mathcal{CF}$-SELF$_{\text{gen}}$ classes, we introduce one more class that is located between $\mathcal{C}$-SELF and $\mathcal{CF}$-SELF$_{\text{gen}}$:

$$\mathcal{C}\text{-TALLY-SELF} = \{ \ L \ : \ \exists T \text{ tally } [L \leq_{\mathrm{T}}^{\mathrm{P}} T \ \wedge \ T \in \mathcal{C}^{L}] \ \}.$$

The relationship among $\mathcal{C}$-SELF, $\mathcal{C}$-TALLY-SELF, and $\mathcal{CF}$-SELF$_{\text{gen}}$ is summarized as follows.

PROPOSITION 4.1. *Let $\mathcal{C}$ be any standard complexity class of languages, and let $\mathcal{CF}$ be the corresponding complexity class of functions.*

(1) $\mathcal{CF}$-SELF$_{\text{gen}}$ $\subseteq$ $\mathcal{C}$-TALLY-SELF $\subseteq$ $\mathcal{C}$-SELF.

(2) $\mathcal{C}$-TALLY-SELF $\subseteq$ PF$^{\mathcal{C}}$-SELF$_{\text{gen}}$.

(3) $\mathcal{C}$-SELF $\subseteq$ NP$^{\mathcal{C}}$-TALLY-SELF.

(4) *If $\mathcal{C}$ is closed under $\leq_{\mathrm{c}}^{\mathrm{NP}}$ reducibility, then* $\mathcal{C}$-SELF $=$ $\mathcal{C}$-TALLY-SELF.

Remark. (i) For any class $\mathcal{C}$ containing Δ_3^{P}, $\mathcal{CF}$-SELF$_{\text{gen}}$ $=$ $\mathcal{C}$-TALLY-SELF $=$ $\mathcal{C}$-SELF (see part (1) of the proposition). This is because P/poly $\subseteq$ Δ_3^{P}-SELF$_{\text{gen}}$ [21]. However, the analog relation is not known for lower complexity classes.

(ii) From parts (2) and (3) we have $\mathcal{C}$-SELF $\subseteq$ PF$^{\mathrm{NP}^{\mathcal{C}}}$-SELF$_{\text{gen}}$.

(iii) From part (4) we have Σ_k^{P}-SELF $=$ Σ_k^{P}-TALLY-SELF for any $k \geq 1$.

In the following we state only the proof of part (3) of the proposition and omit the other proofs. (Inclusions in parts (1) and (2) are either immediate from the definition or easy to prove, and part (4) is proved similarly to part (3).)

Proof of part (3). Let L be any set in $\mathcal{C}$-SELF. Thus there exists some sparse set S such that $L \leq_{\mathrm{T}}^{\mathrm{P}} S$ and $S \in \mathcal{C}^{L}$.

Define a tally set T_S as follows:

$$\begin{aligned} T_S \ = \ \{ \langle 0^n, 0^m, 0^i, \ 0^j, 0^b \rangle_{\mathrm{T}} \ &: \ n, m, i, j \geq 0, \ b \in \{0, 1\}, \\ &\exists s_1 < \cdots < s_m \in S^{\leq n} \ [\text{ the } i\text{th bit of } s_j \text{ is } b \ ] \ \}. \end{aligned}$$

Then it is easy to see that $S^{\leq n}$ can be enumerated in polynomial time by using T_S as an oracle. (Without loss of generality, we are assuming that S does not contain the empty string.) Thus $L \leq_{\mathrm{T}}^{\mathrm{P}} T_S$. On the other hand, clearly $T_S \in \mathrm{NP}^{S}$; thus $T_S \in \mathrm{NP}^{\mathcal{C}^{L}}$. □

Among many relations generated from Proposition 4.1 we are interested, in particular, in the following ones.

COROLLARY 4.2.

(1) PF-SELF$_{\text{gen}}$ $=$ P-TALLY-SELF $\subseteq$ P-SELF.

(2) PF$^{\mathrm{NP}}$-SELF$_{\text{gen}}$ $=$ Δ_2^{P}-TALLY-SELF $\subseteq$ Δ_2^{P}-SELF.

(3) PF$^{\Sigma_2^{\mathrm{P}}}$-SELF$_{\text{gen}}$ $=$ Δ_3^{P}-TALLY-SELF $=$ Δ_3^{P}-SELF. *Thus* P/poly $=$ PF$^{\Sigma_2^{\mathrm{P}}}$-SELF$_{\text{gen}}$.

Note that PF$^{\Sigma_2^{\mathrm{P}}}$ is a general upper bound for the complexity of finding circuits, essentially the same one we know for the complexity of a sparse-set description.

Now let us discuss two lower bounds for the complexity of finding circuits that are consequences of our main results. Recall that recognizing a sparse-set description is easier than finding circuits. Thus the lower-bound results we had in §3 yield at least the same lower bounds for the complexity of finding circuits.

The first lower bound is from Theorem 3.1.

THEOREM 4.3. *There is a set in* P/poly *that is not in* PF-SELF$_{\text{gen}}$.

Proof. Note that P/poly $=$ R$_{\mathrm{T}}$(SPARSE) and that PF-SELF$_{\text{gen}}$ $\subseteq$ P-SELF. Thus the set A defined in Theorem 3.1 clearly satisfies the theorem. □

We should note here that the same lower bound is obtained from observations in [1], [6]. Balcázar and Book [6] showed that if L is in PF-SELF$_{\text{gen}}$ (in their notation, L has a self-p-producible circuit), then L has a certain lowness property (more specifically, L has EL_1-lowness). On the other hand, Allender and Hemachandra [1] constructed a sparse set S_0 that does not have such lowness. Thus S_0 satisfies the theorem. Note that every sparse set is trivially in P-SELF; hence S_0 satisfies the theorem because $S_0 \in$ P-SELF $-$ PF-SELF$_{\mathrm{gen}}$, and A does so because $A \in$ P/poly $-$ P-SELF.

Next we improve the lower bound by using Theorem 3.8. In order to state our result we need a complexity class of functions — NPSV — that is often used in the literature (see, e.g., [10]). Let NPSV (respectively, NPSV^L) be the class of functions computed by polynomial-time nondeterministic single-valued transducers (relative to L).

THEOREM 4.4. *There is a set in* P/poly *that is not in* NPSV-SELF$_{\mathrm{gen}}$.

Proof. Note that (NP$\cap$co-NP)-TALLY-SELF $\subseteq$ (NP$\cap$co-NP)-SELF; then it immediately follows from the next lemma that the set B defined in Theorem 3.8 satisfies the theorem. $\square$

LEMMA 4.5. NPSV-SELF$_{\mathrm{gen}}$ = (NP $\cap$ co-NP)-TALLY-SELF.

Proof. First we prove NPSV-SELF$_{\mathrm{gen}}$ $\subseteq$ (NP $\cap$ co-NP)-TALLY-SELF. Let L be any set in NPSV-SELF$_{\mathrm{gen}}$. Then L has a circuit generator $g \in$ NPSV^L. Define a tally set T by $T = \{ \langle 0^n, 0^i, 0^b \rangle_{\mathrm{T}} :$ the ith bit of $g(0^n)$ is $b \in \{0, 1\} \}$. Then one can easily produce $g(0^n)$ by using T as an oracle; hence $L \leq_{\mathrm{T}}^{\mathrm{P}} T$.

Let N be a polynomial-time nondeterministic single-valued oracle transducer that computes g relative to L. Note that $g(0^n)$ is defined for all $n \geq 0$; hence for every n, N^L on 0^n has an accepting computation and N^L outputs $g(0^n)$ on every accepting computation. Once $g(0^n)$ is obtained, one can easily determine whether a given $\langle 0^n, 0^i, 0^b \rangle_{\mathrm{T}}$ is in T. Thus by modifying N we can obtain polynomial-time nondeterministic oracle acceptors M_1 and M_2 that respectively accept T and $\overline{T}$ relative to L.

Next we prove (NP $\cap$ co-NP)-TALLY-SELF $\subseteq$ NPSV-SELF$_{\mathrm{gen}}$. Let L be any set in (NP $\cap$ co-NP)-TALLY-SELF; then there exists a tally set T such that $L \in \mathrm{P}^T$ and $T \in \mathrm{NP}^L \cap \text{co-NP}^L$. Let M and p_M respectively denote a polynomial-time deterministic acceptor that accepts L relative to T and a polynomial time-bound for M. (Note that M on x cannot query strings of length $> p_M(|x|)$.) Define $g : 0^* \to \Sigma^*$ as follows. For each $n \geq 0$, $g(0^n) = \sigma_1 \cdots \sigma_{p_M(n)}$, where $\sigma_i = 1 \leftrightarrow 0^i \in T$. Clearly, g is a circuit generator for L (when an appropriate circuit evaluator is used).

Now to complete the proof we need only to construct a polynomial-time nondeterministic single-valued oracle transducer N that computes g relative to L, thereby showing $g \in$ NPSV^L. Let M_1 and M_2 be polynomial-time nondeterministic oracle acceptors that respectively accept T and $\overline{T}$ relative to L. The computation of N^L on input 0^n proceeds as follows: For each i, $1 \leq i \leq p_M(n)$, N^L guesses the value of σ_i (i.e., $\sigma_i = 1$ if $0^i \in T$, and $\sigma_i = 0$ if $0^i \notin T$) and verifies it by using either M_1 or M_2. Then N^L outputs $\sigma_1 \cdots \sigma_{p_M(n)}$ on each nondeterministic computation that guesses all the σ_i correctly and that verifies their correctness. Clearly, N^L is single-valued and computes g. $\square$

4.2. Equivalence and reducibility to sparse sets. Intuitively, the (relative) complexity of an easiest sparse-set description for a language L gives a lower bound on the power of the reduction that we must use to recognize it (relative to L). Similarly, we have seen that the complexity of its easiest tally-set description gives lower bounds on the resources used to generate any sparse-set description for L.

Note that up to now we have insisted on keeping a $\leq_{\mathrm{T}}^{\mathrm{P}}$ reduction between L and its descriptions because we wanted to study sets in P/poly. If we relax this condition, we can then ask what is the minimum reduction type that allows us to *simultaneously* recognize L and its easiest sparse-set description relative to each other. This leads naturally to the definition of classes E_r(SPARSE) and E_r(TALLY), which are extensively studied in [2], [3], [22].

For a reduction type r, two sets A and B are $\leq_r$-equivalent if $A \leq_r B$ and $B \leq_r A$. We define $\mathrm{E}_r(\mathrm{SPARSE})$ as the class of sets L for which there is a sparse set S such that L and S are $\leq_r$-equivalent. $\mathrm{E}_r(\mathrm{TALLY})$ is defined similarly. If reduction r is transitive, the class $\mathrm{E}_r(\mathrm{SPARSE})$ is also called the *equivalence degree* of the sparse sets under r. As explained in [23], this is not a proper name for nontransitive reduction types, such as $\leq_{\mathrm{T}}^{\mathrm{NP}}$ and $\leq_{f(n)\text{-tt}}^{\mathrm{P}}$, because these reducibilities do not define equivalence relations.

It follows directly from the definitions that P-SELF $= \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$; in particular, every set in P-SELF is $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to its easiest sparse-set description. Also, since $\leq_{\mathrm{T}}^{\mathrm{P}}$ implies $\leq_{\mathrm{T}}^{\mathrm{SN}}$ and $\leq_{\mathrm{T}}^{\mathrm{NP}}$ reducibilities, clearly (NP $\cap$ co-NP)-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE})$ and NP-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{NP}}(\mathrm{SPARSE})$.

Let us consider now a class defined by equivalence to tally sets. If we recall the definition in §4.1, it is immediate that P-TALLY-SELF $= \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{TALLY})$. Therefore, from Corollary 4.2 (1) we have that $\text{PF-SELF}_{\text{gen}} = \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{TALLY})$. This fact was already in [6, Thm. 3.1], where it is shown that sets with self-p-producible circuits are exactly those $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to tally sets.

Concerning this class, Allender and Watanabe ask in [3] whether $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{TALLY}) = \mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE})$. This question arises from the fact that any set $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to a sparse set is also $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible (and thus $\leq_{\mathrm{tt}}^{\mathrm{P}}$-reducible) to some tally set [8]. Therefore, the reduction classes of tally and sparse sets are equal for reducibilities $\leq_{\mathrm{T}}^{\mathrm{P}}$ and $\leq_{\mathrm{tt}}^{\mathrm{P}}$ (and all of them equal P/poly). Similarly, when considering equivalence to a tally set, we may ask whether it is possible to trade the power of $\leq_{\mathrm{T}}^{\mathrm{P}}$ reductions for the access to a slightly more complex oracle, namely, a sparse set.

We can now formulate again the results presented in §3 to give various separations between the equivalence classes mentioned in the preceding. For example, we have pointed out that the construction in Theorem 3.1 can use any unbounded and easily computable function $m(n)$ to give a set A that is not in P-SELF $= \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$. The reader will easily verify that A is $\leq_{m(n)\text{-tt}}^{\mathrm{P}}$-reducible to the sparse set S_A given in that theorem. Therefore, we have the following theorem.

THEOREM 4.6. *For any function $m(n)$ that is unbounded and computable in time polynomial in n,* $\mathrm{R}_{m(n)\text{-tt}}^{\mathrm{P}}(\mathrm{SPARSE}) \not\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Furthermore, the reduction to S_A is conjunctive, so that the result holds in fact for $\leq_{m(n)\text{-c}}^{\mathrm{P}}$ reducibility.

Because the set A is in (NP $\cap$ co-NP)-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE})$, we have the following corollary.

COROLLARY 4.7. $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{P/poly} \cap \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE})$.

Similarly, from Theorem 3.8 and Corollary 3.12 we know that the set $B \in$ NP-SELF is not in $\mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE})$. As a consequence we have the following corollary.

COROLLARY 4.8. $\mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE}) \cap \mathrm{P/poly} \subsetneq \mathrm{E}_{\mathrm{T}}^{\mathrm{NP}}(\mathrm{SPARSE}) \cap \mathrm{P/poly}$.

We next answer the last question in [22] concerning $\mathrm{E}_r(\mathrm{SPARSE})$ classes, that is, we show that $\mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$. The separation does not seem to follow directly from the sets in §3, but we can use the same technique here. To distinguish between $\leq_{\mathrm{T}}^{\mathrm{P}}$ and $\leq_{\mathrm{tt}}^{\mathrm{P}}$ we exploit the adaptiveness of $\leq_{\mathrm{T}}^{\mathrm{P}}$ reducibility, namely, its ability to do prefix search.

THEOREM 4.9. *There is a set D in* $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE}) - \mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Proof. We construct D that meets the following two requirements:

(I) For some sparse set S_D, $D \leq_{\mathrm{T}}^{\mathrm{P}} S_D$ and $S_D \leq_{\mathrm{T}}^{\mathrm{P}} D$.

(II) For every set X, if $D \leq_{\mathrm{tt}}^{\mathrm{P}} X \leq_{\mathrm{tt}}^{\mathrm{P}} D$, then X is not sparse.

Let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial-time deterministic tt-reductions, where the running time of each M_i is bounded by p_i. The construction proceeds in stages, and at each stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ we define $D^{(k)}$ and $S^{(k)}$ that meet the same requirements as in the proof of Theorem 3.1.

Sets $D^{(k)}$ and $S^{(k)}$ at stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ are defined as follows.

(1) Execute steps (1), (2), and (3) as in Theorem 3.1, defining τ, W_1, W_2, L_1, and L_2.

(2) Define also the following sets: $\mathrm{Pref}(\tau) = \{ \langle 0^n, u\rangle \ : \ u \text{ is a prefix of } \tau \}$, $S_1 = \mathrm{Pref}(\tau) \oplus \{ 0\tau \}$, and $S_2 = \mathrm{Pref}(\tau) \oplus \{ 1\tau \}$ (where $X \oplus Y$ is the set $0X \cup 1Y$). Intuitively, S_1 and S_2 contain all the information about τ, plus an indication of whether W_1 or W_2 is used. Moreover, we will see that this information can only be accessed if some kind of prefix search can be used.

(3) Define $D_1 = D^{(k-1)} \cup (L_1 \oplus S_1)$ and $D_2 = D^{(k-1)} \cup (L_2 \oplus S_2)$. Then set $D^{(k)}$ and $S^{(k)}$ as in step (4) of Theorem 3.1, using D_1, D_2, S_1, and S_2 in place of A_1, A_2, W_1, and W_2.

Finally, define $D = \cup_{k\geq 1} D^{(k)}$ and $S_D = \cup_{k\geq 1} S^{(k)}$.

Notice that D is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to S_D by the following reduction: On inputs of the form $1x$, accept if and only if $x \in S_D$. On an input of the form $0x$, with $|x| = n \cdot m$ for some n used in the construction, find the word τ (of length $2n^2$) used in that stage by prefix search on S_D. Compute the sets W_1 and W_2 from τ. Then, if $10\tau \in S_D$, accept if and only if all blocks of x are in W_1; otherwise, accept if and only if all blocks of x are in W_2.

Since D is of the form $L \oplus S_D$ for some set L, it is immediate that $S_D \leq_{\mathrm{T}}^{\mathrm{P}} D$.

Now to finish the proof of the theorem, we must only show that either Case (a) or Case (b) holds at every stage. We do this in the following lemmas. □

Consider again a fixed stage k and all the definitions made at that stage. Of course, Lemma 3.2 is also valid in this context. Compose machines M_i and M_j to give a new oracle machine M_s, which will also be a tt-machine whose running time is bounded by a polynomial p_s.

The next lemma is the analog of Lemma 3.3.

LEMMA 4.10. (i) *For all strings x and y in $L_1 \cup L_2$, if $M_s(0x)$ queries $0y$, then $x = y$.*

(ii) *For all strings x in $L_1 \cup L_2$, $M_s(0x)$ queries neither 110τ nor 111τ.*

Proof. For part (i) suppose that $x \in L_1 \cup L_2$ and that $0y$ is the ℓth word in the list of queries made by $M_s(0x)$ such that $y \in L_1 \cup L_2$ and $y \neq x$. Notice that we can write simply $M_s(0x)$ because this list is independent of the oracle used by M_s.

Define d_3 as the description of a program that, being given $\overline{\ell} \cdot x$ as an input, simulates $M_s(0x)$ and prints its ℓth query minus the first symbol, which is y. Then, on input $\overline{d_3} \cdot (\overline{\ell} \cdot x)$ $(= (d_3\#\ell) \cdot x)$, our universal machine prints a word in $L_1 \cup L_2$ other than x. By the same argument as in Lemma 3.3, the choice of c such that $|d_3\#\ell| \leq c \log n + c$ leads to the contradiction.

Part (ii) is proved similarly by using the following argument: If $M_s(0x)$ produces 110τ or 111τ, then many words in $L_1 \cup L_2$ can be described too easily from x plus a short program. □

LEMMA 4.11. *If Case* (a) *fails, then Case* (b) *holds.*

Proof. Suppose that Case (a) fails, and let X_1 and X_2 be two sets such that $D_1 \leq_{\mathrm{tt}}^{\mathrm{P}} X_1 \leq_{\mathrm{tt}}^{\mathrm{P}} D_1$ and $D_2 \leq_{\mathrm{tt}}^{\mathrm{P}} X_2 \leq_{\mathrm{tt}}^{\mathrm{P}} D_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)+1}$ and $X_2^{\leq p_i(nm)+1}$ is not p_l-sparse.

Define function C as follows. For any $x \in L_1 \cup L_2$, $C(x)$ is the first query in the list produced by $M_i(0x)$ that has different answers from oracles X_1 and X_2.

As before, it is easy to show that the function C is defined on $L_1 \cup L_2$. We also make the following claim.

CLAIM. *The function C is one-to-one on $L_1 \cup L_2$.*

Proof of the claim. Take any $x \in L_1 \cup L_2$, and name $y = C(x)$.

We show first that $M_j(y)$ must query $0x$. This is because D_1 and D_2 differ only in the words of $0L_1$ and $0L_2$ and in the two words 110τ and 111τ. But by Lemma 4.10 $M_j(y)$ cannot query any of these words except $0x$. Therefore, if $M_j(y)$ does not query $0x$, it gets the same list of answers from oracles D_1 and D_2. But then y is in both X_1 and X_2 or in none of them. This contradicts the definition of $y = C(x)$.

To prove that C is one-to-one, assume for a moment that also $C(x') = y$ and hence $M_i(0x')$ queries y, for some $x' \in L_1 \cup L_2$ that is not x. We have now that $M_s(0x')$ queries $0x$ when simulating $M_j(y)$. But this is impossible by Lemma 4.10. This proves the claim. □

Following the same argument as in Lemma 3.4, we can finish the proof of the lemma. □

Notice now that the sparse set S_D is defined, essentially, by adding at every stage the set of prefixes of a single word. It is easy to see that there is a deterministic polynomial-time machine that, relative to S_D, prints all the elements of S_D up to a given length n. This means that S_D is in PF-SELF$_{\text{gen}}$ ($=$ $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(TALLY)) and, because $\leq^{\mathrm{P}}_{\mathrm{T}}$ is transitive, so is D. Thus we have the following corollary.

COROLLARY 4.12. $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{TALLY}) \not\subseteq \mathrm{E}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SPARSE})$.

The last questions that we address deal with the relations between classes of sets reducible to sparse sets under various reducibilities r, R_r(SPARSE). The study of these classes was initiated by Book and Ko [8] and continued later by Ko [15], who focused on conjunctive and disjunctive reducibilities. In [15] four conjectures concerning these classes were formulated. The first one was refuted by Allender, Hemachandra, Ogiwara, and Watanabe.

THEOREM 4.13 [2]. $\mathrm{R}^{\mathrm{P}}_{\mathrm{btt}}(\mathrm{SPARSE}) \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{d}}(\mathrm{SPARSE})$.

Very recently, the second of the conjectures was refuted by Buhrman, Longpré, and Spaan.

THEOREM 4.14 [9]. $\mathrm{R}^{\mathrm{P}}_{\mathrm{bdtt}}(\mathrm{SPARSE}) \subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{c}}(\mathrm{SPARSE})$.

By using the constructions in §3, we prove the remaining two of Ko's conjectures. The first one states that disjunctive reducibility does not have all the power of conjunctive reducibility when applied to sparse sets.

THEOREM 4.15. $\mathrm{R}^{\mathrm{P}}_{\mathrm{c}}(\mathrm{SPARSE}) \not\subseteq \mathrm{R}^{\mathrm{P}}_{\mathrm{d}}(\mathrm{SPARSE})$.

Proof. Build a set A as in Theorem 3.1 but using function $t(n) = 2^{3n}$ instead of $t(n) = 2^n$. We only use the following properties of A:

(i) A is $\leq^{\mathrm{P}}_{\mathrm{c}}$-reducible to some sparse set,

(ii) A is not sparse;

(iii) for every $c \geq 0$, for all sufficiently large n, and for every two different words x and y in $A^{=n}$, $y \notin \mathrm{KT}[c\log n + c, t(n) \mid x]$.

Part (iii) follows by an easy extension of Lemma 3.2. Note also that the construction in Theorem 3.1 can be made much simpler if we are interested only in obtaining these properties.

Now assume that A is $\leq^{\mathrm{P}}_{\mathrm{d}}$-reducible to some sparse set S through a function f, and we will derive a contradiction. For every x we view $f(x)$ as a set so that x is in A if and only if $f(x) \cap S$ is not empty. Assume that f is computable in time $p_i(n)$ and that S has at most $p_j(n)$ elements of length at most n, for two polynomials p_i and p_j.

Since A is not sparse, there are infinitely many n such that $A^{=n}$ has more than $p_j(p_i(n))$ elements. For any such n there are two strings x and y in $A^{=n}$ such that $x > y$ and $S \cap f(x) \cap f(y) \neq \emptyset$; that is, x and y share a string witnessing that they are in $A^{=n}$.

Let $z \in S \cap f(x) \cap f(y)$ be the ℓth query in $f(x)$. Let d be a program that, on input $(n\#\ell) \cdot x$, computes the ℓth element in $f(x)$, which is z, and finds the smallest string u of length n such that $z \in f(u)$. Observe that u must be in $A^{=n}$ and that it must be different from x (by the assumption $x > y$).

The time used by d to do this task can be bounded by $p_k(n) \cdot 2^n$, for some polynomial p_k. Hence $U((d\#n\#\ell) \cdot x)$ prints u in time at most $|d|^2 \cdot (p_k(n) \cdot 2^n)^2$. If n is sufficiently large, this is less than $t(n)$. Furthermore, because f is computable in time bounded by p_i, there is a constant c independent of n such that $|n\#\ell| \leq c\log n + c$. Therefore, for these particular x, u, and c, $u \in \mathrm{KT}[c\log n + c, t(n) \mid x]$.

With the assumption $A \le_{\mathrm{d}}^{\mathrm{P}} S$, this must happen for infinitely many n. But this contradicts (iii). □

As in Theorem 4.6, this result holds not only for $\le_{\mathrm{c}}^{\mathrm{P}}$ but also for $\le_{m(n)\text{-c}}^{\mathrm{P}}$ reducibility, where $m(n)$ is any reasonable unbounded function. And Theorem 4.13 states precisely that it cannot be extended to bounded $m(n)$.

Note also that Buhrman, Longpré, and Spaan [9] have obtained an improvement of this theorem. They use an extension of our techniques to show that the set witnessing the separation can be made $\le_{m(n)\text{-c}}^{\mathrm{P}}$-reducible to some *tally* set, so $\mathrm{R}_{m(n)\text{-c}}^{\mathrm{P}}(\mathrm{TALLY}) \not\subseteq \mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE})$.

The last of Ko's conjectures is an immediate corollary of Theorem 4.15.

COROLLARY 4.16. $\mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$.

5. Final remarks.

Two aspects of our main results are worth noting. The first one is that sets A and B in §3 are recursive and that we can give good bounds for their time complexity. The second one is that the constructions can be used for sparseness bounds much larger than polynomial.

Let us consider first the time complexity of set A in Theorem 3.1. A small modification of its construction will make A decidable in exponential time. The only hard steps in the construction are steps (3) and (4). Step (3) requires finding the string $\tau \notin \mathrm{KT}[2n^2-1, t(2n^2)]$, and this can be done by exhaustive search in time $2^{O(n^2)}$. Step (4), Case (a), can be detected by cycling through all words of length at most nm and using each one to simulate machines M_i and M_j; since M_i and M_j run in polynomial time, this simulation takes time less than 2^n if n is chosen sufficiently large with respect to p_i and p_j. Thus the total time in this step is $2^{O(nm)} \cdot 2^n$, which is $2^{O(n^2)}$.

A similar exponential bound for step (4), Case (b), fails because the polynomial p_i can be arbitrarily large and thus the number of strings to test grows faster than any fixed exponential. However, for the rest of the proof it is enough to test only the strings that are actually queried in the reduction from A to $L(M_j, A)$. There are at most $p_i(nm) \cdot 2^{nm+1}$ such strings, which is less than 2^{n^2} for n sufficiently large. Thus the total running time of this modified construction is exponential in n. It is an open question whether there exists some set satisfying Theorem 3.1 that is decidable in subexponential time.

A similar argument shows that set B in Theorem 3.8 can be built in time $2^{f(n)}$, where f is any function that majorizes every polynomial. Our technique requires a superexponential bound because we have to simulate arbitrary $\le_{\mathrm{T}}^{\mathrm{SN}}$ reductions in a deterministic way.

As for the sparseness bound, we can use the construction in Theorem 3.1 to prove the following: Say that a function $f(n)$ is subexponential if for every $\epsilon > 0$ and for all but finitely many n, $f(n) < 2^{n^\epsilon}$. Then there is some $A \in \mathrm{P/poly}$ that is $\le_{\mathrm{T}}^{\mathrm{P}}$-equivalent to no set whose census is bounded by a subexponential function.

Indeed, take the function $m(n)$ in the proof of Theorem 3.1 to be $\sqrt{n}$ and modify the construction of A as follows:

Step (4), *Case* (b): If $\|L(M_j; A_1)^{\le p_i(nm)}\| > \|L(M_j; A_2)^{\le p_i(nm)}\|$, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$. Informally, we choose the extension of A that maximizes the number of elements in $L(M_j, A)$ at that stage.

Suppose that set X is $\le_{\mathrm{T}}^{\mathrm{P}}$-equivalent to the resulting set A, and let M_i be any $\le_{\mathrm{T}}^{\mathrm{P}}$ reduction from A to X. Recall that polynomial p_i bounds the running time of M_i. With the argument in the proof of Lemma 3.4 we can show that

$$\|X^{\le p_i(n\sqrt{n})}\| \ge \|A^{=n\sqrt{n}}\| = \binom{n}{\sqrt{n}}$$

for infinitely many n — otherwise, at some stage of the construction of A we diagonalize against the reductions making A and X equivalent. Since $\binom{n}{\sqrt{n}}$ is $2^{\Omega(\sqrt{n})}$, it is easy to see that in these conditions

$$\|X^{\leq n}\| \geq 2^{n^{\epsilon}}$$

for some $\epsilon > 0$ depending on p_i and infinitely many n. In other words, the census of X is not bounded by any subexponential function.

Analogous bounds can be achieved for sets B and D in Theorems 3.8 and 4.9, respectively. Note that these bounds are essentially optimal: a standard padding argument shows that for any set L and any $\epsilon > 0$ there is some set $\leq_{\mathrm{m}}^{\mathrm{P}}$-equivalent to L whose census is bounded by $2^{n^{\epsilon}}$.

Acknowledgments. We thank the two referees for their thorough reading and many helpful suggestions. We thank Harry Buhrman and Luc Longpré for providing us with their work [9]. The first author thanks José Balcázar for many discussions on this subject and Ronald Book for inviting him to the University of California at Santa Barbara.

REFERENCES

[1] E. Allender and L. Hemachandra, *Lower bounds for the low hierarchy*, J. Assoc. Comput. Mach., 39 (1992), pp. 234–251.

[2] E. Allender, L. Hemachandra, M. Ogiwara, and O. Watanabe, *Relating equivalence and reducibility to sparse sets*, SIAM J. Comput., 21 (1992), pp. 521–539.

[3] E. Allender and O. Watanabe, *Kolmogorov complexity and degrees of tally sets*, Inform. and Comput., 86 (1990), pp. 160–178.

[4] D. Angluin, *Learning regular sets from queries and counterexamples*, Inform. and Comput., 75 (1987), pp. 87–106.

[5] ———, *Queries and concept learning*, Mach. Learning, 2 (1988), pp. 319–342.

[6] J. Balcázar and R. Book, *Sets with small generalized Kolmogorov complexity*, Acta Inform. 23 (1986), pp. 679–688.

[7] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, EATCS Monographs on Theoretical Computer Science 11, Springer-Verlag, Berlin, 1988.

[8] R. Book and K. Ko, *On sets truth-table reducible to sparse sets*, SIAM J. Comput., 17 (1988), pp. 903–919.

[9] H. Buhrman, L. Longpré, and E. Spaan, SPARSE *Reduces Conjunctively to* TALLY, in Proc. Structure in Complexity Theory 8th Annual Conference, IEEE Computer Society, Washington, DC, 1993, pp. 208–214.

[10] J. Grollmann and A. Selman, *Complexity measures for public-key cryptosystems*, SIAM J. Comput., 17 (1988), pp. 309–335.

[11] J. Hartmanis, *Generalized Kolmogorov complexity and the structure of feasible computations*, in Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1983, pp. 439–445.

[12] J. Hartmanis and Y. Yesha, *Computation times of* NP *sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[13] R. Karp and R. Lipton, *Some connections between nonuniform and uniform complexity classes*, in Proc. 12th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1980, pp. 302–309.

[14] K. Ko, *Continuous optimization problems and a polynomial hierarchy of real functions*, J. Complexity, 1 (1985), pp. 210–231.

[15] ———, *Distinguishing conjunctive and disjunctive reducibilities by sparse sets*, Inform. and Comput., 81 (1989), pp. 62–87.

[16] ———, *On adaptive versus nonadaptive bounded query machines*, Theoret. Comput. Sci., 82 (1991), pp. 51–69.

[17] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[18] M. Li and P. Vitányi, *Kolmogorov complexity and its applications*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, New York, 1990, pp. 187–254.

[19] T. Long, *Strong nondeterministic polynomial-time reducibilities*, Theoret. Comput. Sci., 21 (1982), pp. 1–25.

[20] N. PIPPENGER, *On simultaneous resource bounds*, in Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1979, pp. 307–311.

[21] U. SCHÖNING, *Complexity and Structure*, Lecture Notes in Computer Science 211, Springer-Verlag, Berlin, 1986.

[22] S. TANG AND R. BOOK, *Separating polynomial-time Turing and truth-table reductions by tally sets*, in Proc. 15th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 317, Springer-Verlag, Berlin, 1988, pp. 591–599.

[23] ———, *Reducibilities on tally and sparse sets*, RAIRO Inform. Théor. Appl., 25 (1991), pp. 293–302 (Extended version of [22]).

[24] O. WATANABE, *A formal study of learning via queries*, in Proc. 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 443, Springer-Verlag, Berlin, 1990, pp. 139–152.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1276–1285, December 1993

VC DIMENSION AND UNIFORM LEARNABILITY OF SPARSE POLYNOMIALS AND RATIONAL FUNCTIONS*

MAREK KARPINSKI† AND THORSTEN WERTHER‡

Abstract. The authors prove upper and lower bounds on the VC dimension of sparse univariate polynomials over reals and apply these results to prove uniform learnability of sparse polynomials and rational functions. As an application the solution to the open problem of Vapnik [in *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, Berlin, 1982] on computational approximation of the regression in a class of polynomials used in the theory of empirical data dependences is given.

Key words. VC dimension, computational learning theory, sparse polynomials, regression function

AMS subject classifications. 68T05, 68T10, 68R05, 62J99, 26C05

1. Introduction.

This paper studies the problem of identification (learnability) of sparse real polynomials and rational functions. For the corresponding problem of identification of polynomials and rational functions over arbitrary fields see [1], [8], [9].

We derive linear upper $(4t-1)$ and lower $(3t)$ bounds on the VC dimension of the class of t-sparse polynomials over the real numbers implying uniform probably approximately correct (pac) learnability of this class of functions [3]. The results generalize to uniform distribution-free learnability of sparse polynomials even in the extended metric model of Haussler [10]. Applying this result, we solve Vapnik's open problem on uniform estimation of the polynomial regression function [16].

The analysis of the computational complexity of learning sparse polynomials and rational functions has several motivations. The complexity analysis of algorithms manipulating polynomials usually measures the input length in terms of the degree of polynomials. For polynomials with a small number of terms, this model is not reasonable since sparse polynomials are usually represented by a list of nonzero coefficients and the corresponding exponents. Hence the natural measure of the size of a polynomial is given in terms of its sparsity when the uniform cost model of computation is applied. Recent results indicate also that sparse polynomials play a key role in the harmonic analysis of Boolean circuits [4], [5] and, surprisingly, in the area of learnability of Boolean functions as well [13].

The problems of sparse polynomial interpolation provide a second motivation. In the black-box model, a learning algorithm for sparse polynomials has access to an oracle that gives the value of the polynomial for an arbitrary evaluation point [11]. Grigoriev and Karpinski [7], Ben-Or and Tiwari [1], and Grigoriev, Karpinski, and Singer [8] show that in this oracle model there are efficient algorithms for exact learning (interpolation) of sparse determinants [7], sparse polynomials over fields of characteristic zero [1], and also over finite fields [8].

It is known (see [6], [19]) that elements of vector spaces of real-valued functions are pac learnable. Therefore, the third motivation of this work is to explore the learnability of sparse polynomials as a function class that is not embedded in some vector that space.

Throughout this paper we employ the model of machine learning introduced by Valiant [15] and usually referred to as the PAC model. Valiant used this new model of distribution-

*Received by the editors December 5, 1991; accepted for publication (in revised form) July 21, 1992.

†Department of Computer Science, University of Bonn, Römerstraße 164, 5300 Bonn, Germany, and International Computer Science Institute, 1947 Center Street, Berkeley, California 94704-1105 (marek@theory.cs.uni-bonn.de). The research of this author was supported in part by Leibniz Center for Research in Computer Science, by Deutsche Forschungsgemeinschaft grant KA 673/2-1, and by Science and Engineering Research Council grant GR-E 68297.

‡Department of Computer Science, University of Bonn, Römerstraße 164, 5300 Bonn, Germany (thorsten@theory.cs.uni-bonn.de). Part of the research was done while the author was visiting the International Computer Science Institute, Berkeley, California.

free learning from examples to exhibit and analyze several learning algorithms for Boolean functions.

In this model a concept c from a class C is a subset of an instance space X. Let C be a class of concepts from X. Each example $\langle x, l\rangle$ of a target concept c consists of an instance $x \in X$ and its classification (label) $l \in \{0, 1\}$ as either a positive example $\langle x, 1\rangle$ or a negative example $\langle x, 0\rangle$, indicating whether or not x is a member of the set c. In the Valiant model of learning, each example is drawn independently from a fixed but unknown distribution P on X and is labeled consistently with the unknown target concept c.

In the PAC model, a *learning function* for the concept class C is a function that, given a large enough, randomly drawn sample, returns a *hypothesis* that is with high probability a good approximation (with respect to the unknown distribution P) to the target concept, no matter which concept from C we are trying to learn. The error of the hypothesis is the probability that the hypothesis disagrees with the target on a randomly (with respect to P) drawn example.

Upper and lower bounds on the sample complexity for learning various concept classes have been given in [3], [6], [18]. These bounds are based on the Vapnik–Chervonenkis (VC) dimension of a class C.

DEFINITION 1.1. For a concept class C on X and for $S \subset X$, let $\Pi_C(S)$ be the set of subsets T of S such that $T = S \cap c$ for some concept c in C. Thus $\Pi_C(S)$ is the restriction of concept class C to the set S. If $\Pi_C(S) = 2^S$, then the set S is *shattered* by C. The *Vapnik–Chervonenkis dimension* (VC dimension) of the class C is the largest integer d such that some set $S \subset X$ of size d is shattered by C.

Let $\mathcal{F}$ be a collection of real-valued functions on a set X. We investigate learnability of the concept class $\mathrm{pos}(f_0 - \mathcal{F})$ defined as the collection of all concepts $\mathrm{pos}(f_0 - f) = \{x \in X \mid f_0(x) - f(x) > 0\}$ for $f \in \mathcal{F}$ and $f_0 \notin \mathcal{F}$ an arbitrary real function on X.

This paper explores the VC dimension of the concept class $\mathcal{P}_t$, where $\mathcal{P}_t \subset \mathbb{R}[x]$ denotes the set of t-sparse univariate polynomials over the real numbers, i.e, for each $p \in \mathcal{P}_t$ the number of nonzero coefficients of p is bounded by t. We define the VC dimension of $\mathcal{P}_t$ to be the VC dimension of the concept class $\mathrm{pos}(y - \mathcal{P}_t) \subset \mathbb{R}^2$.

Valiant's model of learning can also be thought of as learning the border between positive and negative examples. In this situation we consider $\{0, 1\}$-valued indicator functions. Hence, in context to the problem of pattern recognition [16], we define examples (x, y) from the instance space $X = \mathbb{R}^2$ to be labeled positive if the point (x, y) lies above the t-sparse polynomial f ($f \in \mathcal{P}_t$ the unknown target concept), and negative if the point (x, y) lies below the t-sparse polynomial, i.e.,

$$\langle (x, y), 1\rangle \ \text{ iff } \ y \geq f(x) \qquad \text{and} \qquad \langle (x, y), 0\rangle \ \text{ iff } \ y < f(x).$$

Let $S \subset \mathbb{R}^2$ be the set of points $\{(x_i, y_i)\}_{i=1,\dots,d}$ of size d for $x_1 < x_2 < \cdots < x_d$. A t-sparse polynomial f is said to satisfy a labeling $\sigma \in \{0, 1\}^d$ on S if the points (x_i, y_i) are positive examples for f if $\sigma(i) = 1$ and negative examples for f if $\sigma(i) = 0$. The set S is shattered by the class of t-sparse polynomials $\mathcal{P}_t$, if and only if for each labeling $\sigma \in \{0, 1\}^d$ there exists a t-sparse polynomial f_σ satisfying σ on S.

One of the shortcomings of the standard PAC model is that it is defined only for $\{0, 1\}$-valued functions. Haussler [10] proposes a generalization of the PAC model for distribution-free learning of functions that take values in an arbitrary metric space. This is of particular interest when real-valued functions are learned.

Let $\mathcal{F}$ be a family of functions from a domain X to a set Y with metric d_Y. Similar to the results of Blumer, Ehrenfeucht, Haussler, and Warmuth [2], the results of Haussler show that the essential condition for distribution-free uniform learnability of $\mathcal{F}$ is the finiteness of the VC *dimension of the graphs of functions in* $\mathcal{F}$, which is an extension of the standard VC dimension.

For each $f \in \mathcal{F}$ we denote by $I(f)$ the function from $X \times Y \times \mathbb{R}^+$ into $\{0, 1\}$ defined by

$$I(f)(x, y, \epsilon) = \begin{cases} 1 & \text{if } d_Y(f(x), y) \le \epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

Let $I(\mathcal{F}) = \{I(f) \mid f \in \mathcal{F}\}$. We define the *metric* VC *dimension* of $\mathcal{F}$ as the (standard) VC dimension of $I(\mathcal{F})$. Let m-VCdim($\mathcal{F}$) denote the metric VC dimension of $\mathcal{F}$.

This paper is organized as follows. Section 2 gives lower and upper bounds on the VC dimension of sparse polynomials, proving that the class of sparse polynomials is uniformly learnable (see [3]). Section 3 generalizes these bounds for the metric VC dimension of t-sparse polynomials. Applying results of Vapnik [17], we show that the regression function can be approximated uniformly by sparse polynomials.

2. Bounds on the VC dimension of $\mathcal{P}_t$. We show that the VC dimension of $\mathcal{P}_t$ is linear in t.

2.1. Lower bounds. We start with a lower bound on the VC dimension of $\mathcal{P}_1$.

LEMMA 2.1. *The* VC *dimension of* $\mathcal{P}_1$ *is bounded from below by* 3.

Proof. We show that for each labeling $\sigma \in \{0, 1\}^3$ there is a 1-sparse polynomial f_σ satisfying σ on the set $S = \{(-3, 4), (1, 2), (7, 6)\}$ of size 3. Choose, for example, $f_{000} = 7$, $f_{001} = 5$, $f_{010} = x^2$, $f_{011} = -2x$, $f_{100} = 3x$, $f_{101} = 3$, $f_{110} = x$, and $f_{111} = 1$ (see Fig. 1). □

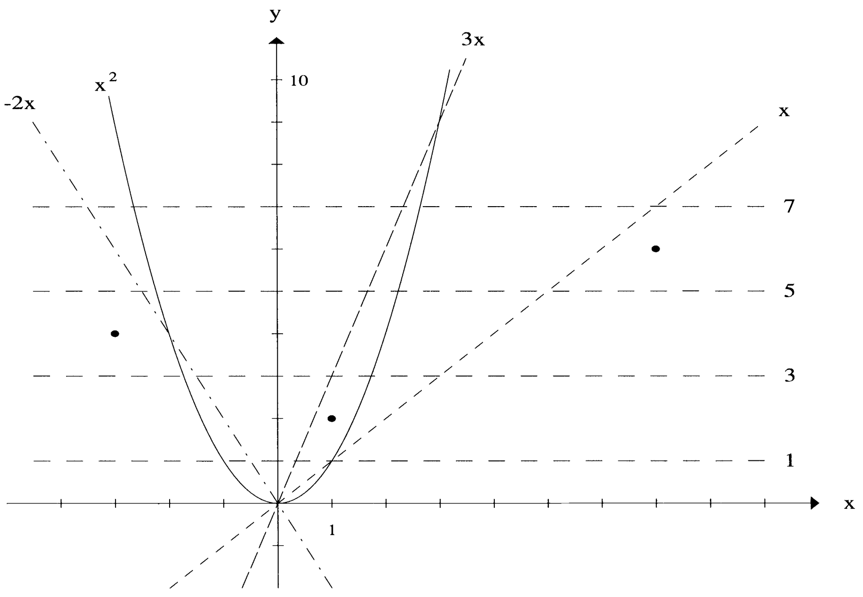


FIG. 1. *Monomials shattering the set S of size* 3.

In the following proofs, it will be convenient to assume that no element of a set S, which is shattered by some set of sparse polynomials, lies on the graph of these polynomials. In Proposition 2.2 we prove that this assumption can be made without loss of generality.

PROPOSITION 2.2. *Let a set S of size d be shattered by the class of t-sparse polynomials. Then there are a set* $Z = \{(x_i, y_i)\}_{i=1,\ldots,d}$ *and constants* $\epsilon_i > 0$, $i = 1, \ldots, d$, *such that every set* $S' = \{(\bar{x}_i, \bar{y}_i)\}_{i=1,\ldots,d}$ *with* $|(\bar{x}_i, \bar{y}_i) - (x_i, y_i)| \le \epsilon_i$ *is shattered by t-sparse polynomials.*

Proof. For each $\sigma \in \{0,1\}^d$ there is a t-sparse polynomial f_σ satisfying σ on S. For $i = 1, \dots, d$, we define the regions

$$M_i = \left\{ (x,y) \mid \forall \sigma \in \{0,1\}^d : \left\{ \begin{array}{ll} y > f_\sigma(x) & \text{if } \sigma(i) = 1 \\ y < f_\sigma(x) & \text{if } \sigma(i) = 0 \end{array} \right\} \right\}.$$

Since S is shattered by $\{f_\sigma\}_{\sigma\in\{0,1\}^d}$, there exists a point (x_i, y_i) and a constant $\epsilon_i > 0$ such that the ball

$$B_{\epsilon_i}(x_i, y_i) = \{ (x,y) \mid |(x,y) - (x_i,y_i)| \le \epsilon_i \}$$

is a proper subset of M_i. Hence each set S' defined as in the proposition is shattered by the t-sparse polynomials $\{f_\sigma\}_{\sigma\in\{0,1\}^d}$. $\square$

Lemma 2.3 states that the VC dimension of sparse polynomials is subadditive. We use this lemma to derive a lower bound.

Given a set shattered by t_1-sparse polynomials and a set shattered by t_2-sparse polynomials, we construct a set shattered by $(t_1 + t_2)$-sparse polynomials.

LEMMA 2.3. *For $t_1, t_2 \in \mathbb{N}$ let d_1, d_2 denote the* VC *dimension of $\mathcal{P}_{t_1}, \mathcal{P}_{t_2}$, respectively. Then the* VC *dimension of $\mathcal{P}_{t_1+t_2}$ is at least $d_1 + d_2$.*

Proof. Let S_1 and S_2 denote some sets of points of size d_1, d_2, respectively, shattered by t_1-sparse polynomials, t_2-sparse polynomials, respectively.

Let $S_1 = \{(x_i^{(1)}, y_i^{(1)})\}_{i=1,\dots,d_1}$ and $S_2 = \{(x_j^{(2)}, y_j^{(2)})\}_{j=1,\dots,d_2}$. For a labeling $\sigma^{(1)} \in \{0,1\}^{d_1}$ let $f_{\sigma^{(1)}}$ satisfy $\sigma^{(1)}$ on S_1, and for a labeling $\sigma^{(2)} \in \{0,1\}^{d_2}$ let $g_{\sigma^{(2)}}$ satisfy $\sigma^{(2)}$ on S_2.

In order to show that the VC dimension of $\mathcal{P}_{t_1+t_2} \ge d_1 + d_2$, we modify the sets S_1 and S_2 (and the corresponding polynomials shattering S_1 and S_2) such that the union of these modified sets is shattered by polynomials derived by adding some of the modified polynomials.

First, we pull the sets S_1 and S_2 apart such that the absolute values of the x-coordinates of points in S_1 are at most $\frac{1}{2}$ and the absolute values of the x-coordinates of points in S_2 are at least 2.

Let

$$c_1 > 2 \cdot \max_{(x_i,y_i)\in S_1} \{|x_i|\} \qquad \text{and} \qquad c_2 < \frac{1}{2} \cdot \min_{(x_j,y_j)\in S_2} \{|x_j|\}.$$

By Proposition 2.2 we may assume that $c_2 > 0$.

Then, the set

$$\bar{S}_1 = \{(\bar{x}_i, \bar{y}_i)\}_{i=1,\dots,d_1} \quad \text{with } (\bar{x}_i, \bar{y}_i) = \left(\frac{x_i}{c_1}, y_i\right), \ (x_i, y_i) \in S_1,$$

is of size d_1 and is shattered by the set of t_1-sparse polynomials $\{\bar{f}_{\sigma^{(1)}}\}_{\sigma^{(1)}\in\{0,1\}^{d_1}}$, where, $\bar{f}_{\sigma^{(1)}}(x) = f_{\sigma^{(1)}}(c_1 x)$.

Similarly, the set

$$\bar{S}_2 = \{(\bar{x}_j, \bar{y}_j)\}_{j=1,\dots,d_2} \quad \text{with } (\bar{x}_j, \bar{y}_j) = \left(\frac{x_j}{c_2}, y_j\right), \ (x_j, y_j) \in S_2,$$

is of size d_2 and is shattered by the set of t_2-sparse polynomials $\{\bar{g}_{\sigma^{(2)}}\}_{\sigma^{(2)}\in\{0,1\}^{d_2}}$, where $\bar{g}_{\sigma^{(2)}}(x) = g_{\sigma^{(2)}}(c_2 x)$.

$\bar{S}_1$ and $\bar{S}_2$ satisfy the conditions claimed in the preceding, i.e., $\forall (x,y) \in \bar{S}_1 : |x| < \frac{1}{2}$ and $\forall (x,y) \in \bar{S}_2 : |x| > 2$.

Let ϵ_i denote the minimal distance of the point $(x_i, y_i) \in \bar{S}_1$ to some point from $\{(x_i, \bar{f}_{\sigma^{(1)}}(x_i))\}_{\sigma^{(1)} \in \{0,1\}^{d_1}}$, i.e.,

$$\epsilon_i = \min_{f \in \{\bar{f}_{\sigma^{(1)}}\}} |f(x_i) - y_i|.$$

Similarly, for each point $(x_j, y_j) \in \bar{S}_2$ define δ_j by

$$\delta_j = \min_{g \in \{\bar{g}_{\sigma^{(2)}}\}} |g(x_j) - y_j|.$$

Again, by Proposition 2.2 we assume that $\epsilon_i, \delta_j > 0$.

Our next step is to modify the set $\bar{S}_2$ and the polynomials from $\{\bar{g}_{\sigma^{(2)}}\}$ to some set $= 3\bar{S}a\bar{\bar{S}}_2$ shattered by the modified polynomials $\{= 3\bar{g}a\bar{\bar{g}}_{\sigma^{(2)}}\}$ such that for each $\sigma^{(1)} \in \{0, 1\}^{d_1}$ and $\sigma^{(2)} \in \{0, 1\}^{d_2}$ the polynomial $\bar{f}_{\sigma^{(1)}} + = 3\bar{g}a\bar{\bar{g}}_{\sigma^{(2)}}$ satisfies $\sigma^{(1)}$ on $\bar{S}_1$ and $\sigma^{(2)}$ on $= 3\bar{S}a\bar{\bar{S}}_2$.

For each point $(x_j, y_j) \in \bar{S}_2$ define Δ_j by

$$\Delta_j = \max_{f \in \{\bar{f}_{\sigma^{(1)}}\}} |f(x_j)|.$$

For some even integer N we transform the set $\bar{S}_2$ into the set $\bar{S}_2^N$ by

$$(x_j, y_j) \in \bar{S}_2 \implies (x_j, x_j^N \cdot y_j) \in \bar{S}_2^N.$$

Since N is even, x^N is positive and the set $\bar{S}_2^N$ is shattered by the set of t_2-sparse polynomials $\{x^N \cdot \bar{g}_{\sigma^{(2)}}\}$. The minimal distance of the point $(x_j, y_j) \in \bar{S}_2^N$ to some point from $\{(x_j, x_j^N \bar{g}_{\sigma^{(2)}}(x_j))\}$ is $x_j^N \cdot \delta_j$.

We choose the parameter N to be large enough to satisfy the following two conditions:

(i) The polynomials $\{x^N \cdot \bar{g}_{\sigma^{(2)}}\}$ may not interfere with the shattering of $\bar{S}_1$, i.e.,

$$x_i^N \cdot \bar{g}_{\sigma^{(2)}}(x_i) < \epsilon_i \quad \text{for all } (x_i, y_i) \in \bar{S}_1 \text{ and for all } \sigma^{(2)} \in \{0, 1\}^{d_2}.$$

Let G be the maximum of the absolute values of the polynomials $\{\bar{g}_{\sigma^{(2)}}(x)\}$ for $|x| \le 1/2$ (all $|x_i| < \frac{1}{2}$) and ϵ the maximum over all ϵ_i. Then we choose N according to

$$G \cdot \left(\frac{1}{2}\right)^N < \epsilon, \qquad \text{i.e.,} \quad N > \log_2\left(\frac{G}{\epsilon}\right).$$

(ii) The polynomials $\{\bar{f}_{\sigma^{(1)}}\}$ may not interfere with the shattering of $\bar{S}_2^N$, i.e.,

$$\Delta_j < x_j^N \cdot \delta_j \quad \text{for all } (x_j, y_j) \in \bar{S}_2.$$

Since the absolute value of the x_j's is at least 2 and N is even, there exists such an N.

Let $\bar{S}_1 = \{(x'_i, y'_i)\}_{i=1,\cdots,d_1}$ with $x'_1 < \cdots < x'_{d_1}$, and let $\bar{S}_2^N = \{(x''_j, y''_j)\}_{j=1,\ldots,d_2}$ with $x''_1 < \cdots < x''_{d_2}$.

Let $S = \bar{S}_1 \cup \bar{S}_2^N$. $S = \{(x_k, y_k)\}_{k=1,\ldots,d_1+d_2}$, and $x_1 < \cdots < x_{d_1+d_2}$. For $\sigma \in \{0, 1\}^{d_1+d_2}$ we define $\sigma_1 \in \{0, 1\}^{d_1}$ and $\sigma_2 \in \{0, 1\}^{d_2}$ by

$$\sigma_1(i) = \sigma(k) \text{ iff } x_k = x'_i \qquad \text{and} \qquad \sigma_2(j) = \sigma(k) \text{ iff } x_k = x''_j.$$

S is of size $d_1 + d_2$ and is shattered by the set of $(t_1 + t_2)$-sparse polynomials $\{h_\sigma\}_{\sigma \in \{0,1\}^{d_1+d_2}}$, where $h_\sigma = \bar{f}_{\sigma_1} + x^N \bar{g}_{\sigma_2}$. Hence the VC dimension of the class of $(t_1 + t_2)$-sparse polynomials is at least $d_1 + d_2$. □

We are now able to state our lower bound on the VC dimension of t-sparse polynomials.

THEOREM 2.4. *The* VC *dimension of t-sparse polynomials is at least* $3t$.

Proof. Combine Lemmas 2.3 and 2.1. □

2.2. Upper bounds. The main tool in this section is *Descartes' Rule of Signs*, used to derive an upper bound on the number of roots of t-sparse polynomials. This leads to a first upper bound on the VC dimension of t-sparse polynomials. Considering the structure of sparse polynomials with the maximal number of roots, we derive a (slight) improvement of the upper bound.

We begin with the well-known Descartes' Rule. Let $f_t = \sum_{i=1}^{t} c_i x^{e_i} \in \mathbb{R}[x]$, $f \not\equiv 0$, be a t-sparse polynomial with $e_i < e_{i+1}, i = 1, \dots, t-1$. The sequence $c = (c_1, c_2, \dots, c_t)$ is said to have a sign alternation at position i if $c_i c_{i+1} < 0$ (zero coefficients are deleted from the sequence). Denote by $s(f_t)$ the number of sign alternations in c. Let $n^+(f_t)$ denote the number of positive real roots of f_t counted with multiplicity.

THEOREM 2.5 (DESCARTES' RULE). *Let $f \in \mathbb{R}[x]$, $f \not\equiv 0$, be a t-sparse polynomial. Then $s(f) - n^+(f)$ is a nonnegative even integer.*

Hence the number of positive real roots of a t-sparse real polynomial $f \not\equiv 0$ is strictly less than its sparsity t. The (total) number of real roots of f is bounded by $2t-1$ (where the root at the origin is counted without multiplicity).

Let $f \in \mathbb{R}[x]$. f is said to be *even* if and only if $f_t(x) = f_t(-x)$ (i.e., $\forall i = 1, \dots, t$: e_i is even), and f is said to be *odd* if and only if $f_t(x) = -f_t(-x)$ (i.e., $\forall i = 1, \dots, t$: e_i is odd). We call f *symmetric* if and only if f is odd or even.

LEMMA 2.6. *Let $f_t \in \mathbb{R}[x]$, $f_t \not\equiv 0$, be a t-sparse polynomial. If f_t has the maximal number of $2t-2$ nonzero real roots, then f is symmetric.*

Proof. Let $f_t = \sum_{i=1}^{t} c_i x^{e_i} \in \mathbb{R}[x]$, for $e_i < e_{i+1}, i = 1, \dots, t-1$. Assume f_t has $2t-2$ nonzero real roots. Then f_t has $t-1$ positive roots. Hence the sequence of coefficients c of f_t has $t-1$ sign alternations. Furthermore, the $t-1$ negative roots are positive roots for $f_t(-x)$. Let $c' = ((-1)^{e_1} c_1, \dots, (-1)^{e_t} c_t)$ denote the sequence of coefficients of $f_t(-x)$. Suppose f_t is not symmetric. Then there is an index i such that e_i and e_{i+1} are not both even or both odd. Therefore, $(-1)^{e_i} c_i \cdot (-1)^{e_{i+1}} c_{i+1} = (-1) \cdot c_i c_{i+1} > 0$ since $c_i c_{i+1} < 0$. Hence c' has at most $t-2$ sign alternations, contradicting the assumption that f_t has $t-1$ negative real roots. □

Using Descartes' estimate on the number of positive real roots of a sparse polynomial, we deduce the (exact) VC dimension of $\mathcal{P}_t$ restricted to the right half-space.

LEMMA 2.7. *The* VC *dimension of the concept class* $\mathrm{pos}(y - \mathcal{P}_t)$ *restricted to the right half-space equals* $2t$.

Proof. Let d denote the VC dimension of $\mathcal{P}_t$ restricted to the right half-space. From Fig. 1 the VC dimension of $\mathcal{P}_1$ restricted to the right half-space is at least 2. By Lemma 2.3 we have $d \geq 2t$. Hence we have to show $d \leq 2t$.

Let $S = \{(x_i, y_i)\}_{i=1,\dots,d}$, $0 < x_1 < x_2 < \cdots < x_d$, be a set of points shattered by t-sparse polynomials. Let f_1 and f_2 be t-sparse polynomials satisfying the two alternating labelings $\sigma_1 = (1, 0, 1, 0, \dots, 1, 0)$ and $\sigma_2 = (0, 1, 0, 1, \dots, 0, 1)$. Let $F = (f_1 - f_2)$. Note that F is $2t$-sparse and $s(F) \leq 2t-1$. Furthermore, $F(x_i) \cdot F(x_{i+1}) < 0$ for $i = 1, \dots, d-1$, forcing F to have at least $d-1$ positive real roots. By Descartes' Rule $d - 1 < 2t$, proving the statement. □

By Lemma 2.7 the VC dimension of $\mathcal{P}_t$ is bounded by $4t$. With Theorem 2.4 the VC dimension of $\mathcal{P}_t$ is linear in t.

Theorem 2.8 gives an improvement of the upper bound on the VC dimension of $\mathcal{P}_t$. As a consequence, the VC dimension of 1-sparse polynomials is exactly 3.

THEOREM 2.8. *The* VC *dimension of* $\mathcal{P}_t$ *is at most* $4t - 1$.

Proof. Assume for the purpose of contradiction that the set $S = \{(x_i, y_i)\}_{i=1,\dots,4t}$ for $x_1 < x_2 < \cdots < x_{2t} < 0 < x_{2t+1} < \cdots < x_{4t}$ is shattered by t-sparse polynomials.

Let α_1, α_2 denote the two alternating labelings of size $2t$, i.e.,

$$\alpha_1 = \underbrace{(1,0,1,0,\ldots,1,0)}_{2t} \qquad \text{and} \qquad \alpha_2 = \underbrace{(0,1,0,1,\ldots,0,1)}_{2t}.$$

Consider the following four (catenated) labelings of size $4t$ on the set S:

$$\sigma_1 = \alpha_1 \circ \alpha_1, \qquad \sigma_2 = \alpha_2 \circ \alpha_2, \qquad \gamma_1 = \alpha_2 \circ \alpha_1, \qquad \gamma_2 = \alpha_1 \circ \alpha_2.$$

Let the t-sparse polynomials f_1, f_2 and g_1, g_2 satisfy the labelings σ_1, σ_2 and γ_1, γ_2.

Define $F = f_1 - f_2$ and $G = g_1 - g_2$. Note that both F and G are $2t$-sparse. By the alternating structure of the labelings, both F and G have at least $4t-2$ nonzero real roots. From Lemma 2.6, F and G are symmetric.

We show that F is odd and G is even. Assume F is even, and let $|x_{2t}| < x_{2t+1}$. Then $F(-x_{2t}) = F(x_{2t}) > 0$ and $F(x_{2t+1}) < 0$, i.e., F has an extra positive root in the interval $(-x_{2t}, x_{2t+1})$, contradicting the upper bound on the number of positive real roots. For $|x_{2t}| > x_{2t+1}$ F has an extra negative root in the interval $(x_{2t}, -x_{2t+1})$. The proof that G is even is similar.

Note that F is odd implies that both f_1 and f_2 are odd (if some monomial occurs in f_1 and in f_2 as well, F would be at most $2t-1$-sparse and hence, by Descartes' Rule, F would have at most $4t-3$ real roots). Similarly, both g_1 and g_2 are even. Then, without loss of generality, we may assume (for the sake of simplicity of notation) that the x-values of the points from S are symmetric as well, i.e., $x_i = -x_{4t+1-i}$, $i = 1, \ldots, 2t$.

We define $2t-1$ intervals J_i on the negative real line by $J_i = (x_i, x_{i+1})$, $i = 1, \ldots, 2t-1$. We prove that for each $i = 1, \ldots, 2t-1$ at least two polynomials from $\{f_1, f_2, g_1, g_2\}$ have a (negative) root in the interval J_i. We distinguish two cases.

1. Let y_i and y_{i+1} have different signs. Assume $y_i < 0$, $y_{i+1} > 0$, and i odd. Then, by definition of the labelings, $f_1(x_i), g_2(x_i) < y_i < 0$ and $f_1(x_{i+1}), g_2(x_{i+1}) > y_{i+1} > 0$. Hence f_1 and g_2 have a root in J_i. If i is even, f_2 and g_1 have a root in J_i. The case $y_i > 0$, $y_{i+1} < 0$ is symmetric.

2. Let y_i and y_{i+1} have equal signs. We show that f_1 or g_2 and f_2 or g_1 have a root in J_i.

Assume $y_i, y_{i+1} > 0$ and i odd. Then $f_1(x_{i+1}), g_2(x_{i+1}) > y_{i+1} > 0$. Assume f_1 has no root in J_i (f_1 is strictly positive in J_i). Then f_1 is strictly negative in the interval (x_{4t-i}, x_{4t-i+1}) (f_1 is odd). Since g_2 is even, $g_2(x_{4t-i+1}) > 0$ and $g_2(x_{4t-i}) < f_1(x_{4t-i}) < 0$. Hence g_2 has a root in the interval (x_{4t-i}, x_{4t-i+1}) and (g_2 is symmetric) g_2 has a root in J_i. Similarly, we can show that either f_2 or g_1 has a root in J_i. The remaining cases are symmetric.

Hence the total number of negative roots of the polynomials from $\{f_1, f_2, g_1, g_2\}$ is at least $2 \cdot (2t-1) = 4t-2$, contradicting the assumption that each polynomial from $\{f_1, f_2, g_1, g_2\}$ is t-sparse (each polynomial has at most $t-1$ negative roots summing up to at most $4t-4$ negative roots). This proves the claimed upper bound of $4t-1$ on the VC dimension of t-sparse polynomials. $\square$

Note that the bounds derived in this subsection remain valid when restricted to t-sparse polynomials over the rational numbers and t-sparse polynomials over the integers.

Let $\mathcal{R}_t$ denote the set of real rational functions with t-sparse numerator and t-sparse denominator. Following the proof of Lemma 2.7, we derive the upper bound of $4t^2$ on the VC dimension of $\mathrm{pos}(y - \mathcal{R}_t)$, proving uniform learnability of t-sparse rational functions for any fixed t.

THEOREM 2.9. *The* VC *dimension of* $\mathcal{R}_t$ *is at most* $4t^2$.

Proof. Let d denote the VC dimension of $\mathrm{pos}(y - \mathcal{R}_t)$. Consider the two rational functions $f_1 = \frac{g_1}{h_1}$, $f_2 = \frac{g_2}{h_2}$ from $\mathcal{R}_t$ satisfying the alternating labelings. Then $f_1(x) = f_2(x)$ for at least $d-1$ points, that is, the $2t^2$-sparse polynomial $g_1h_2 - g_2h_1$ has to have at least $d-1$ real roots. From Theorem 2.5 we have $d-1 \leq 4t^2-1$. □

Because of the finiteness of the VC dimension of the classes $\mathcal{P}_t$ and $\mathcal{R}_t$ we can state the following theorem without explicitly stating learning algorithms for these classes [3].

THEOREM 2.10. *The classes of sparse polynomials and sparse rational functions are uniformly learnable.*

3. Approximating polynomial regression. In this subsection we investigate the learnability of sparse polynomials in the generalized PAC model of Haussler. The essential condition for distribution-free uniform learnability (in this model) of the class of sparse polynomials is the finiteness of the metric VC dimension of $\mathcal{P}_t$. We prove linear bounds for m-VCdim$(\mathcal{P}_t)$.

As an application we consider a central problem in computational regression theory—the problem of determining the number of terms in an arranged system of functions. A special case of this problem is the approximation of polynomial regression (see [16], pp. 254–266).

The problem is as follows. Suppose that a statistical model associates a quantity y with the variable x by the equation

$$y = R(x) + \xi,$$

where $R(x)$ is a polynomial of unknown degree and ξ is an error not depending on x (with zero mean and finite variance). The problem is to *estimate* the polynomial $R^*(x)$ that is close to $R(x)$.

The classical scheme of approximating polynomial regression, which involves the determination of the true degree n of regression and approximates the regression in a class of polynomials of this degree, can be successfully implemented only when large samples are used. For small samples the classical scheme may yield errorneous results. The reason for this is the possible large degree of the regression and therefore the large metric VC dimension (capacity) of the class of polynomials of degree n. Applying the method of structural risk minimization, Vapnik [16] avoids the determination of the actual degree and proves that the polynomial regression can be estimated uniformly by *sparse polynomials.* In this case the sample size depends only on the *sparsity* (!) of the regression and the capacity of the class of sparse polynomials.

Therefore, the problem reduces to the determination of the metric VC dimension of sparse polynomials (independent of the degree). We prove linear bounds on the metric VC dimension of t-sparse polynomials and, as a direct consequence, derive the surprising result that the regression function can be approximated uniformly by sparse polynomials.

First, we construct a lower bound on the metric VC dimension of the class $\mathcal{P}_t$.

LEMMA 3.1. m-VCdim$(\mathcal{P}_t) \geq$ VCdim$(\mathcal{P}_t)$.

Proof. Let $d =$ VCdim$(\mathcal{P}_t)$, and let $S = \{(x_i, y_i)\}_{i=1,\dots,d}$ be a set of points shattered (in the standard sense) by the set of t-sparse polynomials $\{f_\sigma\}_{\sigma\in\{0,1\}^d} \subset \mathcal{P}_t$, i.e.,

$$\forall i = 1,\dots,d \;\; \forall\sigma \in \{0,1\}^d \; : \; f_\sigma(x_i) - y_i \begin{cases} \leq 0 & \text{if } \sigma(i) = 1, \\ > 0 & \text{if } \sigma(i) = 0. \end{cases}$$

Let ϵ be defined by

$$\epsilon = \max_{i=1,\dots,d} \; \max_{\sigma,\sigma(i)=1} \; y_i - f_\sigma(x_i).$$

Then

$$\forall i = 1,\dots,d \;\; \forall\sigma \in \{0,1\}^d \; : \; |f_\sigma(x_i) - (y_i - \epsilon)| \begin{cases} \leq \epsilon & \text{if } \sigma(i) = 1, \\ > \epsilon & \text{if } \sigma(i) = 0, \end{cases}$$

i.e., the set $S_\epsilon = \{(x, y - \epsilon, \epsilon) \mid (x, y) \in S\}$ of size d is shattered (in the metric sense) by the set of t-sparse polynomials $\{f_\sigma\}_{\sigma \in \{0,1\}^d} \subset \mathcal{P}_t$. Hence m-VCdim$(\mathcal{P}_t) \geq$ VCdim$(\mathcal{P}_t)$. □

We introduce the following lemma to derive an upper bound on m-VCdim$(\mathcal{P}_t)$.

LEMMA 3.2. *Let $S = \{(x_i, y_i, \epsilon_i)\}_{i=1,\dots,4}$, where $x_1 < x_2 < x_3 < x_4$. Let $\sigma_1 = (1, 0, 0, 1)$, $\sigma_2 = (0, 1, 1, 0)$, $\sigma_3 = (1, 0, 1, 0)$, $\sigma_4 = (0, 1, 0, 1)$ be labelings on S. Let $\{f_i\}_{i=1,\dots,4}$ be continuous functions satisfying σ_i on S (in the metric sense). Then at least one of the pairs of functions (f_1, f_2), (f_1, f_3), (f_1, f_4), (f_3, f_4) has an intersection point in the interval (x_1, x_4).*

Proof. Consider the 2^8 cases for $f_i(x_j) > y_j + \epsilon_j$ or $f_i(x_j) < y_j - \epsilon_j$ if $\sigma_i(j) = 0$. □

THEOREM 3.3. *The metric* VC *dimension of the class of t-sparse polynomials is at most* $48t - 9$.

Proof. Let $d =$ m-VCdim$(\mathcal{P}_t)$ and $S = \{(x_i, y_i, \epsilon_i)\}_{i=1,\dots,d}$, where $x_1 < x_2 < \cdots < x_d$. Assume S is shattered by t-sparse polynomials. Consider the labelings $\sigma_1 = (1, 0, 0, 1, 0, 0, 1, 0, 0, \dots)$, $\sigma_2 = (0, 1, 1, 0, 1, 1, 0, 1, 1, \dots)$, $\sigma_3 = (1, 0, 1, 0, 1, 0, \dots)$, and $\sigma_4 = (0, 1, 0, 1, 0, 1, \dots)$. Let $f_1, \dots, f_4$ be t-sparse polynomials satisfying $\sigma_1, \dots, \sigma_4$. Then by Lemma 3.2 and the pigeon-hole principle there are two polynomials with at least m intersections within the first $4 + 12(m - 1)$ points of S. By Descartes' Rule $m < 4t$ since $f_i - f_j$ is at most $2t$-sparse. Hence the size of S is strictly less than $4 + 12(4t - 1) = 48t - 8$. □

COROLLARY 3.4. *For any fixed $t \in \mathbb{N}$ the class of t-sparse polynomials is uniformly and distribution-free learnable in the metric* PAC *model.*

We prove linear bounds (Theorem 3.3) on the metric VC dimension of t-sparse polynomials (independent of the degree) implying Corollary 3.5.

COROLLARY 3.5. *The polynomial regression can be estimated uniformly for small samples (depending only on the sparsity of the regression).*

4. Further research.

4.1. Learnability of multivariate polynomials. From [19] degree-bounded multivariate polynomials are of finite VC dimension for any fixed number of variables. There is no corresponding result for sparse multivariate polynomials. As described in §2, the main tool for proving the finiteness of the VC dimension in the sparse univariate case is the upper bound on the number of roots of sparse polynomials derived from Descartes' Rule. A promising approach for the multivariate case might be the work of Khovanskii [12]. Khovanskii generalizes Descartes' estimate to the sparse multivariate case and proves that the number of nondegenerated solutions of a system of sparse polynomial equations can be bounded in terms of the sparsity and the number of variables. In spite of these results, it is not clear in the multidimensional case how to relate the VC dimension to an upper bound on the number of common roots of sparse multivariate polynomials.

4.2. Efficient learning algorithms. It is an open problem whether there exists a hypothesis finder (i.e., an algorithm that finds a concept consistent with a given sample) for the class of t-sparse polynomials such that the time complexity of the algorithm depends only on their sparsity and the sample size (see §4).

A related question is the problem of whether or not the class of univariate polynomials is learnable with respect to *target complexity* (see [3]), where the complexity of a polynomial is given by its sparsity. The results of Linial, Mansour, and Rivest [14] imply that this is equivalent to the question of whether or not the class of sparse polynomials is polynomially uniformly decomposable. This may be reduced to the problem of the existence of a polynomial-time algorithm for sparse solutions of a linear-programming problem. Note that the existence of such an algorithm would not imply polynomial learnability of the class of t-sparse polynomials for fixed t since the appropriate degree is unknown.

Acknowledgment. We thank Manuel Blum, Allan Borodin, Sally Floyd, Dima Grigoriev, Les Valiant, and Manfred Warmuth for a number of interesting conversations. The discussion with Vladimir Vapnik led us to the solution of the general regression problem.

REFERENCES

[1] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, in Proc. 20th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1988, pp. 301–309.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, *Classifying learnable geometric concepts with the Vapnik–Chervonenkis dimension*, in Proc. 18th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1986, pp. 273–282.

[3] ———, *Learnability and the Vapnik–Chervonenkis dimension*, J. Assoc. Comput. Mach., 36 (1989), pp. 929–965.

[4] J. Bruck, *Harmonic analysis of polynomial threshold functions*, SIAM J. Discrete Math., 3 (1990), pp. 282–287.

[5] J. Bruck and R. Smolensky, *Polynomial threshold functions, AC^0 functions, and spectral norms*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1990, pp. 632–641.

[6] S. Floyd, *On Space-bounded Learning and the Vapnik–Chervonenkis Dimension*, Ph.D. thesis, University of California, Berkeley, CA, 1989.

[7] D. Grigoriev and M. Karpinski, *The matching problem for bipartite graphs with polynomially bounded permanent is in NC*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1987, pp. 166–172.

[8] D. Grigoriev, M. Karpinski, and M. Singer, *Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields*, SIAM J. Comput., 19 (1990), pp. 1059–1063.

[9] ———, *Interpolation of sparse rational functions without knowing bounds on exponents*, SIAM J. Comput., 24 (1994), to appear.

[10] D. Haussler, *Generalizing the pac model: Sample size bounds from metric dimension-based uniform convergence results*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1989, pp. 40–45.

[11] M. Karpinski, *Boolean circuit complexity of algebraic interpolation problems*, in Proc. 2nd Workshop on Computer Science Logic, Lecture Notes in Computer Science 385, Springer-Verlag, Berlin, 1989, pp. 138–147.

[12] A. Khovanskii, *Fewnomials and Pfaff manifolds*, in Proc. International Congress of Mathematics, Warsaw, 1983.

[13] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1991, pp. 455–464.

[14] N. Linial, Y. Mansour, and R. Rivest, *Results on learnability and the Vapnik–Chervonenkis dimension*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1988, pp. 120–129.

[15] L. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[16] V. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, Berlin, 1982.

[17] ———, *Inductive principles of the search for empirical dependences* (*methods based on weak convergence of probability measures*), in Proc. 2nd Workshop on Computational Learning Theory, 1989, pp. 3–21.

[18] V. Vapnik and A. Chervonenkis, *On the uniform convergence of relative frequencies of events and their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.

[19] R. Wenocur and R. Dudley, *Some special Vapnik–Chervonenkis classes*, Discrete Math., 33 (1981), pp. 313–318.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1286–1302, December 1993

COMPUTING A FACE IN AN ARRANGEMENT OF LINE SEGMENTS AND RELATED PROBLEMS*

BERNARD CHAZELLE†, HERBERT EDELSBRUNNER‡, LEONIDAS GUIBAS§, MICHA SHARIR¶, AND JACK SNOEYINK[1]

Abstract. This paper presents a randomized incremental algorithm for computing a single face in an arrangement of n line segments in the plane that is fairly simple to implement. The expected running time of the algorithm is $O(n\alpha(n)\log n)$. The analysis of the algorithm uses a novel approach that generalizes and extends the Clarkson–Shor analysis technique [in *Discrete Comput. Geom.*, 4 (1989), pp. 387–421]. A few extensions of the technique, obtaining efficient randomized incremental algorithms for constructing the entire arrangement of a collection of line segments and for computing a single face in an arrangement of Jordan arcs are also presented.

Key words. computational geometry, arrangements, randomized incremental algorithms, probabilistic backwards analysis, Davenport–Schinzel sequences

AMS subject classifications. 68P05, 68Q20, 68R99, 51M99

1. Introduction.

We consider the following problem. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a collection of n line segments in the plane, and let p be a point not lying on any of the segments. We wish to compute the face that contains p in the arrangement $\mathcal{A}$ of S. This problem arises in many applications, such as motion planning [9]. It has been shown in [9], [15] that the combinatorial complexity of such a single face is $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. This bound is shown in [19] to be tight in the worst case; as a matter of fact, the construction in [19] gives a set S of n line segments whose lower envelope has complexity $\Omega(n\alpha(n))$.

The problem of computing a single face has been studied by Edelsbrunner, Guibas, and Sharir [6]; they have given a deterministic algorithm that takes time $O(n\alpha(n)\log^2 n)$ in the worst case. This is less efficient than the best-known algorithm for computing the envelope of n segments, due to Hershberger [10], which runs in optimal $O(n \log n)$ time. This discrepancy between the two algorithms is intriguing because the maximum combinatorial complexity of a single face and of the lower envelope in an arrangement of n segments is asymptotically the same. We remark that in the special case where S is a collection of lines, computing a single face can be trivially done in time $O(n \log n)$. Another special case is when S is a collection of rays. A recent paper [1] shows that the complexity of a single face in this case is $O(n)$ and that the face can be constructed in time $O(n \log n)$. Both these algorithms are deterministic.

In this paper we (almost) close the gap by providing a simple randomized incremental algorithm for computing a single face in an arrangement of general segments, whose expected

*Received by the editors June 27, 1991; accepted for publication (in revised form) July 21, 1992. A preliminary version of this paper appeared in *Proc. 2nd ACM–SIAM Symposium on Discrete Algorithms*, Association for Computing Machinery, New York, 1991, pp. 441–448.

†Department of Computer Science, Princeton University, Princeton, New Jersey 08540. The research of this author was supported by National Science Foundation grant CCR-87-00917.

‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. The research of this author was supported by National Science Foundation grant CCR-89-21421.

§DEC Systems Research Center, Digital Equipment Corporation, Palo Alto, California, 94301, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, and Department of Computer Science, Stanford University, Stanford, California 94305.

¶School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The research of this author was supported by Office of Naval Research grant N00014-90-J-1284, by National Science Foundation grants CCR-89-01484 and CCR-91-22103, and by grants from the U.S.–Israeli Binational Science Foundation, the Fund for Basic Research administered by the Israeli Academy of Sciences, and the German-Israeli Foundation for Scientific Research and Development.

[1]Department of Computer Science, Stanford University, Stanford, California 94305.

running time is $O(n\alpha(n)\log n)$. (The expectation is taken over the randomizations used by the algorithm, and the bound holds for any input data.) We have learned that Clarkson has obtained a similar result in an unpublished work, using a different approach. The algorithm is similar in some features to the trapezoidal decomposition algorithm of [18], the intersection algorithms of [2] and [14], and the Delaunay triangulation algorithm of [8]. Like the latter, it is a purely on-line algorithm that needs no prior information about the as-yet uninserted segments. We also mention that recently Mitchell [13] has obtained a deterministic algorithm for constructing a single face, whose running time is $O(n\log^2 n)$.

A main novel feature of our algorithm is its analysis, which provides a useful extension of the probabilistic technique of Clarkson and Shor [5] to a domain where the interesting events that need to be counted are more difficult to specify. The reason is that the decision of what features of the arrangement of the segments in S appear on the desired face is global and cannot be determined from the local structure of the features. Such a locality is required in Clarkson and Shor's analysis and, for that matter, in all the randomized algorithms we have mentioned. Our analysis finesses this issue by applying a more general framework, which, as a consequence, also leads to simplified proofs. We expect that there will be additional applications of our technique to other contexts, thus extending the usefulness of the Clarkson–Shor method.

Our technique also can be generalized to other contexts, as discussed in §4. These problems include the construction of the entire arrangement of a given collection of line segments (§4.1) and computing a single face in an arrangement of curved segments (§4.2). In these extensions our technique yields algorithms with optimal or close-to-optimal expected time and storage complexities, matching or improving previously known algorithms. Section 2 presents the incremental algorithm, developing it to a level of detail that shows that it is indeed easy to implement. Section 3 gives the analysis of the algorithm. We conclude the paper in §5 with a discussion of our results and some open problems.

2. The algorithm. As mentioned in the introduction, the algorithm to be described in this section is incremental, that is, it computes the desired face by adding the segments one at a time. Section 3 will show that if the segments are inserted in a random order, then the expected behavior of the algorithm is very good. We describe the algorithm in detail, to convince the reader that the algorithm is easy to implement. A compact description of the algorithm in pseudocode is given at the end of this section. Let $s_1, s_2, \dots, s_n$ be the insertion sequence, so that at the ith step the algorithm adds s_i to the data structure built for s_1 through s_{i-1}. For convenience we start with a rectangular frame big enough to enclose all line segments in S, as well as the special point p defining the face f that we want to compute. We will be interested only (without loss of generality) in the portion of f within the frame. For $0 \le i \le n$ let f_i denote the face in the arrangement defined by $s_1, s_2, \dots, s_i$ that contains p, clipped to within the frame (f_0 is just the frame). We also assume that there are no degenerate cases, such as three segments meeting at a point, an endpoint of one segment lying on another segment, or two intersections with the same x coordinate; this assumption is justified by the algorithmic method of [7].

Although the face f_i is uniquely determined by the first i line segments, the data structure that we use to represent it is not — it also depends on the sequence in which the line segments are added. This is very much like in the case of a binary search tree constructed by repeated insertions, but without a balancing operation: the sorted sequence of the input is unique, but the tree that represents it depends on the sequence of insertions.

The main idea that leads to the data structure and algorithm of this paper is that while the central aim is to construct the face marked by p, we keep around everything ever built (typically portions of the earlier versions of the same face) as an aid in the search operations.

An important rule is that these older parts of the structure are not further refined during the insertion process — this helps keep the size of the extra structure within limits.

2.1. The conceptual level. The data structure that represents f_i, the face after adding the first i line segments, consists of three sorts of geometric information. These three parts should be considered as fundamentally different at a conceptual level, although we will represent them in a uniform way at a lower level. The three parts are the *city* (the face), the *suburbs* (the complement of the face), and the *history*.

2.1.1. The city. After i line segments are added, the face f_i is the city. It is a (not necessarily simply connected) polygonal region, as shown in Fig. 1. Its boundary consists of a finite number of contour cycles; one is the outer cycle (which, in case f_i is unbounded, coincides with the frame boundary), and all others define holes in the city. We represent the city by a collection of trapezoids generated by drawing a vertical line up and down from each vertex until it hits the boundary of the city again. These vertical edges, called *sides*, are drawn only inside the city—see Fig. 2. Two trapezoids are said to be adjacent if they (partially) share a vertical side.

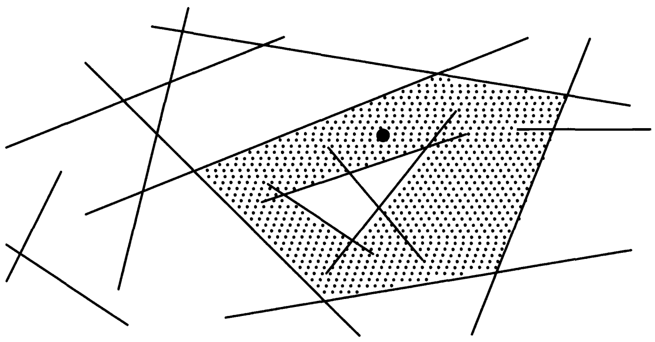


FIG. 1. *The input consists of a set of line segments and a point inside a frame. It defines a face, which we call the city.*

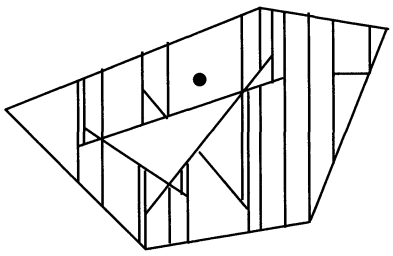


FIG. 2. *The city is decomposed into trapezoids by drawing vertical sides through endpoints and intersection points.*

There is a small number of different types of trapezoids, each defined by at most four line segments. A unique line segment contributes the *floor* (the bottom edge) of a trapezoid Δ, and, similarly, a unique line segment contributes the *ceiling* (the top edge). The left and right sides are each defined

(a) by an endpoint of another line segment,

(b) by another line segment intersecting the floor line segment,

(c) by another line segment intersecting the ceiling line segment, or

(d) as the intersection of the floor with the ceiling.

This makes 16 types of trapezoids altogether. One of the types is impossible, namely, where both the left and the right sides are case (d). Four of the 15 remaining types are shown in Fig. 3.

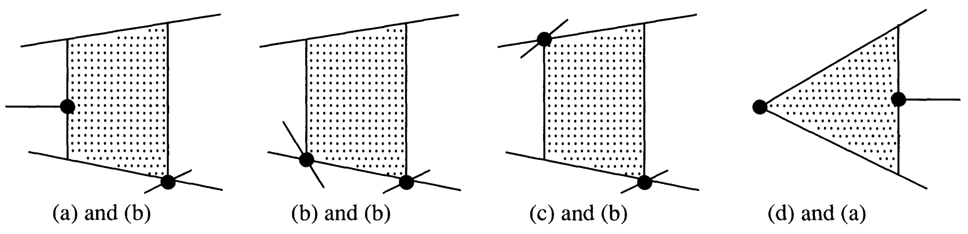


FIG. 3. *Four possible types of trapezoids.*

A trapezoid thus defined has one, two, three, or four adjacent trapezoids; four only if both the left and the right sides are case (a). Notice that the trapezoids that compose the city at stage i depend only on the set of segments $s_1, s_2, \ldots s_i$ and not on the particular order in which these segments were inserted.

2.1.2. The suburbs. As new line segments are added the city gets smaller. Each new line segment may chop off parts of the city by separating them from the point p. When a portion of the city is thus disconnected from p, it is properly decomposed into trapezoids and these trapezoids are added to the representation of the complement of the city, the so-called suburbs. It is thus natural to represent the suburbs in the same way as the city, namely, as a collection of trapezoids with adjacency relations.

At any point in time the trapezoids of the city and the suburbs define a decomposition (a tiling) of the entire frame. It should be noted, however, that this decomposition is not edge-to-edge, in the sense that a vertex of some trapezoid may lie in the middle of an edge of another trapezoid. We view each edge of our diagram as two sided, so that the above vertex is not part of the description of the second trapezoid — each trapezoid is bounded by one-sided edges. The same distinction was necessary in the analysis of [14]. There is, however, an important difference between vertical and nonvertical edges. By our general position assumptions, at most one point of a left or right side can also be a corner of (two) other trapezoids (in case (a)), but arbitrarily many such points can lie on the floor or ceiling. For this reason we define and store adjacencies only across vertical sides.

An important difference between city and suburbs is that the former gets further refined as new line segments are added, while the latter only expands by the addition of new trapezoids chopped off from the city. A trapezoid of the suburbs, once created, remains part of the suburbs forever.

2.1.3. The history. There is a third type of trapezoid in our structure. This type consists of trapezoids that belonged to earlier versions of the city and had to be removed because they were cut by a new line segment. Such a trapezoid Δ is not deleted from the structure. Instead, it remains as part of the history. The new trapezoids, generated by the addition of the new line segment, that overlap Δ are added to the structure as children of Δ. Depending on how the new line segment cuts Δ, it can have two, three, or four children (see Fig. 4). In effect, Δ is removed from the representation of the city and is now part of a hierarchical structure of trapezoids built on top of the decomposition described as city plus suburbs. As will be detailed in the following, certain children trapezoids are merged with adjacent children trapezoids of neighbor trapezoids.

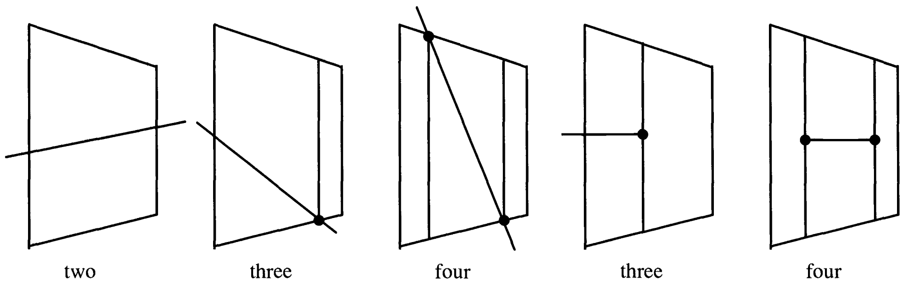


FIG. 4. *The different ways in which a trapezoid can be split by a new line segment.*

The collection (hierarchy) of such trapezoids that once belonged to the city but were later destroyed is called the *history*. The history trapezoids, together with those defining the city and suburbs, are all connected together by the children pointers in a directed acyclic graph. (Because of the merging of children trapezoids that was previously mentioned, the graph is not necessarily a tree.)

2.1.4. Adding a line segment. To understand exactly how our data structure looks at any stage of the incremental process, we need to understand how a line segment, say, s_{i+1}, is added. As mentioned earlier, already existing trapezoids of the suburbs and the history are unaffected by this insertion.

Here is how the city trapezoids are updated. First, we compute and draw $f_i \cap s_{i+1}$, which is a collection of portions (*edges*) of s_{i+1}. These new edges make it necessary to update the decomposition of f_i: the trapezoids of f_i that intersect s_{i+1} become part of the history, and the new trapezoids generated are included in the structure as their children. To understand this process, let us define the *transient* city g_i as f_i after s_{i+1} has been added and the decomposition of f_i into trapezoids has been updated. Of course, g_i may contain several trapezoids, some newly created and some pre-existing as part of the city at stage i, that are no longer accessible from p and thus not part of f_{i+1}. To obtain f_{i+1} we must thus remove all these trapezoids from g_i and place them in the suburbs. Figure 5 shows the development of the suburbs when the line segments are added in the indicated sequence.

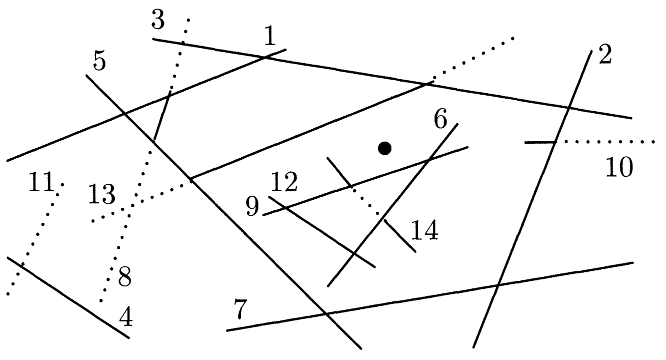


FIG. 5. *The subdivision is constructed by inserting the line segments in the indicated sequence. Line segment* 11 *is not drawn at all because it lies completely outside the city at the time it is added. Only portions of line segments* 8, 10, 13, *and* 14 *are drawn. The boundary of the final city consists of two contour cycles.*

To help us reflect on the process of adding line segments and updating the structure, let us look at what distinguishes suburb trapezoids from history trapezoids. For a trapezoid Δ

defined by line segments s_a, s_b, s_c, and s_d to be part of the suburbs or the history after the first $i + 1$ line segments are added, it must have occurred as part of at least one of the cities f_j or g_j for $1 \leq j \leq i$.

1. If there is a j, $j \leq i$, so that Δ is part of f_j but not part of g_j, then Δ is now part of the history because it was cut by s_{j+1} while being a city trapezoid.

2. If there is a j, $j \leq i$, so that Δ is part of g_j but not part of f_{j+1}, then it is now part of the suburbs because it was cut off from the city by s_{j+1}.

There is a subtlety in condition 2, which we take the time to discuss now. The trapezoid Δ can be of the type that belongs to f_j and to g_j but not to f_{j+1}, or it can be of the type that belongs to g_j but neither to f_j nor to f_{j+1}. In the latter case we call Δ a *transient* trapezoid because it lives a particularly short life. It will be important later to remember that transient trapezoids can only be part of the suburbs, not of the history.

2.2. The data structure. We will now be more specific about the data structure that is incrementally constructed by the algorithm. It consists of a directed acyclic graph (a dag) that stores the city, the suburbs, and the history, all at once, a linear array for the line segments; and a union-find structure for the line segments. We discuss the easy structures first.

2.2.1. The linear array. By keeping the line segments in a linear array we can use a single index rather than four real numbers wherever a line segment is to be stored. We assume that the segments are stored in the array in their insertion order.

2.2.2. The union-find structure. This structure allows us to keep track of topological changes that happen to the boundary of the city as line segments are added. Each set in the structure represents a connected component of the union of line segments and portions of line segments as drawn by the algorithm. Although a single line segment can have several disjoint portions drawn, they all belong to the same connected component. We can thus represent such a component by the set of line segments that contribute edges to it. Note that each contour cycle is part of a possibly bigger connected component. However, we will need the union-find structure only to the extent that it represents contour cycles. We will use a simple union-find structure, in which every element (segment) has a pointer to its current subset (contour cycle), so that each find operation takes $O(1)$ time. To form the union of two subsets we change the pointers of all the elements in the smaller set to be the same as those of the elements in the larger set. The overall cost of all unions is thus $O(n \log n)$.

2.2.3. The dag. Each node of the dag stores a unique trapezoid (city, suburbs, or history), represented by four indices (line segments) and a few bits to indicate the type. The dag has a unique root that stores the frame as a single trapezoid. Each interior node stores a history trapezoid and contains pointers to its (at most) four children. The city and suburb trapezoids are stored in the leaves of the dag, and each leaf has pointers to the at-most four leaves storing adjacent trapezoids. To distinguish the three types of nodes we mark history and suburb trapezoids as such and leave city trapezoids unmarked.

2.3. How it really works. Recall the basic steps that have to be performed when a line segment s_{i+1} is added.

1. We compute all portions of $s_{i+1} \cap f_i$.

2. Using these portions, we update the trapezoidal decomposition of f_i to get g_i. Destroyed trapezoids become history.

3. The new city f_{i+1} is the component of g_i that contains p. All other trapezoids in g_i need to be labeled as suburbs.

The portions of $s_{i+1} \cap f_i$ are computed by propagating s_{i+1} from the root of the dag down to the leaves. Each trapezoid of f_i intersected by s_{i+1} is updated, and the new city and suburbs are differentiated with the help of the union-find structure. Here are the details of how steps 1 through 3 are implemented.

2.3.1. Intersecting the new line segment with the city. Starting at the root of the dag, the line segment s_{i+1} is propagated downward to all leaves whose trapezoid meets s_{i+1}. When we are at an internal node ν, we know that s_{i+1} meets the history trapezoid of ν and we mark ν as already visited. Next we recursively visit the children of ν whose trapezoids meet s_{i+1} and that are not yet marked. The order in which we visit them is such that they meet s_{i+1} in sequence from left to right. Because of this ordering, the leaves are also visited in the sequence in which their trapezoids meet s_{i+1} from left to right.

2.3.2. Updating the trapezoidal decomposition. When a leaf storing a suburb trapezoid is reached, we do nothing. When a city trapezoid is reached, we do all the work. We distinguish six cases as illustrated in Fig. 6. We denote the leaf by λ.

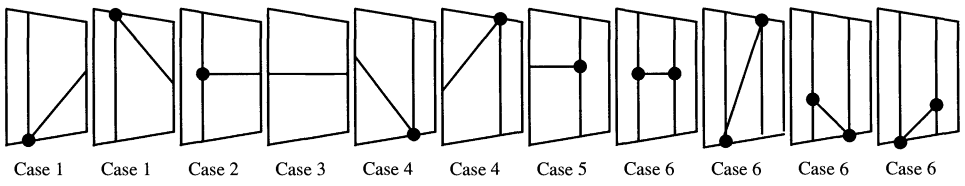


FIG. 6. *Updating the trapezoidal decomposition.*

In every case we construct the appropriate number of children, change λ from city to history, and use λ's former adjacency pointers to connect it to its children. In case 1, depending on whether the endpoint or intersection point that defines the right side of the old trapezoid is above or below s_{i+1}, one of the two children trapezoids that lie above and below s_{i+1} is not a properly defined trapezoid yet. This trapezoid will be merged with the adjacent child trapezoid of the next leaf. The same is true in case 2. The only difference between the two cases is that in case 1 we remember the line segment s_a that contains the left endpoint of the currently processed portion of $s_{i+1} \cap f_i$ (s_a contains the top or bottom edge of λ) and the two children of λ that lie above and below the current portion of $s_{i+1} \cap f_i$. In case 3 one of the child trapezoids is merged with a child trapezoid of the preceding leaf, and the same happens in cases 4 and 5. In case 4 we also take note of the line segment s_b that contains the right endpoint of the current portion of $s_{i+1} \cap f_i$. The pair (s_a, s_b) delimits this portion. The pair will be processed as described below. Finally, case 6 is in a way the easiest, because it requires only the construction of the four children for λ and no merging of trapezoids (nodes) is necessary. If, in case 6, s_{i+1} meets both top and bottom edges of λ, we immediately obtain the corresponding delimiting pair (s_a, s_b). In all other subcases this pair is undefined.

Let us say a few more words about the merging of children trapezoids. As we follow s_{i+1} from left to right, we maintain the current two children trapezoids that lie above and below s_{i+1}. One of these children may be open-ended on the right. When we reach a trapezoid λ and we are in case 3, 4, or 5, we extend the open-ended trapezoid (if any) and merge it with the appropriate child of λ. In case 3 we exit λ on the right with one of the children trapezoid closed and one open ended, as appropriate; in cases 4 and 5 both are closed. In cases 1 and 2 we create two new children accompanying s_{i+1} and leave one of them closed and one open ended, as above. In case 6, as mentioned above, no merging of children is necessary.

2.3.3. Maintaining the topology. After all portions of $s_{i+1} \cap f_i$ are added to the city decomposition as described, we have effectively obtained the trapezoidal decomposition of the transient city g_i. As a by-product, for each portion of $s_{i+1} \cap f_i$ we also get a pair (s_a, s_b)

of line segments that delimit the portion, and two trapezoids, one that lies immediately above it and one immediately below it. This extra information is not properly defined, but it is not needed anyway if the portion contains one of the endpoints of s_{i+1}.

For each such pair (s_a, s_b) we do the following. First we compute c_a and c_b, the names of the connected components containing s_a and s_b, respectively, by doing two find operations. If the two components are different, then we just have to union the two components to reflect the fact that the new segment s_{i+1} has merged the two contours into one; in this case the current portion of $s_{i+1} \cap f_i$ does not disconnect any portion of f_i from p. If $c_a = c_b$, i.e., the two contours are the same, then we have to work harder because the old city area on one side of the current portion of s_{i+1} now becomes suburb. It is not possible to decide locally which side this is. We thus perform two graph traversals in lock-step, starting at the two trapezoids (nodes) provided with s_a, which are trapezoids that lie on the two sides of the current portion of s_{i+1}. These traversals use the adjacency pointers and advance in a strictly alternating fashion, one trapezoid at a time. The traversals stop when one region is exhausted without finding the trapezoid that contains p (the exhausted region is now suburb and its trapezoids must therefore be relabeled) or when the trapezoid containing p is found (in this case the other region becomes suburb and its trapezoids must be relabeled). In either case the amount of time spent is at most proportional to the number of city trapezoids that became suburb.

Up to minor details, such as the fact that s_{i+1} should be added to the proper contour cycle or start a new one of its own, this concludes the description of the algorithm. For the convenience of the reader, we summarize the algorithm in pseudocode.

```
procedure $face(p, S)$;            % $p$ is a point and $S$ is a set of segments
    initialize $dag$ to a single node containing the enclosing frame;
    store a random permutation of $S$ in an array $[s_1, s_2, \ldots, s_n]$;
    initialize a union-find data structure on the segments, each
      stored as a singleton set;

    for all $i = 1, \ldots, n$ do
        perform a depth-first search of $dag$ to find all trapezoids crossed by $s_i$:
            construct a list $x\_trapezoids$;
            visit children of each node of $dag$ that $s_i$ crosses in
              left-to-right order (along $s_i$);
            mark each visited node (so as not to visit it again);
            add each city leaf crossed by $s_i$ to $x\_trapezoids$;

        update the trapezoidal decomposition:
            for each trapezoid $\lambda$ in $x\_trapezoids$ do
                depending on the type of $\lambda$ do
                case 1:
                    initialize $top\_trap$ and $bot\_trap$ to the subtrapezoids of $\lambda$
                      lying above and below $s_i$, respectively;
                    mark which of the two is open-ended on the right and which
                      is closed;
                    change $\lambda$ to a history node in $dag$;
                    add the 3 newly created subtrapezoids to $dag$;
                    store pointers from $\lambda$ to them;
                    the type of the new nodes (city/suburb) is not set as yet;
                    store adjacency pointers between the new nodes as appropriate
                      (the left subtrapezoid also inherits the left-adjacency
        pointer(s) from $\lambda$);
      set $s_a$ to the segment containing the top or bottom portion
        of $\lambda$, whichever $s_i$ intersects;
      set $\tau_1$ and $\tau_2$ to *top_trap* and *bot_trap*, respectively;
  case 2:
      proceed as in case 1 except for setting $s_a$, $\tau_1$, and $\tau_2$;
      both *top_trap* and *bot_trap* are now adjacent to the left
        subtrapezoid of $\lambda$;
  case 3:
      update *top_trap* and *bot_trap* by appending to the open-ended among
        them the corresponding top or bottom subtrapezoid of
        $\lambda$ and setting the remaining variable to the other
        subtrapezoid of $\lambda$;
      again mark which of the two is now open-ended and which is closed;
      add to *dag* the subtrapezoid that is not appended;
      change $\lambda$ to history;
      store pointers from $\lambda$ to these two subtrapezoids;
      store adjacency pointers between the new node of *dag* and the
        subtrapezoid preceding it on the left;
  case 4:
      update *top_trap* and *bot_trap* as in case 3;
      both resulting subtrapezoids are now closed;
      update *dag* as described;
      also, add the right subtrapezoid of $\lambda$ to *dag*;
      store a pointer to it from $\lambda$;
      store adjacency pointers between the subtrapezoids as appropriate
        (the right subtrapezoid also inherits the right-adjacency
        pointer(s) of $\lambda$);
      set $s_b$ to the segment containing the top or bottom portion
        of $\lambda$, whichever $s_i$ intersects;
  case 5:
      proceed as in case 4, except for setting $s_b$;
      the right subtrapezoid is now adjacent to both *top_trap*
        and *bot_trap*;
  case 6:
      construct the 4 subtrapezoids of $\lambda$, all closed, as new nodes of *dag*;
      change $\lambda$ to history;
      add pointers from $\lambda$ to these 4 nodes;
      store adjacency pointers between these nodes as appropriate
        (including the carrying over of adjacency pointers of $\lambda$);
      if $s_i$ crosses the top (respectively, bottom) edge of $\lambda$, set $s_a$
        (respectively, $s_b$) to the segment containing that edge;
  end case;

  if either $s_a$ or $s_b$ is undefined then
      mark all new nodes of *dag* as city;
  else
      find $s_a$ and $s_b$ in the union-find structure;
      if $s_a$ and $s_b$ are in different subsets then
```

```
                    union these subsets;
                    mark all new nodes of $dag$ as city;
                else update the topology of the face:
                    perform simultaneously, in lock-step, two searches of $dag$,
                      starting at $\tau_1$, $\tau_2$, respectively, and following the adjacency
                      pointers, until either one search is exhausted or the
                      trapezoid containing $p$ is encountered;
                    in either case, mark all nodes encountered in one search as city
                      nodes and in the other search as suburb nodes, as appropriate;
                end if;
            end if;
            if either $s_a$ or $s_b$ is defined then
                union $s_i$ with $s_a$ or with $s_b$, whichever is
                  defined;
            end if;
        end for;
    end for;

    return all city leaves of $dag$;

  end procedure;
```

Remark. After completing the algorithm we notice that suburb trapezoids are fairly useless when we add line segments. We could prune the dag by removing all leaves that store suburb trapezoids and, recursively, all nodes storing history trapezoids that thus end up without children. However, the analysis in §3 will reveal that the savings possible by this optimization are not substantial (at least asymptotically).

3. The analysis. The algorithm presented in §2 is a purely on-line algorithm for the single-face problem. In this section we show that if the segments are inserted according to a random permutation, then the expected behavior of our algorithm is very good in terms of both time and storage. We remark that without the randomization there can be situations where the space and time performance of our algorithm become quadratic in n. Such a situation is shown in Fig. 7.

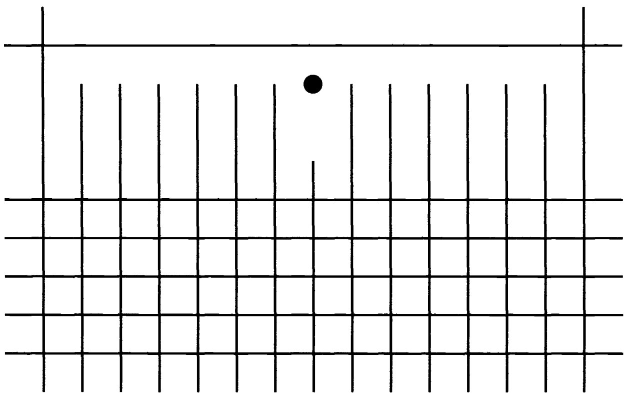


FIG. 7. *An example where our algorithm will require quadratic time if all the vertical line segments are inserted before the horizontal ones, which are then added from bottom to top.*

Recall that the main data structures used in our algorithm are a linear array, a union-find structure, and a dag. The sizes of the first two structures are proportional to n, the number of line segments. The size of the dag is proportional to the number of trapezoids constructed during the course of the algorithm. We will show in §3.1 that the expected number of trapezoids is $O(n\alpha(n))$.

The time spent by the algorithm is split among union-find operations, constructing trapezoids, searching for and labeling suburban trapezoids, and propagating the line segments down the dag. The cost of the union-find operations is at most $O(n \log n)$, even with a simple structure that supports $n - 1$ unions in amortized $O(\log n)$ time per operation and each find operation in constant time. By the results of §3.1 the expected number of find operations is $O(n\alpha(n))$, which thus takes expected time no more than $O(n\alpha(n))$. The same is true for constructing and labeling trapezoids because our lock-step search strategy ensures that the cost of these steps is proportional to the number of constructed trapezoids. Indeed, the cost of the lock-step search is easily seen to be proportional to the number of trapezoids in the smaller of the two face portions traversed and is thus proportional to the number of trapezoids that have been now disconnected from the city. Since a trapezoid can leave the city at most once, the claim follows. To understand the cost of propagating the line segments down the dag, let us define the *weight* of a trapezoid Δ, denoted $w(\Delta)$, as the number of line segments that intersect Δ. The cost is then proportional to $\sum w(\Delta)$, where the sum is taken over all trapezoids Δ constructed by the algorithm. We will show in §3.2 that the expectation of this sum is $O(n\alpha(n) \log n)$.

Hence, anticipating these results, we obtain the main result of the paper.

THEOREM 3.1. *Given a set of n line segments and a point in the plane, the algorithm of §2 constructs the face that contains the point in the arrangement of the line segments, in expected time $O(n\alpha(n)\log n)$ and expected space $O(n\alpha(n))$.*

3.1. The expected number of trapezoids. Before starting the probabilistic analysis, recall that the number of transient trapezoids (that is, trapezoids that are constructed but are never part of the city proper; see §2.1) cannot exceed the number of other trapezoids by more than a factor of 4. This is because all transient trapezoids belong to the suburbs and are therefore stored on the leaf level of the dag and because each inner node of the dag has at most four children. This observation allows us to consider only trapezoids that belonged to the city at the time they were created.

LEMMA 3.2. *The expected number of trapezoids constructed by the algorithm is $O(n\alpha(n))$.*

Proof. Fix a trapezoid Δ, and define the following two events: (i) $X_{r,\Delta}$: Δ is a trapezoid in f_r, which is the city as defined after adding the first r segments. (ii) $Z_{r,\Delta}$: Δ is a trapezoid in some f_i, for $0 \le i \le r$.

Clearly, $Z_{n,\Delta} = \bigcup_{r=0}^{n} X_{r,\Delta}$, and $\sum_\Delta \mathbf{P}[Z_{n,\Delta}]$ is the expected number of non-transient trapezoids constructed by the algorithm, where the sum is taken over all trapezoids Δ defined by at most four line segments each, as detailed in §2.1. By the remark before the lemma, $5\sum_\Delta \mathbf{P}[Z_{n,\Delta}]$ is an upper bound on the expected number of constructed trapezoids, whether transient or not.

Notice that a trapezoid Δ can be constructed only once; thus $\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}$ is nonempty for at most one r, namely, if Δ is constructed at the time the rth segment is added. Therefore, $Z_{n,\Delta}$ is the disjoint union of the events $\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}$, for $1 \le r \le n$. This is true for all trapezoids Δ, except for the frame, which is the only trapezoid of f_0, by definition. It follows that

$$\sum_\Delta \mathbf{P}[Z_{n,\Delta}] = 1 + \sum_\Delta \sum_{r=1}^{n} \mathbf{P}[\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}].$$

By the definition of conditional probability we have

$$\mathbf{P}[\overline{X}_{r-1,\Delta} \cap X_{r,\Delta}] = \mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \cdot \mathbf{P}[X_{r,\Delta}].$$

To estimate the conditional probability, we note that Δ is defined by at most four line segments and, if we assume that Δ is in f_r, then it was also in f_{r-1} if and only if the rth segment to be added was not one of these at most four segments. This implies

$$\mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \le \frac{4}{r}.$$

The preceding equations thus imply

$$\sum_{\Delta} \mathbf{P}[Z_{n,\Delta}] \le 1 + \sum_{\Delta} \sum_{r=1}^{n} \frac{4}{r} \mathbf{P}[X_{r,\Delta}] = 1 + \sum_{r=1}^{n} \frac{4}{r} \sum_{\Delta} \mathbf{P}[X_{r,\Delta}]. \tag{1}$$

However, $\sum_{\Delta} \mathbf{P}[X_{r,\Delta}]$ is the expected number of trapezoids in the city f_r, after r line segments have been added. By the results of [9], [15], [19], f_r can have at most $O(r\alpha(r))$ edges and, therefore, at most $O(r\alpha(r))$ trapezoids. This finally gives

$$\sum_{\Delta} \mathbf{P}[Z_{n,\Delta}] = \sum_{r=1}^{n} O(\alpha(r)) = O(n\alpha(n)). \qquad \square$$

Remark. The analysis just presented is fairly general, and so we would like to restate it in more abstract terms, which will be exploited in §4. In general, we have a set of n objects (line segments in our case) that we add incrementally in random order to form some structure (a single face in our case). This structure is represented as a collection of regions (trapezoids in our case), each defined by at most some constant number b of objects (4 in our case). Let $M(r)$ denote the expected number of regions composing the structure after r objects have been added. Then the expected number of regions ever constructed during the randomized incremental process is at most $\sum_{r=1}^{n} \frac{b}{r} M(r)$, provided that if a region is present in the structure after r steps and the rth object to be added is not one of the b objects defining the region, then the region was also present in the structure after the first $r-1$ objects had been added. (If each region is defined by exactly b objects, then the preceding sum is an exact expression for the expected number of regions.) As an example, we apply this observation to the case in which the objects are n points in the plane, the structure is their Delaunay triangulation, and the regions are Delaunay triangles. This fits well into the setup just discussed. Moreover, we know that $M(r)$ is $2r - h_r - 2$, where h_r is the expected number of vertices appearing on the convex hull of a random sample of r points of the given n. We thus conclude that the expected number of Delaunay triangles constructed during a randomized incremental algorithm is $\sum_{r=3}^{n} \frac{3}{r}(2r - h_r - 2)$. The same expression was recently derived in [20] by using a more involved analysis. This general framework has also been observed by Seidel [16]–[18] and by Mehlhorn [11] (see also [4]) and is referred to as "backwards analysis."

3.2. The expectation of the sum of weights. Recall that the weight of a trapezoid Δ, $w(\Delta)$, is defined as the number of line segments that intersect Δ. As in §3.1, we argue that for the purpose of proving an upper bound on the expectation of the sum of weights of all constructed trapezoids, it suffices to consider only nontransient trapezoids. To see this, let Δ be a transient trapezoid. Distribute its weight among all its parents in the dag so that the share of each parent does not exceed its original weight. This is possible because the union of the trapezoids of all parents of Δ contains Δ and therefore intersects at least as many segments as Δ does. Since any node in the dag has at most four children, its weight can thus go up by at most a factor of 5. Thus 5 times the expected sum of weights of all nontransient trapezoids is an upper bound on the expectation of the sum over all trapezoids.

LEMMA 3.3. *The expected sum of weights of all trapezoids constructed by the algorithm is $O(n\alpha(n)\log n)$.*

Proof. The expected sum of weights over all nontransient trapezoids constructed by the algorithm is equal to $\sum_\Delta w(\Delta)\mathbf{P}[Z_{n,\Delta}]$, where the sum is taken over all trapezoids Δ defined by at most four segments each, and $Z_{n,\Delta}$ is the event that, in the course of adding all line segments, Δ was constructed as a nontransient trapezoid, the same event as in Lemma 3.2. In addition to the events $Z_{r,\Delta}$ and $X_{r,\Delta}$ we define $Y_{r,\Delta}$: Δ is a trapezoid of f_r, and s_{r+1}, the line segment added next, is one of the $w(\Delta)$ segments that intersect Δ.

Note that if Δ is a trapezoid of f_r, then none of the segments intersecting Δ was chosen in the first r steps. So for $Y_{r,\Delta}$ to occur, given that $X_{r,\Delta}$ has occurred, we have to choose at the $(r+1)$th step one of these $w(\Delta)$ segments out of the remaining $n-r$ segments. Hence

$$\mathbf{P}[Y_{r,\Delta}] = \mathbf{P}[X_{r,\Delta}] \cdot \frac{w(\Delta)}{n-r}.$$

Observe also that

$$Y_{r,\Delta} \subseteq X_{r,\Delta} \cap \overline{X}_{r+1,\Delta},$$

and, in general, we may have proper inclusion, because Δ can be removed from the city also by a line segment that does not intersect Δ. Independent of whether proper or improper inclusion, this implies that

$$(2) \qquad \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[Y_{i,\Delta}] \le \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[X_{i,\Delta} \cap \overline{X}_{i+1,\Delta}] \le \sum_\Delta \mathbf{P}[Z_{r,\Delta}] = O(r\alpha(r)).$$

In other words, the expected number of trapezoids that become history during the first $r+1$ insertions is $O(r\alpha(r))$, which is clear because these trapezoids have to be constructed first, and the expected number of such trapezoids, over the course of the first r insertions, is $O(r\alpha(r))$, as shown in §3.1.

Now fix Δ, and recall from the proof of Lemma 3.2 that

$$\mathbf{P}[Z_{n,\Delta}] = \sum_{r=1}^{n} \mathbf{P}[\overline{X}_{r-1,\Delta} | X_{r,\Delta}] \cdot \mathbf{P}[X_{r,\Delta}] \le \sum_{r=1}^{n} \frac{4}{r}\mathbf{P}[X_{r,\Delta}].$$

This implies that

$$w(\Delta)\mathbf{P}[Z_{n,\Delta}] \le 4\sum_{r=1}^{n} \frac{n-r}{n-r} \cdot \frac{w(\Delta)}{r}\mathbf{P}[X_{r,\Delta}] = 4\sum_{r=1}^{n} \frac{n-r}{r}\mathbf{P}[Y_{r,\Delta}].$$

To simplify the notation we set $D_r = \sum_\Delta \mathbf{P}[Y_{r,\Delta}]$, and we can now write

$$(3) \qquad \sum_\Delta w(\Delta)\mathbf{P}[Z_{n,\Delta}] \le 4\sum_{r=1}^{n-1} \frac{n-r}{r} D_r = 4\sum_{r=1}^{n-1} \left(\frac{n-r}{r} - \frac{n-r-1}{r+1}\right) \sum_{i=1}^{r} D_i.$$

However, we have shown that

$$\sum_{i=1}^{r} D_i = \sum_\Delta \sum_{i=1}^{r} \mathbf{P}[Y_{i,\Delta}] = O(r\alpha(r)).$$

Hence we finally obtain

$$(4)\qquad \sum_{\Delta} w(\Delta)\mathbf{P}[Z_{n,\Delta}] \le 4\sum_{r=1}^{n-1} \frac{n}{r(r+1)} O(r\alpha(r)) = O(n\alpha(n)\log n),$$

as claimed. □

Remark. As in the remark at the end of §3.1, these calculations can also be extended to the more general setup discussed there. Specifically, if we denote by $S(r)$ the expected number of regions (trapezoids in our case) formed during the first r steps of the randomized process, then (1) implies that

$$(5)\qquad S(r) \le \sum_{j=1}^{r} \frac{b}{j} M(j)\,, \qquad r = 1, \dots, n,$$

where b, M are as defined in the previous remark. The analysis leading to equations (2)–(4) can then be generalized to yield a bound on $T(n)$, which is defined to be the expectation of the sum of weights of the regions ever formed by the algorithm, where the weight of a region is the number of objects that intersect it. That is, we obtain

$$\begin{aligned} T(n) &\le \sum_{r=1}^{n-1} \frac{bn}{r(r+1)} S(r) = \sum_{r=1}^{n-1} \frac{bn}{r(r+1)} \sum_{j=1}^{r} \frac{b}{j} M(j) \\ &= \sum_{j=1}^{n-1} \frac{b}{j} M(j) \sum_{r=j}^{n-1} \frac{bn}{r(r+1)} = \sum_{j=1}^{n-1} \frac{b^2 n M(j)}{j} \left(\frac{1}{j} - \frac{1}{n}\right), \end{aligned}$$

or

$$(6)\qquad T(n) \le \sum_{r=1}^{n} \frac{b^2(n-r)}{r^2} M(r).$$

4. Extensions. The technique presented in this paper is sufficiently general to be applicable to a variety of other related problems. In this section we present a few such applications. In §4.1 we extend the previous algorithm to compute the entire arrangement of n line segments, and in §4.2 we describe an algorithm for computing a single face in an arrangement of Jordan arcs. The overall strategy is similar to that described earlier, but there are certain additional technical details that are particular to the specific application. In each case we discuss in some detail these difficulties, the modifications to the algorithm that they require, and the analysis of the resulting modified algorithm.

4.1. Computing the entire arrangement of *n* line segments. We first consider a simple extension of our technique to the problem of calculating the entire arrangement of a collection of n line segments in the plane. This can be achieved by applying a simplified version of the technique of §2. In this case there is no need to distinguish between city and suburbs since every face of the arrangement needs to be constructed. Consequently, when a segment is added to the arrangement, all its portions are drawn and there is no need to maintain any face topology by means of a union-find structure. We leave it to the reader to work out the details of this modified and simplified algorithm. The analysis is also easy; it uses the general method described at the end of §3. In this case we have $b = 4$ (the maximum number of segments defining a trapezoid), and $M(r)$, the *expected* number of trapezoids forming the vertical decomposition of the arrangement of the first r segments that were inserted, is bounded by $O(r + K \cdot \frac{r^2}{n^2})$, where K is the total number of intersections between the given segments (see [5] for the simple proof of this bound). Hence the expected storage of the algorithm is

$$S(n) \le \sum_{r=1}^{n} \frac{4}{r} M(r) = O\left(\sum_{r=1}^{n}\left(1 + \frac{K}{n^2} r\right)\right) = O(n + K),$$

and the expected running time is

$$T(n) \le \sum_{r=1}^{n} \frac{16(n-r)M(r)}{r^2} = O\left(\sum_{r=1}^{n}\left(\frac{n-r}{r} + \frac{K(n-r)}{n^2}\right)\right) = O(n \log n + K).$$

We thus obtain an algorithm with optimal (expected) running time and storage. The same performance is achieved by the deterministic (but complicated) algorithm of [3] and by the alternative randomized algorithms of [5], [14] (a variant of the algorithm of [5] also achieves $O(n)$ working storage).

4.2. Computing a face in an arrangement of arcs. Let Γ be a collection of n Jordan arcs $\gamma_1, \ldots, \gamma_n$. We assume that the arcs have a simple shape, which means that any pair of them intersect in at most some fixed number s of points, that each arc consists of a small fixed number of x-monotone pieces, and that it takes constant time to perform any of the following primitive operations — finding the intersection points between a pair of arcs, decomposing an arc into its x-monotone pieces, intersecting an arc with a vertical line, and testing whether a given point lies above or below a given (x-monotone piece of an) arc. To simplify the description of the algorithm, we assume that each arc is already x-monotone; otherwise, we first decompose the arcs into x-monotone pieces and then apply the algorithm. We also assume that the arcs are in general position, in the spirit of the similar assumption we have made for line segments.

As above, let p be a given point not lying on any arc. Our goal is to compute the face in the arrangement of Γ that contains p. To compute the desired face, we apply the same scheme of §2, except that there are several new technical difficulties that need to be addressed. The face (city) and its complement (suburbs) are represented by their vertical decomposition into *pseudotrapezoids*, obtained, as was done earlier, by drawing vertical segments up and down from every endpoint and intersection point until they hit another arc. If general position is assumed, each pseudotrapezoid is defined by at most four arcs, two containing its top and bottom edges and two defining its left and right sides.

The data structures that we use are the same as those in §2 — the dag, the linear array, and the union-find structure. Searching in the dag for the pseudotrapezoids that intersect a newly inserted arc γ is trickier in this case because the intersection of γ with a pseudotrapezoid Δ can consist of several connected components (at most $s+1$ components, as is easily checked). In this case we expect the search through the dag to yield a partition of γ into a (sorted) list of subarcs, each of which is either contained in the suburbs (and is therefore not drawn at all) or intersects a single pseudotrapezoid of the current city. This list is initialized to consist only of γ itself and is refined during the search as follows. Any recursive step involves the processing of some history pseudotrapezoid Δ and some subarc γ' that is a connected component of $\gamma \cap \Delta$. We go over the (constant number of) children of Δ, and for each child Δ' we compute $\gamma' \cap \Delta'$. The constant number of resulting subarcs of γ' are sorted by x coordinate and replace γ' in the output list. The search now continues recursively at each of the new subarcs and at the pseudotrapezoid that contains it. Note that a node Δ of the dag may be visited several times during the search, each time with a different subarc γ'. However, it is easy to show that the cost of searching with γ in the dag is proportional to the number of pseudotrapezoids ever formed that are crossed by γ. At the end of the search we almost obtain the desired sorted partition of γ; since children pseudotrapezoids are merged, the final list of subarcs may contain pairs of adjacent subarcs that share an endpoint and are contained in the same pseudotrapezoid. An additional pass through the final list is needed to merge such pairs.

The remaining steps of the algorithm are the same (with certain trivial modifications) as those of the algorithm of §2. We leave it to the reader to fill in the details.

The analysis is also similar to that in §3. Using the general notations at the end of that section, we observe that in our case we have $b = 4$ (maximum number of arcs defining a pseudotrapezoid) and $M(r) = O(\lambda_{s+2}(r))$ (bound on the complexity of a single face in an arrangement of r arcs as above; see [9]). Then the expected storage of the algorithm is

$$S(n) \le \sum_{r=1}^{n} \frac{4}{r} M(r) = O(\lambda_{s+2}(n)),$$

and the expected running time, again dominated by the cost of the searches through the dag, is

$$T(n) = O\left(\sum_{r=1}^{n-1} \frac{n-r}{r^2} M(r) \right) = O(\lambda_{s+2}(n) \log n).$$

Hence we have the following theorem.

THEOREM 4.1. *Given a collection Γ of n arcs in the plane with the aforementioned properties and a point p not lying on any arc, the face of $\mathcal{A}(\Gamma)$ that contains p can be computed in randomized expected time $O(\lambda_{s+2}(n) \log n)$ and expected storage $O(\lambda_{s+2}(n))$.*

Remark. This result is an improvement of the previous (deterministic) algorithm of Guibas, Sharir, and Sifrony [9], whose running time is $O(\lambda_{s+2}(n) \log^2 n)$.

5. Discussion. In this paper we have presented a randomized incremental technique for computing a single face in an arrangement of line segments and for several related problems. The technique is a variant of several related recent randomized algorithms. It improves the running time of the previously best algorithms for these problems, and it is fairly simple to implement. The main characteristic of the technique is maintaining the history of the random process as a dag of trapezoids, which facilitates efficient location of the new segment to be inserted relative to the current version of the computed face. The analysis of the algorithm is also novel, in the sense that it extends the previous analysis technique of Clarkson and Shor, resulting in a simpler and more general approach.

The problems studied in this paper are only a sample of problems that can be solved efficiently by using our technique. In addition to the earlier algorithm of Guibas, Knuth, and Sharir [8] for computing Delaunay triangulations in the plane, there appeared, after the original preparation of this paper, a few related works that also apply this or closely related techniques. Among these we mention work by Seidel [18] for constructing trapezoidal decompositions of arrangements of nonintersecting line segments and applying them for efficient point location and triangulation of simple polygons and work by Miller and Sharir [12] for computing the union of fat triangles or of pseudodiscs.

There are several other problems that are likely to be amenable to the technique presented here. Among these we mention the problems of computing many faces in an arrangement of lines or of line segments, computing a single cell in an arrangement of triangles in 3-space, computing the zone of a plane in an arrangement of planes in 3-space, and computing many cells in such an arrangement of planes. In all these cases it is straightforward to design the general structure of an appropriate algorithm, along the lines of the algorithms we have described. It is also fairly easy to extend the analysis to obtain sharp bounds on the expected number of regions constructed by the algorithm and on the expected sum of their weights, appropriately defined. The difficulty in completing the algorithm usually lies in the subproblem of maintaining the topology of the constructed structure. For example, in computing a single cell in an arrangement of triangles in space, when we add a new triangle t, we need to determine the way in which it modifies the current cell, which seems to be considerably more difficult than the similar problem in two dimensions.

To conclude, we mention one final open problem, namely, to close the still remaining gap between the expected running time of our main algorithm, i.e., $O(n\alpha(n)\log n)$, and the lower bound of $\Omega(n \log n)$.

Acknowledgment. The authors wish to express their gratitude for the generous support and hospitality of the DEC Palo Alto Systems Research Center.

REFERENCES

[1] P. Alevizos, J. D. Boissonnat, and F. P. Preparata, *An optimal algorithm for the boundary of a cell in a union of rays*, Algorithmica, 5 (1990), pp. 573–590.

[2] J. D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec, *On-line geometric algorithms with good expected behaviours*, in Proc. Journées Géometriques Algorithmiques, INRIA, Sophia-Antipolis, June 1990, pp. 7–13.

[3] B. Chazelle and H. Edelsbrunner, *An optimal algorithm for intersecting line segments in the plane*, J. Assoc. Comput. Mach., 39 (1992), pp. 1–54.

[4] L. P. Chew, *The Simplest Voronoi Diagram Algorithm Takes Linear Expected Time*, Manuscript, 1988.

[5] K. Clarkson and P. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[6] H. Edelsbrunner, L. Guibas, and M. Sharir, *The complexity and construction of many faces in arrangements of lines and of segments*, Discrete Comput. Geom., 5 (1990), pp. 161–196.

[7] H. Edelsbrunner and E. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graphics, 9 (1990), pp. 66–104.

[8] L. Guibas, D. E. Knuth, and M. Sharir, *Randomized incremental construction of Voronoi and Delaunay diagrams*, Algorithmica, 7 (1992), pp. 381–413.

[9] L. Guibas, M. Sharir, and S. Sifrony, *On the general motion planning problem with two degrees of freedom*, Discrete Comput. Geom., 4 (1989), pp. 491–521.

[10] J. Hershberger, *Finding the upper envelope of n line segments in $O(n \log n)$ time*, Inform. Process. Lett., 33 (1989), pp. 169–174.

[11] K. Mehlhorn, Unpublished manuscript, 1990.

[12] N. Miller and M. Sharir, *Efficient Randomized Algorithms for Constructing the Union of Fat Triangles and of Pseudodiscs*, Manuscript, 1991.

[13] J. S. B. Mitchell, *On Computing a Single Face in an Arrangement of Line Segments*, Manuscript, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, July 1990.

[14] K. Mulmuley, *A fast planar partition algorithm* I, J. Symbolic Comput., 10 (1990), pp. 253–280.

[15] R. Pollack, M. Sharir, and S. Sifrony, *Separating two simple polygons by a sequence of translations*, Discrete Comput. Geom., 3 (1988), pp. 123–136.

[16] R. Seidel, *Small dimensional linear programming and convex hulls made easy*, Discrete Comput. Geom., 6 (1991), pp. 423–434.

[17] ———, *Backwards analysis of randomized geometric algorithms*, in New Trends in Discrete and Computational Geometry, J. Pach, ed., Springer-Verlag, Berlin, 1993, pp. 37–67.

[18] ———, *A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons*, Comput. Geom. Theory Appl., 1 (1991), pp. 51–64.

[19] A. Wiernik and M. Sharir, *Planar realization of nonlinear Davenport Schinzel sequences by segments*, Discrete Comput. Geom., 3 (1988), pp. 15–47.

[20] E. Yaniv. *Randomized Incremental Construction of Delaunay Triangulations: Theory and Practice*, M.Sc. thesis, Tel Aviv University, Tel Aviv, Israel, 1991.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1303–1318, December 1993

A BETTER HEURISTIC FOR PREEMPTIVE PARALLEL MACHINE SCHEDULING WITH BATCH SETUP TIMES*

BO CHEN†

Abstract. This paper addresses the problem of scheduling N jobs on M identical parallel machines with the objective of minimizing the makespan. Jobs are divided into B batches. A sequence-independent batch setup time on a machine is incurred whenever the machine starts its processing on or switches it from a job in one batch to a job in another batch. On the basis of a heuristic for this NP-hard problem proposed by Monma and Potts, a modified heuristic that requires the same implementing time $O(N + (M + B)\log(M + B))$ and is asymptotically optimal is presented. Furthermore, for a certain class of problems, which includes the case in which each batch contains a single job, it has the worst-case performance ratio $\tau_M = \max\left\{\frac{3M}{2M+1}, \frac{3M-4}{2M-2}\right\}$.

Key words. identical parallel machines, batch setup times, preemptive scheduling, heuristics, worst-case performance

AMS subject classification. 90B35, 68M20

1. Introduction. Suppose that a computer is presented with a collection of tasks (source-code programs), each with a known processing time (compilation plus execution). Each task has a requirement for a particular compiler to be in memory. If the proper compiler for a task is resident, the task may be instantly started; otherwise, the contents of memory are abandoned and a setup time is incurred while the proper compiler is loaded into memory. Only one compiler can be resident in memory at a time. Thus it may be advisable to continuously schedule several tasks requiring the same compiler's presence.

This sort of problem [2], among others (see, for example, [5]), motivates us to consider the following scheduling model with batch setup times.

We are given N jobs that are divided into B *batches* and are to be scheduled on M (≥ 2) identical parallel machines. Job j of batch b is available for processing at time zero and has a positive *processing time* p_{jb}. At any time, each machine can handle at most one job and each job can be processed by at most one machine. *Preemption*, which allows the processing of a job to be interrupted and resumed later on the same or on any other machine, is permitted. A nonnegative *setup time* s_b on a machine is necessary either when a job from batch b is processed first on the machine or when the machine switches from processing a job in batch b' ($\neq b$) to a job in batch b. Our objective is to find a schedule that minimizes the makespan.

The computational complexity of various scheduling problems with batch setup times is investigated by Bruno and Downey [2] and by Monma and Potts [5] and surveyed by Potts and Van Wassenhove [7]. It is shown that the preemptive parallel machine problem we are considering is NP-hard even when $M = 2$. Monma and Potts [6] propose a heuristic algorithm LSU (largest-setup-time list scheduling and splitting heuristic), which requires $O(N + (M + B)\log(M + B))$ time and generates a schedule for which the makespan does not exceed $\frac{6M-7}{4M-4}$, $\frac{5}{3} - \frac{1}{M}$ times that of an optimal schedule, respectively, for $M \leq 4$ and $M > 3$, where M is a multiple of 3. In this paper we modify the heuristic LSU to LBT (largest-batch-time list scheduling and splitting heuristic), which has the same time requirement as LSU but returns a better worst-case guarantee of $\tau_M = \max\left\{\frac{3M}{2M+1}, \frac{3M-4}{2M-2}\right\}$ and is asymptotically optimal.

In §2 a brief description of heuristic LBT is presented and some notation is introduced. In §3 we examine the asymptotic performance of LBT, which is very similar to the generalized bound on LPT sequencing [4]. The worst-case performance of LBT is analyzed in §§4 and 5.

*Received by the editors June 17, 1992; accepted for publication (in revised form) August 14, 1992.

†Econometric Institute, Erasmus University, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands (bchen@cvx.eur.nl).

Some conclusions and remarks are provided in §6. The appendix deals with some mathematical treatments for §5.

2. Heuristic LBT and notation. The largest-batch-time list scheduling and splitting heuristic, or LBT for short, is formulated as follows:

Phase I: *List scheduling.*

1. Group the jobs into batches and compute for each batch b, $1 \le b \le B$, the total processing time $P_b = \sum p_{jb}$, where the summation is over all jobs j of batch b. Place the batches in a list in the order $2s_1 + P_1 \ge \cdots \ge 2s_B + P_B$. Set $L_i := 0$ for $1 \le i \le M$.

2. Select a machine i for which L_i is as small as possible. Choose the first unscheduled batch b of the list, and assign it to machine i. Set $L_i := L_i + s_b + P_b$. Repeat this step until all batches are scheduled.

3. Relabel the machines so that $L_{i-1} \le L_i$ for $2 \le i \le M$. Set $m := 1$, $m' := M$, and $C_{\max}^{\text{LBT}} = 0$.

Phase II: *Splitting.*

4. Consider the last batch b that is assigned to machine m'. If $L_m + s_b \ge L_{m'}$, then set $C_{\max}^{\text{LBT}} := L_{m'}$ and stop. Otherwise, assign to machine m any processing that occurs after time $(L_m + s_b + L_{m'})/2$ on machine m': for machine m, all original setups and processing are rescheduled $(L_{m'} + s_b - L_m)/2$ units later, at time zero a setup for batch b commences, and from time s_b to $(L_{m'} + s_b - L_m)/2$ the processing transferred from machine m' is executed. If, through rescheduling, the preemption of some job j of batch b is necessary, then job j is scheduled first on machine m and, after insertion of idle time if necessary, last on machine m', to be completed at time $T_{m'} = \max\{(L_m + s_b + L_{m'})/2, s_b + p_{jb}\}$. Set $C_{\max}^{\text{LBT}} := \max\{C_{\max}^{\text{LBT}}, T_{m'}\}$.

5. If $m' - m = 2$, then set $C_{\max}^{\text{LBT}} := \max\{C_{\max}^{\text{LBT}}, L_{m'-1}\}$ and stop. If $m' - m = 1$ or if $L_{m'-1} \le C_{\max}^{\text{LBT}}$, then stop. Otherwise, set $m := m + 1$, $m' := m' - 1$ and go to step 4.

Let, with the machine relabeling produced after the first phase of LBT, the value of $C_{\max}^{\text{LBT}}$ be given by the completion time of machine m', where $m' \ge \lceil M+1 \rceil /2$. Let $m = M - m' + 1$, and let some job of batch b be scheduled last on machine m'. Let L_i' be the total load assigned to machine i, for $1 \le i \le M$, just before the batch b is scheduled in step 2 of LBT. Let $\mathcal{L}_k (k \ge 1)$ be such a set of jobs that when the algorithm LBT applies to $\mathcal{L}_k$, batch b is the kth one on machine m' after the first phase of LBT. Let $C_M(\mathcal{L})$ and $C_M^*(\mathcal{L})$ denote the makespan of LBT and the optimal makespan, respectively, for job set $\mathcal{L}$. If no confusion can arise, we use C and C^* instead. We assume that idle time has not been introduced by LBT in considering the machine pair (m, m') since otherwise we always have $C = C^*$. Let $\theta = L_m - L_{m'}'$. It is clear that $\theta \ge 0$ and that $L_m + s_b \ge L_{m'}$ is equivalent to $\theta \ge P_b$. All the upper bounds obtained in the theorems of the following sections are *tight*, which can be shown by various examples. Interested readers are referred to [3] for those examples.

3. Asymptotic performance. Let us begin with the following lemma.

LEMMA 3.1. *For any $M \ge 2$ we have*

$$C - C^* \le \frac{M-m}{M} s_b + \frac{m-1}{M} P_b \quad \textit{if } m = m' \textit{ or if } m < m' \textit{ and } \theta \ge P_b;$$

$$C - C^* \le \frac{M-m}{M} s_b + \frac{M-2m}{2M} P_b + \frac{4m-2-M}{2M} \theta \quad \textit{if } m < m' \textit{ and } \theta < P_b.$$

Sketch of proof. Case 1: $m = m'$. For $i = 1, 2, \ldots, m-1$, $L_i \ge L_m' = C - (s_b + P_b)$. Also, we have $L_i \ge C$ for $i \ge m$. Hence $\sum_{i=1}^{M} L_i + (m-1)(s_b + P_b) \ge MC$. Since $MC^* \ge \sum L_i$, we get $C - C^* \le \frac{m-1}{M}(s_b + P_b)$, as we desired. Therefore, we suppose $m < m'$.

Case 2: $\theta \geq P_b$. Then $C = L_{m'}$. Since, for $i = 1, 2, \ldots, m-1$, $L_i + \theta \geq L'_i + \theta \geq L_m$ and $L_m = C + \theta - (s_b + P_b)$, we have

$$\begin{aligned} MC^* &\geq \sum_{i=1}^{m-1} L_i + \sum_{i=m}^{m'-1} L_i + \sum_{i=m'}^{M} L_i \geq (M-m)L_m - (m-1)\theta + mL_{m'} \\ &= MC - (M-m)(s_b + P_b) + (M - 2m + 1)\theta. \end{aligned}$$

Using $\theta \geq P_b$, we get

$$C - C^* \leq \frac{M-m}{M}(s_b + P_b) - \frac{M-2m+1}{M}\theta \leq \frac{M-m}{M}s_b + \frac{m-1}{M}P_b.$$

Case 3: $\theta < P_b$. This case is similar to Case 2. □

COROLLARY 3.2.

$$C - C^* \leq \frac{M-1}{M}\left(s_b + \frac{P_b}{2}\right).$$

Proof. The proof is a matter of careful calculations. Interested readers are referred to [3] for details. □

COROLLARY 3.3. *If* $m = 1$, *then*

$$\begin{aligned} C - C^* &\leq \frac{M-1}{M}s_b \quad \textit{if } \theta \geq P_b, \\ C - C^* &\leq \frac{M-1}{M}s_b + \frac{M-2}{2M}(P_b - \theta) \quad \textit{otherwise.} \end{aligned}$$

With Lemma 3.1 and Corollaries 3.2 and 3.3 we can establish the following result.

THEOREM 3.4. *For* $M \geq 2$, $k \geq 3$, *we have*

$$\frac{C_M(\mathcal{L}_k)}{C^*_M(\mathcal{L}_k)} \leq \frac{Mk}{M(k-1)+1}.$$

Proof. Let the $(k-1)$ batches before batch b be $i_1, \ldots, i_{k-1}$. Then for any $i \in I_k \equiv \{i_1, \ldots, i_{k-1}\}$ we have $s_i + P_i \geq s_i + \frac{1}{2}P_i \geq s_b + \frac{1}{2}P_b$. Since $L'_{m'} = \min_{1 \leq j \leq M} L'_j$, we have

$$\begin{aligned} C^* &\geq L'_{m'} + \frac{s_b + P_b}{M} = \sum_{i \in I_k}(s_i + P_i) + \frac{s_b + P_b}{M} \\ &\geq (k-1)\left(s_b + \frac{1}{2}P_b\right) + \frac{1}{M}\left(s_b + \frac{1}{2}P_b\right). \end{aligned}$$

Hence

$$\frac{1}{M}\left(s_b + \frac{1}{2}P_b\right) \leq \frac{1}{M(k-1)+1}C^*.$$

By Corollary 3.2 we then have

$$C - C^* \leq \frac{M-1}{M}\left(s_b + \frac{1}{2}P_b\right) \leq \frac{M-1}{M(k-1)+1}C^*. \qquad \square$$

Corollary 3.5 follows directly from the proof of Theorem 3.4.

COROLLARY 3.5. *For any set* $\mathcal{L}$ *of jobs, if* $L'_{m'} \geq (k-1)(s_b + \frac{1}{2}P_b)$, *then* $\frac{C(\mathcal{L})}{C^*(\mathcal{L})} \leq \frac{Mk}{M(k-1)+1}$.

4. Worst-case performance: 2$\le M \le$ 4. After analyzing the asymptotic case, we turn to consider the worst case. Our consideration will be in two steps: $2 \le M \le 4$ and $M \ge 5$. The former step is completed in this section, and the latter step is completed in §5. First we have the following observation.

LEMMA 4.1. *If for some job set* $\mathcal{L}_2$

$$\frac{C_M(\mathcal{L}_2)}{C^*_M(\mathcal{L}_2)} > \frac{3M}{2M+1},$$

then $M - 3m + 1 \ge 0$.

Proof. Case 1. $m < m'$ and $\theta < P_b$. Then $C = (L_{m'} + L_m + s_b)/2 = L'_{m'} + s_b + \frac{1}{2}(P_b + \theta)$. On the other hand, $MC^* \ge \sum L_i \ge (m-1)L'_{m'} + (M-2m+1)L_m + mL_{m'}$. If $3m - M - 2 \ge 0$, then, noticing that $M - 2m + 1 \ge 0$ and $L_m + s_b < L_{m'}$, we get

$$\begin{aligned} MC^* &\ge (m-1)\left(s_b + \frac{P_b}{2}\right) + (M-2m+1)L_m + mL_{m'} \\ &\quad - \frac{3m-M-1}{2}(L_{m'} - L_m - s_b) - (M-2m+1)\frac{P_b}{2} \\ &= (M-m+1)C + (3m-M-2)\left(s_b + \frac{P_b}{2}\right). \end{aligned}$$

Since $C > \frac{3M}{2M+1}C^*$, Corollary 3.2 implies that $s_b + \frac{1}{2}P_b > \frac{M}{2M+1}C^*$. Hence

$$MC^* > (M-m+1)\frac{3M}{2M+1}C^* + (3m-M-2)\frac{M}{2M+1}C^* = MC^*,$$

which is a contradiction. Therefore, in this case $M - 3m + 1 \ge 0$.

Case 2. $m < m', \theta \ge P_b$, or $m = m'$. Then $C = L_{m'} = L'_{m'} + (s_b + P_b)$. Noticing that $L_m = (C - s_b) + (\theta - P_b)$ if $m < m'$ and $L_m = C$ if $m = m'$, we have $L_m \ge C - s_b$. Hence

$$\begin{aligned} MC^* &\ge (m-1)L'_{m'} + (M-2m+1)(C-s_b) + mC \\ &= ML'_{m'} + ms_b + (M-m+1)P_b. \end{aligned} \tag{4.1}$$

It is implied by Lemma 3.1 that

$$C^* < \frac{2M+1}{M-1}\left(\frac{M-m}{M}s_b + \frac{m-1}{M}P_b\right). \tag{4.2}$$

Combining (4.1) and (4.2), we then have

$$L'_{m'} < \left(\frac{3y}{M-1} - 1\right)s_b + \left(2 - \frac{3y}{M-1}\right)P_b, \text{ where } y = M - m \ge \frac{M-1}{2}.$$

Since $L'_{m'} \ge s_b + \frac{1}{2}P_b$,

$$\left(\frac{3y}{M-1} - 2\right)(s_b - P_b) > \frac{1}{2}P_b, \text{ implying } \frac{3y}{M-1} \ne 2.$$

Assume $\frac{3y}{M-1} - 2 < 0$; then $s_b < P_b$. Let $s_b = (1-\alpha)P_b$ for some $0 < \alpha \le 1$. Then

$$(1-\alpha)P_b + \frac{1}{2}P_b \le L'_{m'} < P_b - \left(\frac{3y}{M-1} - 1\right)(P_b - s_b) = P_b - \alpha\left(\frac{3y}{M-1} - 1\right)P_b,$$

or

$$\alpha > \frac{1}{2(2-\frac{3y}{M-1})} \geq \frac{1}{2(2-\frac{3}{2})} = 1,$$

a contradiction. Hence

$$\frac{3y}{M-1} - 2 > 0 \quad \text{or} \quad M - 3m + 1 \geq 0,$$

completing our proof of Lemma 4.1. □

Now we are ready to prove a main result.

THEOREM 4.2. *For* $2 \leq M \leq 4$ *we have*

$$\frac{C_M(\mathcal{L}_2)}{C_M^*(\mathcal{L}_2)} \leq \frac{3M}{2M+1}.$$

Proof. Given $2 \leq M \leq 4$, suppose $\mathcal{L}_2$ is such that it violates the preceding inequality. Then we must have $m = 1$ and $m' = M$ according to Lemma 4.1. Recall that the schedule, which algorithm LBT had produced at the moment batch b was to be assigned, is $(L'_1, \dots, L'_M)$. Let

$$\mathcal{B}_1 = \{i \mid s_i + P_i = L'_j \text{ for some } 1 \leq j \leq M\}$$

and

$$\mathcal{B} = \{1, 2, \dots, b\}, \quad \mathcal{B}_2 = \mathcal{B} \backslash \mathcal{B}_1.$$

CLAIM 1. *In any optimal schedule, any batch of* $\mathcal{B}_1$ *cannot be wholly assigned on the same machine with another whole batch of* $\mathcal{B}$.

Proof. Suppose batch $a' \in \mathcal{B}_1$ is on the same machine as $b' \in \mathcal{B}$ in an optimal schedule. Then we get

$$C^* \geq (s_{a'} + P_{a'}) + (s_{b'} + P_{b'}) \geq L'_M + (s_{b'} + P_{b'}). \tag{4.3}$$

Case 1. $L'_1 + s_b \geq L_M$. Then $C = L_M = L'_M + (s_b + P_b)$. According to (4.3) and the batch sequencing by LBT, we have $C - C^* \leq (s_b + P_b) - (s_{b'} + P_{b'}) \leq s_{b'} - s_b$. Hence

$$s_{b'} \geq s_b + (C - C^*) > s_b + \frac{M-1}{2M+1}C^*,$$

which implies that

$$C^* \geq L'_M + (s_{b'} + P_{b'}) > 2s_b + \frac{M-1}{2M+1}C^*,$$

or $s_b < \frac{M+2}{2(2M+1)}C^*$, which contradicts Corollary 3.3:

$$\frac{M-1}{2M+1}C^* < C - C^* \leq \frac{M-1}{M}s_b.$$

Case 2. $L'_1 + s_b < L_M$. Then $C = L'_M + s_b + \frac{1}{2}(\theta + P_b)$ for some $0 \leq \theta < P_b$. By (4.3) we have

$$C - C^* \leq (s_b + P_b) - (s_{b'} + P_{b'}) - \frac{P_b - \theta}{2} \leq s_{b'} - \left(s_b + \frac{P_b}{2}\right) + \frac{\theta}{2},$$

or

$$s_{b'} > \left(s_b + \frac{P_b}{2}\right) - \frac{\theta}{2} + \frac{M-1}{2M+1}C^*,$$

which, together with (4.3) (if we notice that $L'_M \geq s_b + \frac{1}{2}P_b$), implies that

$$C^* \geq \left(s_b + \frac{P_b}{2}\right) + s_{b'} \geq 2s_b + P_b - \frac{\theta}{2} + \frac{M-1}{2M+1}C^* \geq 2\left(s_b + \frac{P_b - \theta}{2}\right) + \frac{M-1}{2M+1}C^*.$$

Again, this contradicts Corollary 3.3:

$$\frac{M-1}{2M+1}C^* < C - C^* \leq \frac{M-1}{M}\left(s_b + \frac{P_b - \theta}{2}\right).$$

Therefore, Claim 1 is true. □

CLAIM 2. *In any optimal schedule of $\mathcal{L}_2$, no machine can receive three complete batches of $\mathcal{B}$.*

Proof. Suppose in an optimal schedule some machine is assigned with, among others, three complete batches, say, i_1, i_2, and i_3. Then

$$C^* \geq \sum_{k=1}^{3}(s_{i_k} + P_{i_k}) \geq 3\left(s_b + \frac{1}{2}P_b\right).$$

But by Corollary 3.2 we also have

$$\frac{M-1}{2M+1}C^* < C - C^* \leq \frac{M-1}{M}\left(s_b + \frac{1}{2}P_b\right),$$

which then implies a contradiction $3M < 2M + 1$. Therefore, Claim 2 is also true. □

Suppose first that at least one preemption is imposed on batch $b' \in \mathcal{B}\backslash\{b\}$, in some optimal schedule π^*, and batch b' was assigned by LBT on machine i. Let

$$(4.4) \qquad (2s_b + P_b) - (s_a + P_a) = \sigma,$$

where batch a is the first batch on machine $m' = M$, assigned by LBT.

It follows that $L_i + s_{b'} \geq 2s_{b'} + P_{b'} \geq 2s_b + P_b = (s_a + P_a) + \sigma$. But we also have $L_i \geq s_a + P_a + \theta$. Thus

$$(4.5) \qquad L_i + s_{b'} \geq \max\{s_a + P_a + \sigma,\ s_a + P_a + \theta\} = (s_a + P_a) + \max\{\theta, \sigma\}.$$

Since C^* is not less than the mean value of total load, we get

$$(4.6) \quad C^* \geq (s_a + P_a) + \frac{1}{M}(s_b + P_b) + \frac{M-2}{M}\theta + \frac{1}{M}s_{b'} + \begin{cases} \frac{L_i - (s_a + P_a)}{M} & \text{if } i \neq M, \\ \frac{1}{M}\theta & \text{if } i = M. \end{cases}$$

From (4.5) and the fact that $s_a \geq \sigma$, (4.6) implies

$$(4.7) \qquad (s_a + P_a) + \frac{1}{M}(s_b + P_b) + \frac{M-2}{M}\theta + \frac{1}{M}\max\{\theta, \sigma\} \leq C^*.$$

Summing up $(M-1)$ times of (4.4) and $3M$ times of (4.7), we obtain

$$\begin{aligned} &(2M+1)((s_a + P_a) + (s_b + P_b)) - (M-1)(P_b - \theta) \\ &\leq 3MC^* - 3\max\{\theta, \sigma\} + (M-1)\sigma - (2M-5)\theta \\ &\leq 3MC^* - \tfrac{4-M}{2}\max\{\theta, \sigma\} \leq 3MC^*, \end{aligned}$$

or

$$(s_a + P_a + s_b + P_b) - \frac{M-1}{2M+1}(P_b - \theta) \le \frac{3M}{2M+1}C^*.$$

Considering that $C = s_a + P_a + s_b + P_b - \max\{0, \frac{1}{2}(P_b - \theta)\}$, we then have $C \le \frac{3M}{2M+1}C^*$, a contradiction.

Secondly, suppose all the batches of $\mathcal{B}\backslash\{b\}$ are not preempted in optimal schedule π^*. Consider the assignment of $\mathcal{B}\backslash\{b\}$ in π^*. According to Claim 1, all the batches of $\mathcal{B}_1$ keep separate from batches of $\mathcal{B}\backslash\{b\}$ in this assignment. Also, no three whole batches of $\mathcal{B}$ can be on the same machine according to Claim 2. Therefore, in π^* any machine receives at least one whole batch of $\mathcal{B}_1$ or two whole batches of $\mathcal{B}_2\backslash\{b\}$. On the other hand, it is apparent that at least one machine receives $1/M$ of load of batch b. Noticing that, for $i, j \in \mathcal{B}$, $(s_i + P_i) + (s_j + P_j) > L'_M$, according to Corollary 3.5 for $k = 3$, and that $s_i + P_i \ge L'_M$ for all $i \in \mathcal{B}_1$, we then obtain

$$C^* \ge L'_M + s_b + \frac{1}{M}P_b \ge 2s_b + \frac{M+2}{2M}P_b. \tag{4.8}$$

Case 1. $\theta \ge P_b$. Then $C = L'_M + (s_b + P_b)$, which, together with (4.8), implies that $\frac{M-1}{2M+1}C^* < C - C^* \le \frac{M-1}{M}P_b$, or

$$P_b > \frac{M}{2M+1}C^*. \tag{4.9}$$

But at the same time Corollary 3.3 gives that

$$s_b > \frac{M}{2M+1}C^*. \tag{4.10}$$

Combining (4.8), (4.9), and (4.10), we have

$$C^* > 2s_b + \frac{1}{M}P_b > \frac{2M}{2M+1}C^* + \frac{1}{2M+1}C^* = C^*,$$

a contradiction.

Case 2. $\theta < P_b$. Then $C = L'_M + s_b + P_b - \frac{1}{2}(P_b - \theta)$, which, together with (4.8), implies that

$$\frac{M-1}{2M+1}C^* < C - C^* \le \frac{M-1}{M}P_b - \frac{1}{2}(P_b - \theta),$$

or

$$P_b > \frac{M}{2M+1}C^* + \frac{M}{2(M-1)}(P_b - \theta). \tag{4.11}$$

But Corollary 3.3 implies that

$$s_b > \frac{M}{2M+1}C^* - \frac{M-2}{2(M-1)}(P_b - \theta). \tag{4.12}$$

Substituting (4.11) and (4.12) into (4.8) yields

$$C^* \ge \frac{3}{2}s_b + \frac{M+2}{2M}P_b > C^* + \left(\frac{M+2}{4(M-1)} - \frac{3(M-2)}{4(M-1)}\right)(P_b - \theta) \ge C^*,$$

another contradiction, which completes the proof of Theorem 4.2. $\square$

5. Worst-case performance: $M \geq 5$. In this section our interest is on problems in which no batch contains an amount of processing sufficiently large that we can deduce immediately that it must be processed on at least two machines. More formally, if for each batch i, $C^*_{\max} \geq s_i + P_i$, then such problems will be called *small-batch problems*. For problems other than small-batch problems, there is a linear heuristic that has a worst-case performance ratio of $2 - 1/(\lfloor M/2 \rfloor + 1)$ [6]. The main result is the following.

THEOREM 5.1. *For $M \geq 5$ and for any small-batch problem $\mathcal{L}_2$, we have*

$$\frac{C_M(\mathcal{L}_2)}{C^*_M(\mathcal{L}_2)} \leq \frac{3M-4}{2M-2}.$$

We are going to prove the theorem by contradiction, by assuming the existence of a counterexample. Let $\mathcal{L}_2$ be a counterexample to the theorem, and let π^* be an optimal schedule of $\mathcal{L}_2$. Denote the makespan of any schedule π by $C_{\max}(\pi)$. Our proof is in two parts. In the first part we shall derive a lower bound on $C^*_M(\mathcal{L}_2)$ by relaxation of $\mathcal{L}_2$ in π^* and by rearrangements of π^*. In the second part we will deal with some mathematical relations.

5.1. Lower bound on C^*_M ($\mathcal{L}_2$). Let $\mathcal{B}$, $\mathcal{B}_1$, and $\mathcal{B}_2$ be defined as in the proof of Theorem 4.2, and let $\mathcal{B}' = \{b+1, \ldots, N\}$. Let the first batch on machine m' be a. Denote $\alpha = s_a + P_a$. *All the following relaxations (replacements, transformations) will reserve the load—the sum of setup and/or processing times concerned.*

Step 1. *First relaxation of $\mathcal{L}_2$ into $\mathcal{L}'$.* Consider the configuration that LBT had created in its first phase.

(a) For every $i \in \mathcal{B}'$ let an additional job with processing time s_i be added to batch i and then let all jobs in batch i have zero setup time. In other words, batches of $\mathcal{B}'$ are replaced by a newly created batch with zero setup time but the same load. This batch will be called batch i_0.

Observe that if $\mathcal{B}_2 \backslash \{b\} \neq \phi$, then $\mathcal{B}_2 \backslash \{b\}$ consists of an even number of batches. In fact, for any j, $1 \leq j \leq M$, L'_j is the total load of at most two batches. Otherwise, $L'_{m'} = \alpha$ would exceed the total load of at least two complete batches in $\mathcal{B}$, which implies that $\alpha \geq 2(s_b + \frac{1}{2}P_b)$, contradicting Corollary 3.5. Therefore, batches in $\mathcal{B}_2 \backslash \{b\}$ are in pairs.

(b) If in some pair of batches in $\mathcal{B}_2 \backslash \{b\}$, say, $\{k_1, k_2\}$, there is a batch, say, k_2, satisfying $s_{k_2} + P_{k_2} \geq \alpha$, then jobs of batch k_1 are replaced by some corresponding new jobs of batch i_0 with total processing time $s_{k_1} + P_{k_1}$.[1]

Let the number of pairs that have this property be $\frac{1}{2}|\mathcal{B}_2 \backslash \{b\}| - q \geq 0$.

(c) If $q \geq 1$ and $2s_b + P_b \leq C^*_M(\mathcal{L}_2)$, then each of the q pairs of batches in $\mathcal{B}_2 \backslash \{b\}$, say, $\{k_1, k_2\}$, is jointly replaced by a new batch $i_{\{k_1,k_2\}}$ that has zero setup time and total processing time $2s_b + P_b$ ($> \alpha$ by Corollary 3.5) and by a new job of batch i_0 with processing time $(s_{k_1} + P_{k_1}) + (s_{k_2} + P_{k_2}) - (2s_b + P_b)$.[1]

The case

$$q \geq 1, \quad 2s_b + P_b > C^*_M(\mathcal{L}_2) \tag{5.1}$$

is considered in a parallel manner.

After the preceding relaxation, the resulting new list $\mathcal{L}'$ consists of $M+2$ ($M+q+2$ if (5.1) holds) batches including batch b, reindexed as $i_0, \ldots, i_{M+1}$ ($i_0, \ldots, i_{M+q+1}$, respectively, with the q pairs of batches being reindexed last), where batch i_0 has zero setup time. Furthermore, we have the following guarantees:

(1) $C^*_M(\mathcal{L}') \leq C^*_M(\mathcal{L}_2)$.

[1]The relaxations could be made meticulously. See the footnote for relation (5.2).

The inequality holds because all the changes made are nothing other than conversions of machine setups into job processings.

(2) For each batch i_v, $v = 1, \dots, M+1$ ($v = 1, \dots, M+q+1$, respectively), $2s_{i_v} + P_{i_v} \geq 2s_b + P_b$ and $s_{i_v} + P_{i_v} \leq C_M^*(\mathcal{L}_2)$. Furthermore, for $v = 1, \dots, M+1$ ($v = 1, \dots, M-q+1$, respectively), $s_{i_v} + P_{i_v} \geq \alpha$, except possibly the former batch b.

(3) If (5.1) holds, then for $v = 1, \dots, 2q$, $s_{i_{M-q+1+v}} + P_{i_{M-q+1+v}} < \alpha$.

Let π' be an optimal schedule of $\mathcal{L}'$.

Step 2. *Rearrangements of* π' *to* π''.

Convention. First of all, we can always regard the jobs as different from machine to machine in π'. In other words, no job preemption is introduced in π' since otherwise we can always make it so by modifying the list $\mathcal{L}'$. For example, if job J is split in π' into several parts, $J_1, J_2, \dots$, then we just remove J from and add $J_1, J_2, \dots$ to $\mathcal{L}'$.

LEMMA 5.2. *Given machine i with loaded jobs, suppose these jobs belong to at least two different batches $j_1, \dots, j_k$ with $k \geq 2$ and $s_{j_1} \geq \cdots \geq s_{j_k}$. Then for any $1 \leq v \leq k-1$, through interchanges with jobs on machine i belonging to batches $j_{v+1}, \dots, j_k$, jobs on other machines belonging to batch j_v can be so reassigned onto machine i without increasing the makespan that either jobs belonging to batches $j_{v+1}, \dots, j_k$ are all moved out of machine i or all jobs of batch j_v are moved onto machine i.*

Proof. We prove the lemma for $v = 1$. Fix any t: $2 \leq t \leq k$. Suppose on machine i' there are jobs of batch j_1.

Let P_{ij} be the total processing time of jobs on machine i belonging to batch $j, i = 1, \dots, M, j = i_1, \dots, i_{M+1}$ ($i_1, \dots, i_{M+q+1}$, respectively).

Case 1. $s_t + P_{it} < P_{i'j_1}$. Move all jobs of batch t on machine i to machine i' and part of the jobs, with a total processing time $(s_t + P_{it})$, of batch j_1 on machine i' to machine i.

This makes $P_{it} := 0$, $P_{ij_1} := P_{ij_1} + (s_t + P_{it})$, and $P_{i't} := P_{i't} + P_{it}$, $P_{i'j_1} := P_{i'j_1} - (s_t + P_{it})$.

Case 2. $s_t + P_{it} \geq P_{i'j_1}$. Move all jobs on machine i' of batch j_1 to machine i, and move either all jobs (if $P_{i'j_1} \geq P_{it}$) or part of the jobs with a total processing time $P_{i'j_1}$ (if $P_{i'j_1} < P_{it}$) of batch t on machine i to machine i'.

This makes $P_{ij_1} := P_{ij_1} + P_{i'j_1}$, $P_{i'j_1} := 0$, and either $P_{it} := 0$, $P_{i't} := P_{i't} + P_{it}$ (if $P_{i'j_1} \geq P_{it}$) or $P_{it} := P_{it} - P_{i'j_1}$, $P_{i't} := P_{i't} + P_{i'j_1}$ (if $P_{i'j_1} < P_{it}$).

During these movements, if preemptions have to be imposed on some jobs, we just regard the parts of jobs as different jobs, because we can always make it so by modifying the list $\mathcal{L}'$. See the Convention in Step 2.

After these movements, we reschedule the jobs on machines i and i' so that jobs of the same batch are scheduled consecutively. We see that the schedules of all machines except machines i and i' remain the same; also, the completion times of machines i and i' cannot be increased.

Continue the preceding interchanges until reaching one of the two cases mentioned in the lemma. □

With Lemma 5.2 we can rearrange π' without increasing its makespan beyond $C_M^*(\mathcal{L}_2)$, so that each machine contains a complete batch of $I \equiv \{i_1, \dots, i_{M+1}\}$ (of $I' \equiv \{i_1, \dots, i_{M+q+1}\}$, respectively). The procedure is as follows.

PROCEDURE

(i) $i := 1$.

With interchange argument, we can always assume that on machine i there is at least one job of some batch from I (I', respectively).

(ii) Let the jobs on machine i be from batches $\{j_1, \dots, j_k\} \subseteq I$ (I', respectively), with $s_{j_1} \geq \cdots \geq s_{j_k}$, and possibly also from batch i_0.

If $k = 1$, then move all the jobs of batch j_1 on other machines to machine i (possibly some interchanges with jobs on machine i belonging to batch i_0 are needed) and goto (iv). This is justified by the fact that we are considering small-batch problems and have the guarantee (2) in Step 1: $s_{i_v} + P_{i_v} \leq C^*_M(\mathcal{L}_2)$.

For $k \geq 2$, if all jobs of batch j_1 are already on machine i, i.e., $P_{ij_1} = P_{j_1}$, goto (iv); otherwise, make rearrangements according to Lemma 5.2 for $v = 1$ to reach one of the two cases.

(iii) Return to (ii).

(iv) If $i = M$, stop. Otherwise, let $i := i + 1$ and goto (ii).

Let the resulting schedule of $\mathcal{L}'$ after Step 2 be π''. Now in π'' each machine contains a complete batch other than i_0.

If (5.1) holds, then we have a nicer structure.

LEMMA 5.3. *Suppose* (5.1) *is true. Then, after some relaxations, the* $2q$ *batches* $I_2 \equiv \{i_{M-q+2}, \dots, i_{M+q+1}\}$ *can be assigned on exactly q machines, which are loaded full to* $C^*_M(\mathcal{L}_2)$, *and the other* $M - q$ *machines each contain a complete batch of* $I_1 \equiv \{i_1, \dots, i_{M-q+1}\}$.

Proof. First we see that in π'', if a machine contains a complete batch $i \in I_1$ other than batch b, then for any $i' \in I_2$ no job of batch i' can be on the machine since otherwise the total load of the machine would exceed $(s_i + P_i) + s_{i'} \geq \alpha + s_{i'} > 2s_{i'} + P_{i'} \geq 2s_b + P_b > C^*_M(\mathcal{L}_2)$, a contradiction (see guarantees (2) and (3) in Step 1). Consider the machines that already contain a complete batch of I_2. Let them be, for instance, machines $1, \dots, M'$. The other $M - M'$ machines each contain a complete batch of I_1, and at most one of those machines contains jobs of batches from I_2. This machine, if any, contains the complete batch b.

We claim that $M' \geq 2$. In fact, (i) no two complete batches of I_2 can be on the same machine since for $i \in I_2$, $s_i + P_i \geq \frac{1}{2}(2s_b + P_b) > \frac{1}{2}C^*_M(\mathcal{L}_2)$, and (ii) no complete batch of I_2 can be on the same machine with the complete batch b since for $i \in I_2$, $s_i + P_i + s_b + P_b \geq 2s_b + P_b > C^*_M(\mathcal{L}_2)$. Therefore, if $M' \leq 1$, then either all the jobs of $2q \geq 2$ batches of I_2 have to be assigned on $M' \leq 1$ machine, which is impossible according to (i), or these $2q$ batches, together with batch b, have to be assigned on $M' + 1 \leq 2$ machines. In the latter case, some batch splitting must be introduced owing to (i) and (ii), but this implies that, for some $i', i'' \in I_2$, $2C^*_M(\mathcal{L}_2) \geq (s_{i'} + P_{i'}) + (s_{i''} + P_{i''}) + (s_b + P_b) + s_{i'} \geq 2(2s_b + P_b)$, which is again a contradiction.

Now we relax the complete batches on machines $1, \dots, 2\lfloor \frac{M'}{2} \rfloor$ in the following way. Let i be such a batch. Then transform $s_i + P_i - \frac{1}{2}C^*_M(\mathcal{L}_2)$ of the load of batch i to batch i_0. By doing this, we are able to move every two of these $2\lfloor \frac{M'}{2} \rfloor$ batches together on the same machine. These $\lfloor \frac{M'}{2} \rfloor$ machines are loaded full to $C^*_M(\mathcal{L}_2)$. Let the index set of the other $2q'$ batches be I'_2, where $q' = q - \lfloor \frac{M'}{2} \rfloor$. Execute the Procedure for the remaining $M - \lfloor \frac{M'}{2} \rfloor$ machines, so that each contains a complete batch of $I_1 \cup I'_2$.

If $q' = 0$, we are done. Otherwise, replace M by $M - \lfloor \frac{M'}{2} \rfloor$ and q by q' and repeat the operations just described until the desired result is obtained. □

Step 3. *Further relaxation of* $\mathcal{L}'$ *to* $\mathcal{L}''$. Without loss of generality, we now assume the following: If (5.1) does not hold, then machine k contains a complete batch i_k, $k = 1, 2, \dots, M$. If (5.1) does hold, then (i) machine i is loaded full to $C^*_M(\mathcal{L}_2)$ with batches of I_2, $i = M - q + 1, \dots, M$, and (ii) machine k contains a complete batch i_k, $k = 1, \dots, M - q$. Denote $i'_1 = i_{M+1}$ or $i'_1 = i_{M-q+1}$, and let $b \in \{i_1, i'_1\}$. We see that the load of each of machines $2, \dots, M$ (if $i_1 \neq b$, then also of machine 1), excluding the contribution from batch i_0 and from batch i'_1, exceeds α. We further relax our counterexample by transforming all these exceeding parts of the loads to batch i_0. This completes our final relaxation.

Since in all of our relaxations the total load is conserved, we have

$$P_{i_0} \geq \begin{cases} (m-1)(s_b + P_b) + (M-2m)\theta & \text{if } b \neq i_1',^2 \\ (m-1)(s_b + P_b) + (M-2m+1)\theta & \text{if } b = i_1'. \end{cases} \tag{5.2}$$

Now consider the following auxiliary problem. Given a number of jobs that are divided into three batches i_0, i_1, and i_1', with

$$\begin{aligned} &s_j + P_j \leq C_M^*(\mathcal{L}_2), \quad 2s_j + P_j \geq 2s_b + P_b, \quad j \in \{i_1, i_1'\}, \\ &s_j + P_j \geq \alpha \quad \text{if } j \in \{i_1, i_1'\}\backslash\{b\}, \end{aligned} \tag{5.3}$$

and $b \in \{i_1, i_1'\}$, $s_{i_0} = 0$, P_{i_0} satisfying (5.2). Let π_0'' be an optimal schedule of these jobs with the restrictions that (i) machines $2, \dots, M$ are available at time α, and (ii) if $b = i_1'$, then machine 1 is also available at time α. If $b \neq i_1'$, then machine 1 is assigned first the complete batch b. It is then apparent that

$$C_{\max}(\pi_0'') \leq C_M^*(\mathcal{L}_2). \tag{5.4}$$

5.2. Some mathematical relations. Throughout this subsection and the appendix we denote $(3M-4)/(2M-2)$ by τ_M. First consider the following problem $Q(l_1, l_2, s)$, where $s, l_1 \geq 0, l_2 > 0$.

Let J_1 and J_2 be two special jobs that can be preempted anywhere and be simultaneously processed by several machines but need setup times in shifting processing from one job to another. They are to be scheduled on M identical machines with the objective of minimizing the makespan. The setup times of J_1 and J_2 are 0 and s, respectively, and their processing times are l_1 and l_2, respectively. For $k = 1, 2, \dots, M-1$ let π_k be a schedule such that each of machines $\{1, \dots, k\}$ is assigned l_2/k processing of job J_2 and each of machines $\{k+1, \dots, M\}$ is assigned $l_1/(M-k)$ processing of job J_1 if $l_1/(M-k) \leq s + l_2/k$; otherwise, distribute the remaining processing $l_1 - (M-k)(s + l_2/k)$ of job J_1 evenly among M machines. Let π_M be a schedule such that each of machines $\{1, \dots, M\}$ is assigned l_1/M processing of job J_1 and l_2/M processing of job J_2. Then it is easy to see that an optimal solution to $Q(l_1, l_2, s)$ must be among $\{\pi_1, \dots, \pi_M\}$.

Let $A_k = (M-k)(s + l_2/k)$, $k = 1, 2, \dots, M$. Then we have

$$C_{\max}(\pi_k) = s + \frac{l_2}{k} + \max\left\{0, \frac{1}{M}(l_1 - A_k)\right\}.$$

The following lemma establishes the optimality of π_k.

LEMMA 5.4. *If* $s = 0$, *then* π_M *is optimal. If* $s > 0$, *then schedule* π_k *is optimal if and only if*

$$A_k - s \leq l_1 \leq A_{k-1} - s,$$

where $1 \leq k \leq M$ *and* $A_0 = +\infty$.

Proof. The case $s = 0$ is obvious. Suppose $s > 0$. First we observe that

$$A_k - A_{k+1} = s + \frac{M}{(k+1)k} l_2, \quad k = 1, 2, \dots, M-1.$$

[2]Here we assume batch i_1' comes from machine j, $1 \leq j \leq m'-1$, or $s_{i_1'} + P_{i_1'} \leq L_m$ since otherwise it is either some batch k_2 in (b) of Step 1 or some newly created batch $i_{\{k_1,k_2\}}$ in (c) of Step 1. In each case, our assumption can be made true by using meticulous relaxations on the pair of batches there.

(The *if* part.) Suppose $A_k \le l_1 + s \le A_{k-1}$ for some $1 \le k \le M$. Then $\pi_{k'}$ cannot be better than π_k for any $k' \ne k$. In fact, if $1 \le k' \le k-1$, then $l_1 \le A_{k-1} - s \le A_{k'} - s$, implying that $C_{\max}(\pi_{k'}) = s + l_2/k' \ge s + l_2/(k-1) \ge C_{\max}(\pi_k)$ since $\frac{1}{M}(l_1 - A_k) \le \frac{1}{M}(A_{k-1} - s - A_k) = \frac{1}{(k-1)k} l_2$. Let $k+1 \le k' \le M$. Since

$$A_k - A_{k'} = \sum_{i=k}^{k'-1}(A_i - A_{i+1}) = \sum_{i=k}^{k'-1}\left(s + \frac{M}{i(i+1)}l_2\right) = (k'-k)s + \left(\frac{1}{k} - \frac{1}{k'}\right)Ml_2,$$

we have

$$A_k \ge s + A_{k'} + \left(\frac{1}{k} - \frac{1}{k'}\right)Ml_2,$$

and hence, if we consider that $l_1 \ge A_k - s$,

$$C_{\max}(\pi_{k'}) = s + \frac{l_2}{k'} + \frac{1}{M}(l_1 - A_{k'}) \ge s + \frac{l_2}{k} + \max\left\{0,\ \frac{1}{M}(l_1 - A_k)\right\} = C_{\max}(\pi_k).$$

(The *only if* part.) Now suppose $l_1 < A_k - s$ for some $1 \le k \le M$ (notice that $k < M$ since $A_M = 0 \le l_1$). Then

$$C_{\max}(\pi_k) = s + \frac{l_2}{k} > s + \frac{l_2}{k+1} + \max\left\{0,\ \frac{1}{M}(l_1 - A_{k+1})\right\} = C_{\max}(\pi_{k+1}),$$

implying that π_{k+1} is better than π_k. Suppose, on the other hand, that $l_1 > A_{k-1} - s = A_k + \frac{M}{(k-1)k} l_2$ for some $1 \le k \le M$ (notice that $k > 1$ since $l_1 < +\infty$). Then

$$C_{\max}(\pi_k) = s + \frac{l_2}{k} + \frac{1}{M}(l_1 - A_k) > s + \frac{l_2}{k-1} + \max\left\{0,\ \frac{1}{M}(l_1 - A_{k-1})\right\} = C_{\max}(\pi_{k-1}),$$

implying again that π_k is not optimal. □

Define piecewise-linear function

$$H(l_1, l_2, s) = s + \frac{1}{k}l_2 + \frac{1}{M}\Big(\min(l_1,\ A_{k-1} - s) - A_k\Big),\quad A_k \le l_1 < A_{k-1},\ k = 1, \dots, M.$$

Then, by Lemma 5.4, $H(l_1, l_2, s)$ is the optimal makespan for problem $Q(l_1,\ l_2,\ s)$.

Now we are ready to continue our proof of Theorem 5.1. There are two possibilities we have to take into consideration: $b \ne i_1$ and $b = i_1$. Suppose first that $b \ne i_1$.

We can easily derive from (5.2), (5.3), and (5.4) that

$$C^*_M(\mathcal{L}_2) \ge C_{\max}(\pi''_0) \ge \alpha + H(l_1, P_b, s_b), \tag{5.5}$$

where $l_1 = (m-1)(s_b + P_b) + (M - 2m + 1)\theta$, and

$$\begin{aligned} &C_M(\mathcal{L}_2) = \alpha + s_b + \tfrac{1}{2}P_b + \tfrac{1}{2}\min(\theta,\ P_b), \\ &C^*_M(\mathcal{L}_2) \ge s_b + P_b, \quad \alpha = s_a + P_a \ge s_b + \tfrac{1}{2}P_b. \end{aligned} \tag{5.6}$$

Let

$$\lambda_1 = M - 2m + 1, \quad r_1 = \tfrac{5}{4} - \tfrac{M-1}{4\lambda_1}.$$

Since $M - 3m + 1 \geq 0$ by Lemma 4.1, we have

$$\tfrac{1}{3}(M+1) \leq \lambda_1 \leq M-1 \quad \text{and} \quad \tfrac{1}{2} < \tfrac{M+4}{2(M+1)} \leq r_1 \leq 1. \tag{5.7}$$

Set

$$\alpha := \alpha/P_b, \quad s := s_b/P_b, \quad l_1 := l_1/P_b.$$

Then from (5.5) and (5.6) it follows that

$$\frac{C_M(\mathcal{L}_2)}{C_M^*(\mathcal{L}_2)} \leq \frac{\alpha - \frac{1}{2} + r_1(s+1) + \frac{1}{2\lambda_1}\min\{l_0, l_1\}}{\max\{s+1, \alpha + H(l_1, 1, s)\}}, \tag{5.8}$$

where

$$\begin{aligned} &l_0 = \tfrac{M-1-\lambda_1}{2}(s+1) + \lambda_1, \\ &\alpha \geq s + \tfrac{1}{2}, \quad s \geq 0, \quad l_1 \geq 0. \end{aligned} \tag{5.9}$$

Let

$$R(l_1, s, \alpha) = \frac{\alpha - \frac{1}{2} + r_1(s+1) + \frac{1}{2\lambda_1}l_1}{\max\{s+1, \alpha + H(l_1, 1, s)\}}, \tag{5.10}$$

where (l_1, s, α) satisfy (5.9) and, in addition, $l_1 \leq l_0$.

It is apparent from (5.8) that any upper bound on $R(l_1, s, \alpha)$ is also an upper bound on $C_M(\mathcal{L}_2)/C_M^*(\mathcal{L}_2)$. We are to show that $R(l_1, s, \alpha)$, and hence $C_M(\mathcal{L}_2)/C_M^*(\mathcal{L}_2)$, is upper-bounded by τ_M, which, as we have desired, contradicts our assumption that $\mathcal{L}_2$ is a counterexample to Theorem 5.1.

Because of the special structure of the function $H(\cdot, 1, \cdot)$ and the fact that $l_0 \leq A_1 - s = (M-1)(s+1) - s$, we need only to consider the case $A_k \leq l_1 \leq A_{k-1} - s$ for $k : 2 \leq k \leq M$, where $A_k = (M-k)(s + \frac{1}{k})$. Therefore, Lemma A.3 in the appendix is applicable, and accomplishes our consideration for the case $b \neq i_1$.

Now, let us suppose that $b = i_1$. We divide our discussion into two cases, namely, $\alpha \leq \beta$ and $\alpha > \beta$, where $\beta = s_b + P_b$.

Case A. $\alpha \leq \beta$. We may assume $s_{i_1'} + P_{i_1'} \geq \beta$. Otherwise, we can exchange the roles of batches i_1' and b in π_0'' to get a no worse schedule, which reduces to the case $b \neq i_1$, which we have already considered. More specifically, suppose in π_0'' batch i_1' is split among ν machines with loads $s_{i_1'} + P_{i_1'}^{(j)}$ $(j = 1, \ldots, \nu)$. It is apparent that $\nu \geq 2$. Suppose $s_{i_1'} + P_{i_1'} < \beta$; then for some $\varepsilon \geq 0$ we have $\nu(s_{i_1'} - s_b - \varepsilon) = P_b - P_{i_1'}$ since $2s_{i_1'} + P_{i_1'} \geq 2s_b + P_b$ by (5.3). Now assign the complete batch i_1' in the place of batch b, split batch b into loads $s_b + (s_{i_1'} + P_{i_1'}^{(j)} - s_b - \varepsilon)$ $(j = 1, \ldots, \nu)$, and put them in the place of the corresponding parts of batch i_1'. Then the resulting schedule is no worse than π_0''.

Relax into batch i_0 the partial processing of batch b that is assigned on machine 1 after time α in π_0''. Then, in a manner similar to that just described, we can derive (see [3]) that $C_M(\mathcal{L}_2)/C_M^*(\mathcal{L}_2)$ is upper-bounded by

$$R(l_1', s', \alpha, s, \beta) = \frac{\frac{1}{2\lambda_2}l_1' + (1 + \frac{1}{2\lambda_2})\alpha + \frac{1}{2}s + (r_2 - \frac{1}{2\lambda_2})\beta}{\max\{s'+1,\ \alpha + H(l_1', 1, s')\}},$$

where $(l_1', s', \alpha, s, \beta)$ satisfy

$$\begin{aligned} &l_1' \leq \tfrac{1}{2}(M+\lambda_2)\beta - \alpha - \lambda_2 s, \quad \beta \geq \alpha \geq \tfrac{1}{2}\beta + \tfrac{1}{2}s, \quad \alpha, s, \beta \geq 0, \\ &s' \geq \max\{0, \beta - 1, \tfrac{1}{2}(\beta + s - 1)\}, \quad l_1' \geq \beta - \alpha, \end{aligned} \tag{5.9$'$}$$

and

$$\lambda_2 = M - 2m, \qquad r_2 = \tfrac{3}{4} - \tfrac{M-2}{4\lambda_2}.$$

Thus

$$\tfrac{1}{3}(M-2) \leq \lambda_2 \leq (M-2), \qquad 0 \leq r_2 \leq \tfrac{1}{2}.$$

We are to show that $R(l_1', s', \alpha, s, \beta)$ is upper-bounded by τ_M.

If $l_1' > A_1 - s' = (M-1)(s'+1) - s'$, then $H(l_1', 1, s') = s'+1$. Since $s'+1 \geq \beta$ by (5.9′), we have

$$R(l_1', s', \alpha, s, \beta) \leq \frac{\frac{1}{2\lambda_2}l_0' + (1+\frac{1}{2\lambda_2})\alpha + \frac{1}{2}s + (r_2 - \frac{1}{2\lambda_2})\beta}{\alpha+\beta} = \frac{\alpha+\beta}{\alpha+\beta} = 1.$$

We are done. Therefore, we can suppose that $l_1' \leq A_1 - s'$. Because of the special structure of the function $H(\cdot, 1, \cdot)$, we need only to consider the case $A_k \leq l_1' \leq A_{k-1} - s'$ for $2 \leq k \leq M$. Lemma A.4 in the appendix then serves our purpose.

Case B. $\alpha > \beta$. In this case we need the following two lemmas.

LEMMA 5.5. $\alpha < \beta + P_{i_1'}$.

Proof. Suppose, to the contrary, that $\alpha \geq \beta + P_{i_1'}$. Since $s_{i_1'} + P_{i_1'} \geq \alpha$, then $s_{i_1'} \geq \beta$. If some job of batch i_1' is assigned to some machine k in π_0'', $2 \leq k \leq M$, then $C_{\max}(\pi_0'') \geq \alpha + s_{i_1'} \geq \alpha + \beta$. If all jobs of batch i_1' are assigned to machine 1, then $C_{\max}(\pi_0'') \geq \beta + s_{i_1'} + P_{i_1'} \geq \beta + \alpha$. Therefore, by (5.4), $C^* \geq \alpha + \beta$, which contradicts the fact that $C \leq \alpha + \beta$. □

LEMMA 5.6. *If* $\beta \leq \alpha$, *then* π_0'' *can be assumed to have this property: at least* $\alpha - \beta$ *of the processing of batch* i_1' *is assigned to machine* 1 *right after batch* $i_1 = b$.

Proof. With the Convention in §5.1 and pairwise interchange arguments we can always make π_0'' have this form without increasing its makespan. □

Now deductions similar to those in Case A lead us to conclude that $C_M(\mathcal{L}_2)/C_M^*(\mathcal{L}_2)$ is upper-bounded by (see [3])

$$R'(l_1'', s', \alpha, s, \beta) = \frac{\frac{1}{2\lambda_2}l_1'' + \alpha + \frac{1}{2}s + r_2\beta}{\max\{s' + \alpha - \beta + 1, \alpha + H(l_1'', 1, s')\}},$$

where $(l_1'', s', \alpha, s, \beta)$ satisfy

$$l_1'' \leq \tfrac{1}{2}(M - 2 + \lambda_2)\beta - \lambda_2 s, \quad \alpha \geq \beta \geq s \geq 0, \quad l_1'' \geq 0,$$
$$s' \geq \max\left\{0, \beta - 1, \tfrac{1}{2}(2\beta + s - \alpha - 1)\right\}.$$

We are to show that $R'(l_1'', s', \alpha, s, \beta)$ is upper-bounded by τ_M.

If $l_1'' > A_1 - s' = (M-1)(s'+1) - s'$, then, as we did in Case A, we can similarly get $R'(l_1'', s', \alpha, s, \beta) \leq 1$. Hence confining ourselves to the case for which $A_k \leq l_1 \leq A_{k-1} - s'$ for $2 \leq k \leq M$ can also be justified. Lemma A.5 in the appendix then completes our final discussion.

6. Conclusions and remarks.

We have not considered the class $\mathcal{L}_1$ of job sets. Apparently, for small-batch problems our heuristic obtains an optimal schedule. If $M = 2$, the optimality holds for any $\mathcal{L}_1$ and the bound τ_M is true for any $\mathcal{L}$ [3]. In conclusion, we have

THEOREM 6.1. *For small-batch problems, which include those in which each batch contains a single job, the heuristic* LBT *has the worst-case performance ratio*

$$\tau_M = \max\left\{\frac{3M}{2M+1}, \frac{3M-4}{2M-2}\right\}.$$

On the other hand, Theorem 3.4 supplies us a clear support for the intuitive expectation that LBT comes close to optimal when many batches are scheduled, relative to the number of machines.

Appendix.

LEMMA A.1. *Given a linear fractional program*

$$P = \sup\left\{\frac{c^T x + \xi}{d^T x + \eta} : Ax \le b,\ x \ge 0\right\},$$

where $c \in \mathbb{R}^n$, $d \in \mathbb{R}^n$, $\xi \in \mathbb{R}$, $\eta \in \mathbb{R}$, and A denotes an $m \times n$ matrix, then the dual can be formulated as the following linear program:

$$D = \inf\left\{y : A^T u + yd \ge c,\ -b^T u + y\eta \ge \xi,\ u \ge 0,\ y \in \mathbb{R}\right\}.$$

We have relation $P = D$ provided that at least one of the two sets is nonempty.

Proof. See [1]. □

COROLLARY A.2. *If the linear system*

$$\begin{pmatrix} A^T \\ -b \end{pmatrix} u \ge \begin{pmatrix} c - \tau_M d \\ \xi - \tau_M \eta \end{pmatrix}, \quad u \ge 0,$$

has a solution, then $P \le \tau_M$. □

In the remaining lemmas, λ_i, r_i $(i = 1, 2)$ are as defined in §5.2 and $M \ge 5$. The basic idea for proving these lemmas is, by using Corollary A.2, simply to find a solution of the corresponding linear system, which can be done, for example, by using the software package *Mathematica*. For details, interested readers are referred to [3].

LEMMA A.3. *Let $2 \le k \le M$, and let*

$$R(\lambda_1, k, M) = \sup \frac{\frac{1}{2\lambda_1}x_1 + r_1 x_2 + x_3 + (r_1 - \frac{1}{2})}{\max\{x_2 + 1,\ \frac{1}{M}x_1 + \frac{k}{M}x_2 + x_3 + \frac{1}{M}\}}$$

subject to

$$Ax \le b,\ x \ge 0,$$

where

$$A = \begin{pmatrix} 1 & -(M-k) & 0 \\ -1 & M-k & 0 \\ 1 & -\frac{M-1-\lambda_1}{2} & 0 \\ 0 & 1 & -1 \end{pmatrix}, \qquad b = \begin{pmatrix} \frac{M-k+1}{k-1} \\ -\frac{M-k}{k} \\ \frac{M-1+\lambda_1}{2} \\ -\frac{1}{2} \end{pmatrix}.$$

Then $R(\lambda_1, k, M) \le \tau_M$.

LEMMA A.4. *Let $2 \le k \le M$, and let*

$$R_1(\lambda_2, k, M) = \sup \frac{\frac{1}{2\lambda_2}x_1 + (1 + \frac{1}{2\lambda_2})x_3 + \frac{1}{2}x_4 + (r_2 - \frac{1}{2\lambda_2})x_5}{\max\{x_2 + 1,\ \frac{1}{M}x_1 + \frac{k}{M}x_2 + x_3 + \frac{1}{M}\}}$$

subject to

$$Ax \le b,\ x \ge 0,$$

where

$$A = \begin{pmatrix} 1 & -(M-k) & 0 & 0 & 0 \\ -1 & M-k & 0 & 0 & 0 \\ 1 & 0 & 1 & \lambda_2 & -\frac{M+\lambda_2}{2} \\ -1 & 0 & -1 & 0 & 1 \\ 0 & -2 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -2 & 1 & 1 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} \frac{M-k+1}{k-1} \\ -\frac{M-k}{k} \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then $R_1(\lambda_2, k, M) \le \tau_M$.

LEMMA A.5. *Let* $2 \le k \le M$, *and let*

$$R_2(\lambda_2, k, M) = \sup \frac{\frac{1}{2\lambda_2}x_1 + x_3 + \frac{1}{2}x_4 + r_2 x_5}{\max\{x_2 + x_3 - x_5 + 1,\ \frac{1}{M}x_1 + \frac{k}{M}x_2 + x_3 + \frac{1}{M}\}}$$

subject to

$$Ax \le b, \ x \ge 0,$$

where

$$A = \begin{pmatrix} 1 & -(M-k) & 0 & 0 & 0 \\ -1 & M-k & 0 & 0 & 0 \\ 1 & 0 & 0 & \lambda_2 & -\frac{M-2+\lambda_2}{2} \\ 0 & -2 & -1 & 1 & 2 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} \frac{M-k+1}{k-1} \\ -\frac{M-k}{k} \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then $R_2(\lambda_2, k, M) \le \tau_M$.

REFERENCES

[1] M. AVRIEL, W. E. DIEWERT, S. SCHAIBLE, AND I. ZANG, *Generalized Concavity*, Plenum Press, New York, 1988.

[2] J. BRUNO AND P. DOWNEY, *Complexity of task sequencing with deadlines, set-up times and changeover costs*, SIAM J. Comput., 7 (1978), pp. 393–404.

[3] B. CHEN, *A better heuristic for preemptive parallel machine scheduling with batch set-up times*, Report 9131/A, Econometric Institute, Erasmus University, Rotterdam, The Netherlands, 1991.

[4] E. G. COFFMAN, JR., AND RAVI SETHI, *A generalized bound on* LPT *sequencing*, Rev. Fr. Automat. Inform. Rech. Oper., 10 suppl. 5 (1976), pp. 17–25.

[5] C. L. MONMA AND C. N. POTTS, *On the complexity of scheduling with batch setup times*, Oper. Res. 37 (1989), pp. 798–804.

[6] ———, *Analysis of heuristics for preemptive parallel machine scheduling with batch set-up times*, Oper. Res., 41 (1993), to appear.

[7] C. N. POTTS AND L. N. VAN WASSENHOVE, *Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity*, J. Opl. Res. Soc., 43 (1992), pp. 395–406.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1319–1330, December 1993

A NEW METHOD FOR COMPUTING PAGE-FAULT RATES[*]

JIANG-HSING CHU[†] AND GARY D. KNOTT [‡]

Abstract. When a program is executed in a caching environment, the caching algorithm can be modeled by an associated finite-state automaton. It is assumed that the finite automaton will reach a steady state after processing a long string. By considering the finite automaton, a formula is obtained for the expected page-fault rate in terms of the steady-state probabilities of the automaton. It is possible to derive the steady-state probabilities for the least-recently-used (LRU) algorithm with order-0 and order-1 programs based on a method that describes the page reference strings as regular expressions. The steady-state behavior for caching algorithms with order-1 programs has never been reported before. This analysis method is then applied to obtain an analysis of the caching behavior of a practical storage-and-retrieval algorithm.

Key words. caching, page-fault rate, regular expressions, order-k programs, analysis of algorithms

AMS subject classification. 68M20, 68Q25, 68Q50

1. Introduction.

Caching is a technique to increase the speed of executing computer programs that make accesses to a secondary memory. By holding the frequently used disk pages in main memory, we can save a considerable number of disk accesses that would otherwise be required.

Caching is usually implemented by organizing part of the main memory into fixed-size blocks of locations called *caches* and organizing the file space into matching-size blocks of memory called *pages*. The cache is a buffer pool that contains a fixed number of buffers. When data within a page is requested by the program, the cache is checked first. If the page is in the cache, no access to the secondary memory is required. Otherwise, a *page fault* is said to occur and the page containing the desired data has to be read from the secondary memory and placed in the cache. If the cache is full, a cached page must be discarded and its cache buffer reused. The *expected page-fault rate* is defined to be the limit of the number of page faults divided by the number of pages requested as the number of page requests tends to ∞.

To use the cache efficiently, we have to minimize the movement of pages in and out. The rules used to decide which pages should be moved and when they should be moved are called *caching algorithms*. Since programs tend to reference part of the address space more frequently than other parts for reasonable periods of time, it might thus be assumed that pages that have been recently referenced have a relatively high probability of being referenced in the near future. Therefore, it is reasonable to remove the page that is least recently used. This defines the least-recently-used (LRU) algorithm. Several other caching algorithms, such as MIN, LFU, and FIFO, have also been studied [1]–[4].

Caching has been a subject of considerable interest for a long time. Besides its use in operating systems, applications of caching can be found in many fields of computer science, including database systems [5], [6], multiprocessor systems [10], network file systems [12], and so on. Smith [14] has given a complete bibliography on caching.

Many empirical analyses on caching performance have been conducted. Some of the recent results can be found in [13], [15], [16]. The most popular method is the trace-driven analysis, which is done by actually running a large program that issues a large number, usually millions, of page references. The caching activity is recorded. Then the miss ratio can be computed from the recorded data.

*Received by the editors April 22, 1991; accepted for publication (in revised form) August 28, 1992.
†Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901.
‡Civilized Software, Inc., 7735 Old Georgetown Road, #410, Bethesda, Maryland 20814.

In order to analytically analyze the expected page-fault rate of a program that is executed in a caching environment, we need a mathematical model to describe the behavior of programs. Several models have been proposed [1], [3], [11], [17]. Among them, the order-0 model (also known as the *independent-reference model*) is most popular because of its simplicity, which makes the analytic analysis tractable.

An order-0 program accesses each page with a fixed probability. Although an order-0 program is analytically tractable because of its simplicity, not many practical programs can be modeled by order-0 programs. Thus we decided to introduce the order-1 model with the hope that more practical programs can be modeled by order-1 programs. An order-1 program accesses each page with a probability that depends on the previous page that was accessed. The accessed pages of an order-1 program thus form a realization of a Markov chain.

King [11] has reported the expected page-fault rates for order-0 programs with several caching algorithms. Flajolet, Gardy, and Thimonier [8] have derived the expected page-fault rates for order-0 programs with LRU. It is not obvious how their work on order-0 models can be generalized to order-1 models. Thus we have developed a new analysis method that we call the *regular-expression method*, which is based upon describing a class of page reference strings as a regular expression. This method can be used to derive the expected page-fault rate for the LRU algorithm with an order-0 model.

A derivation of the expected page-fault rate for the LRU algorithm with an order-1 model is also given in the following section. Using the regular-expression method, we have also been able to derive the expected page-fault rates for several page-replacement algorithms with an order-0 model and order-1 model. Complete details can be found in [3].

2. The LRU algorithm. Consider an order-0 n-page program that is executed in a caching environment with an m-buffer cache that uses the LRU algorithm. Remember that the LRU algorithm always replaces the page in the cache that is least recently used. In order to simulate an LRU algorithm with an automaton, we can let each state be a distinct sequence of page numbers representing the associated ordered set of pages that may occur in the cache. Let the set of $n^{\underline{m}}$ such states be denoted by S_m, where $n^{\underline{m}} = n(n-1)\cdots(n-m+1)$. A state $\langle j_1 j_2 \cdots j_m \rangle \in S_m$ represents the situation where page j_1 is the least recently used page in the cache and page j_m is the most recently used page in the cache. We will use $P_m(\langle j_1 j_2 \cdots j_m \rangle)$ to denote the steady-state probability of state $\langle j_1 j_2 \cdots j_m \rangle$.

As the length of a page-reference string becomes very long, the probability of referencing fewer than m pages in total may be assumed to approach zero. Therefore, in studying LRU caching we will assume at least m different pages are referenced in a long page-reference string.

A page-reference string that results in the state $\langle j_1 j_2 \cdots j_m \rangle$ has the following properties: (1) the last referenced page is page j_m, (2) other than page j_m, the last referenced page is page j_{m-1}, (3) other than pages j_m and j_{m-1}, the last referenced page is page j_{m-2}, and so on. Based on these properties, we see that the set of page-reference strings that result in the state $\langle j_1 j_2 \cdots j_m \rangle$ can be described by the regular expression $(1|\cdots|n)^* j_1 (j_2|\cdots|j_m)^* j_2 (j_3|\cdots|j_m)^* j_3 \cdots j_{m-1} j_m^* j_m$.

Given a random page-reference string of length l, the probability that the page-reference string will result in state $\langle j_1 j_2 \cdots j_m \rangle$ is the probability that the given page-reference string is of the form $(1|\cdots|n)^{l_1} j_1 (j_2|\cdots|j_m)^{l_2} j_2 (j_3|\cdots|j_m)^{l_3} j_3 \cdots j_{m-1} j_m^{l_m} j_m$, with $l_1 \geq 0, \ldots, l_m \geq 0$, and $l_1 + l_2 + \cdots + l_m = l - m$. By letting l approach ∞, we obtain $P_m(\langle j_1 j_2 \cdots j_m \rangle)$, the steady-state probability of state $\langle j_1 j_2 \cdots j_m \rangle$.

3. Order-0 programs with caching. To start with, we assume we are dealing with order-0 programs. In the order-0 model, page references made by an n-page program are a sequence of independent, identically distributed random variables $r_1, r_2, \ldots$ with probability $P(r_k =$

$i) = p_i > 0$, for $k = 1, 2, \ldots$, $1 \le i \le n$, and $\sum_{i=1}^{n} p_i = 1$. Note that we exclude the degenerate cases where some pages will never be referenced.

For an n-page order-0 program, the probability of having a page-reference string of length l that is of the form $(1|\cdots|n)^{l_1} j_1 (j_2|\cdots|j_m)^{l_2} j_2 (j_3|\cdots|j_m)^{l_3} j_3 \cdots j_{m-1} j_m^{l_m} j_m$ is

$$\sum_{\substack{l_1+\cdots+l_m=l-m\\0\le l_1,\ldots,0\le l_m}} \Big(\prod_{i=1}^{m} p_{j_i} \prod_{i=2}^{m}\Big(\sum_{k=i}^{m} p_{j_k}\Big)^{l_i}\Big) = \prod_{i=1}^{m} p_{j_i}\Big(\sum_{\substack{l_1+\cdots+l_m=l-m\\0\le l_1,\ldots,0\le l_m}} \prod_{i=2}^{m}\Big(\sum_{k=i}^{m} p_{j_k}\Big)^{l_i}\Big).$$

If we let l approach ∞, we obtain $P_m(\langle j_1 j_2 \cdots j_m\rangle)$, the steady-state probability of state $\langle j_1 j_2 \cdots j_m\rangle$.

Let

$$z_i = \sum_{k=i}^{m} p_{j_k},$$

and define

$$g(k, l) = \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} \prod_{i=2}^{k} z_i^{l_i}$$

for $1 \le k \le m$. Note that $0 < z_2, \ldots, z_m < 1$ and that the z_i's are dependent on $\langle j_1 j_2 \cdots j_m\rangle$. With these definitions, we thus have

$$P_m(\langle j_1 j_2 \ldots j_m\rangle) = \lim_{l\to\infty}\Big(\prod_{i=1}^{m} p_{j_i}\Big) g(m, l).$$

Note that

$$\begin{aligned}
g(k,l) &= \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} \prod_{i=2}^{k} z_i^{l_i}\\
&= \sum_{l_k=0}^{l-k} z_k^{l_k} \sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l_k\\0\le l_1,\ldots,0\le l_{k-1}}} \prod_{i=2}^{k-1} z_i^{l_i}\\
&= \Big(\sum_{l_k=1}^{l-k} z_k^{l_k} \sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l_k\\0\le l_1,\ldots,0\le l_{k-1}}} \prod_{i=2}^{k-1} z_i^{l_i}\Big) + \sum_{\substack{l_1+\cdots+l_{k-1}=l-k\\0\le l_1,\ldots,0\le l_{k-1}}} \prod_{i=2}^{k-1} z_i^{l_i} \qquad (1)\\
&= \Big(z_k \sum_{l'_k=0}^{l-k-1} z_k^{l'_k} \sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l'_k-1\\0\le l_1,\ldots,0\le l_{k-1}}} \prod_{i=2}^{k-1} z_i^{l_i}\Big) + g_{k-1}(l-1)\\
&= \Big(z_k \sum_{\substack{l_1+\cdots+l'_k=l-k-1\\0\le l_1,\ldots,0\le l'_k}} \prod_{i=2}^{k} z_i^{l_i}\Big) + g_{k-1}(l-1)\\
&= z_k g(k, l-1) + g(k-1, l-1).
\end{aligned}$$

We now show that the limit of $g(k, l)$ as l tends to ∞ exists for $1 \le k \le m$. Note that the existence of the limit can also be proved by using the general ergodic theorem for Markov chains [9]. The proof[1] is an inductive proof on k.

[1]We thank an anonymous referee for pointing out an error in an earlier proof. We also thank Zhiping You for inspiring this proof.

To begin, for $k = 1$, $g(1, l) = 1$ by definition; thus the limit for $k = 1$ exists. Now for $k > 1$, assume the limit of $g(k-1, l)$ exists as l tends to ∞ (induction hypothesis). From equation (1) we have

$$\lim_{l\to\infty}(g(k, l+1) - g(k, l)) = \lim_{l\to\infty}(z_k g(k, l) + g(k-1, l) - z_k g(k, l-1) - g(k-1, l-1)).$$

The induction hypothesis allows us to reduce this equation to

$$\lim_{l\to\infty}(g(k, l+1) - g(k, l)) = z_k \lim_{l\to\infty}(g(k, l) - g(k, l-1)).$$

It is well known that $g(k, l)$ converges if the series $1 + z_k + z_k^2 + \cdots$ converges or, equivalently, if $|z_k| < 1$. Since $0 < z_k < 1$, we conclude that $g(k, l)$ converges as l tends to ∞.

Thus we may define

$$f(k) = \lim_{l\to\infty} g(k, l).$$

From equation (1) we have

$$f(k) = z_k f(k) + f(k-1)$$

or, equivalently,

$$f(k) = \frac{1}{1 - z_k} f(k-1)$$

for $2 \le k \le m$. Repeatedly applying the preceding equation gives us

$$f(m) = \frac{1}{1 - z_m} f(m-1) = \frac{1}{1 - z_m}\frac{1}{1 - z_{m-1}} f(m-2) = \cdots = \Big(\prod_{i=2}^{m} \frac{1}{1 - z_i}\Big) f(1).$$

Since

$$f(1) = \lim_{l\to\infty} g(1, l) = \lim_{l\to\infty} \sum_{l_1 = l-1} \prod_{i=2}^{1} z_i^{l_i} = 1,$$

we have

$$f(m) = \prod_{i=2}^{m} \frac{1}{1 - z_i} = \prod_{i=2}^{m} \frac{1}{1 - \sum_{k=i}^{m} p_{j_k}}.$$

Therefore,

$$\begin{aligned}
P_m(\langle j_1 j_2 \cdots j_m \rangle) &= \lim_{l\to\infty} \prod_{i=1}^{m} p_{j_i} \Big(\sum_{\substack{l_1 + \cdots + l_m = l - m \\ 0 \le l_1, \ldots, 0 \le l_m}} \prod_{i=2}^{m} \Big(\sum_{k=i}^{m} p_{j_k} \Big)^{l_i} \Big) \\
&= \lim_{l\to\infty} \Big(\prod_{i=1}^{m} p_{j_i} \Big) g(m, l) \\
&= \Big(\prod_{i=1}^{m} p_{j_i} \Big) f(m) \\
&= \Big(\prod_{i=1}^{m} p_{j_i} \Big) \prod_{i=2}^{m} \frac{1}{1 - \sum_{k=i}^{m} p_{j_k}} \\
&= \Big(1 - \sum_{i=1}^{m} p_{j_i} \Big) \prod_{i=1}^{m} \frac{p_{j_i}}{1 - \sum_{k=i}^{m} p_{j_k}}
\end{aligned}$$

for any $\langle j_1 j_2 \cdots j_m \rangle \in S_m$.

In a particular state $\langle j_1 j_2 \cdots j_m \rangle$, for an order-0 program the probability of having a page fault at the next reference is $(1 - \sum_{i=1}^{m} p_{j_i})$. Thus the expected page-fault rate $F_{\text{LRU}}^0(m)$ for the LRU algorithm using m buffers with an n-page order-0 program is

$$F_{\text{LRU}}^0(m) = \sum_{\langle j_1 j_2 \cdots j_m \rangle \in S_m} \left(P_m(s) \left(1 - \sum_{i=1}^{m} p_{j_i} \right) \right)$$

(2)

$$= \sum_{\langle j_1 j_2 \cdots j_m \rangle \in S_m} \left(\left(1 - \sum_{i=1}^{m} p_{j_i} \right)^2 \prod_{i=1}^{m} \frac{p_{j_i}}{1 - \sum_{k=i}^{m} p_{j_k}} \right).$$

The same result has been computed by a different method by King [11].

As we can see, this formula is complex because of the summation over the $n^{\underline{m}}$ possible terms. It is not likely that we can derive any analytic results from the formula. However, it is possible to find the asymptotic behavior [7].

4. Order-1 programs with caching. We discuss order-1 programs in this section. An n-page program is said to be an order-1 program if the probability of referencing a page in a given state is a function only of the page that was most recently referenced. Therefore, the probability of referencing a particular page is not a constant, as we have with order-0 programs, but depends on the page previously referenced. We will denote the probability that page j is referenced given that page i was previously referenced by p_{ij}. Note that $\sum_{j=1}^{n} p_{ij} = 1$ for all $1 \leq i \leq n$.

We want to compute $P_m(\langle j_1 j_2 \cdots j_m \rangle)$, the steady-state probability of state $\langle j_1 j_2 \cdots j_m \rangle$, which is equal to the probability that a random page-reference string of length l is of the form $(1|\cdots|n)^{l_1} j_1 (j_2|\cdots|j_m)^{l_2} j_2 (j_3|\cdots|j_m)^{l_3} j_3 \cdots j_{m-1} j_m^{l_m} j_m$, with $l_1 \geq 0, \ldots, l_m \geq 0$ and $l_1 + l_2 + \cdots + l_m = l - m$, if we let l approach ∞.

Define $x_1(l_1)$ to be the probability that the last referenced page of the page-reference string $r_1, \ldots, r_{l_1+1}$ of length $l_1 + 1$ is page j_1, that is, $x_1(l_1) = P(r_{l_1+1} = j_1)$. Also, let $x_i(l_i)$ for $2 \leq i \leq m$ be the probability that if a page-reference string is of length $l_i + 2$, where page j_{i-1} is the first referenced page, then page j_i is the last referenced page and only pages j_i through j_m are referenced in between. That is, $x_i(l_i) = P(r_2, \ldots, r_{l_i+2} \in \{j_i, \ldots, j_m\}$ and $r_{l_i+2} = j_i \mid r_1 = j_{i-1})$ for $2 \leq i \leq m$.

With the preceding definitions we know the probability that a random page-reference string of length l is of the form $(1|\cdots|n)^{l_1} j_1 (j_2|\cdots|j_m)^{l_2} j_2 (j_3|\cdots|j_m)^{l_3} j_3 \cdots j_{m-1} j_m^{l_m} j_m$, with $l_1 \geq 0, \ldots, l_m \geq 0$ and $l_1 + l_2 + \cdots + l_m = l - m$, is

$$\sum_{\substack{l_1 + \cdots + l_m = l - m \\ 0 \leq l_1, \ldots, 0 \leq l_m}} \prod_{i=1}^{m} x_i(l_i).$$

Let

$$A_{q_1 q_2 \ldots q_k} = \begin{bmatrix} 0 & p_{q_1 q_2} & p_{q_1 q_3} & \cdots & p_{q_1 q_k} \\ 0 & p_{q_2 q_2} & p_{q_2 q_3} & \cdots & p_{q_2 q_k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & p_{q_k q_2} & p_{q_k q_3} & \cdots & p_{q_k q_k} \end{bmatrix}.$$

Recall a fundamental theorem in graph theory: If A is the adjacency matrix for a graph G, then $\left[A^k\right]_{ij}$, the element of A^k that is in row i, column j, is the number of paths of length k from vertex v_i to vertex v_j. Similarly, we note that $\left[A^l_{j_{i-1} j_i \cdots j_m}\right]_{12}$ is the probability that if a page-reference string is of length $l+1$, where page j_{i-1} is the first referenced page, then page j_i is the last referenced page and only pages j_i through j_m are referenced in between. Thus $x_i(l_i) = \left[A^{l_i+1}_{j_{i-1}j_i\cdots j_m}\right]_{12}$ for $i \le 2 \le m$.

Now, let us use A_i as a shorthand notation for $A_{j_{i-1}j_i\cdots j_m}$. Define

$$g(k,l) = \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} \prod_{i=1}^{k} x_i(l_i) = \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} x_1(l_1)\prod_{i=2}^{k}\left[A_i^{l_i+1}\right]_{12},$$

where $2 \le k \le m$. Note the x_i's are dependent on $\langle j_1 j_2 \cdots j_m\rangle$.

We also define

$$G(k,l) = \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k^{l_k+1}.$$

Hence $g(k,l) = [G(k,l)]_{12}$. We then have

$$\begin{aligned}
G(k,l) &= \sum_{\substack{l_1+\cdots+l_k=l-k\\0\le l_1,\ldots,0\le l_k}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k^{l_k+1}\\
&= \sum_{l_k=0}^{l-k}\sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l_k\\0\le l_1,\ldots,0\le l_{k-1}}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k^{l_k+1}\\
&= \sum_{l_k=1}^{l-k}\left(\sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l_k\\0\le l_1,\ldots,0\le l_{k-1}}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k^{l_k+1}\right)\\
&\quad + \sum_{\substack{l_1+\cdots+l_{k-1}=l-k\\0\le l_1,\ldots,0\le l_{k-1}}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k \qquad (3)\\
&= \sum_{l'_k=0}^{l-k-1}\left(\sum_{\substack{l_1+\cdots+l_{k-1}=l-k-l'_k-1\\0\le l_1,\ldots,0\le l_{k-1}}} \left(x_1(l_1)\prod_{i=2}^{k-1}\left[A_i^{l_i+1}\right]_{12}\right)A_k^{l'_k+1}\right)A_k + g(k-1,l-1)A_k\\
&= G(k,l-1)A_k + g(k-1,l-1)A_k.
\end{aligned}$$

If we can prove that $G(k,l)$ converges, then $g(k,l)$ also converges. By using the same method used in §3, we can prove that $G(k,l)$ converges if the series $I + A_k + A_k^2 + \cdots$ converges. We know that if the absolute values of all eigenvalues of A_k are less than 1, then this series converges. If we exclude the degenerate cases by assuming that the Markov chain is irreducible and aperiodic, then it is obvious that all eigenvalues of A_k are greater than -1 and less than 1. Thus we conclude that both $g(k,l)$ and $G(k,l)$ converge as l tends to ∞. Note that the convergence of the series $I + A_k + A_k^2 + \cdots$ also implies the existence of $(I - A_k)^{-1}$, which will be needed later. Now we can let

$$F(k) = \lim_{l\to\infty} G(k, l)$$

and

$$f(k) = \lim_{l\to\infty} g(k, l).$$

From (3) we have

$$F(k) = F(k)A_k + f(k-1)A_k,$$

which is equivalent to

$$F(k)(I - A_k) = f(k-1)A_k.$$

Recall that $(I - A_k)^{-1}$ exists. Therefore,

$$F(k) = f(k-1)A_k(I - A_k)^{-1}.$$

From the definitions of F_k and f_k we have

$$f(k) = [F(k)]_{12} = f(k-1)\left[A_k(I - A_k)^{-1}\right]_{12}.$$

Repeatedly applying the preceding equation gives us

$$\begin{aligned}
f(m) &= f(m-1)\left[A_m(I - A_m)^{-1}\right]_{12} \\
&= f(m-2)\left[A_{m-1}(I - A_{m-1})^{-1}\right]_{12}\left[A_m(I - A_m)^{-1}\right]_{12} \\
&\vdots \\
&= f(1)\prod_{i=2}^{m}\left[A_i(I - A_i)^{-1}\right]_{12} \\
&= f(1)\prod_{i=2}^{m}\left[(A_i - I + I)(I - A_i)^{-1}\right]_{12} \\
&= f(1)\prod_{i=2}^{m}\left[-I + (I - A_i)^{-1}\right]_{12} \\
&= f(1)\prod_{i=2}^{m}\left[(I - A_i)^{-1}\right]_{12}.
\end{aligned}$$

We know that

$$f(1) = \lim_{l\to\infty} g(1, l) = \lim_{l_1\to\infty} x_1(l_1) = P_1(\langle j_1\rangle).$$

Thus we have proved that

$$\begin{aligned}
P_m(\langle j_1 j_2 \cdots j_m \rangle) &= \lim_{l\to\infty} g(m,l) \\
&= f(m) \\
&= f(1) \prod_{i=2}^{m} \left[(I - A_i)^{-1}\right]_{12} \\
&= P_1(\langle j_1 \rangle) \prod_{i=2}^{m} \left[(I - A_i)^{-1}\right]_{12} \\
&= P_1(\langle j_1 \rangle) \prod_{i=2}^{m} \left[(I - A_{j_{i-1} j_i \cdots j_m})^{-1}\right]_{12}.
\end{aligned}$$

Consider the case when there is only one buffer in the cache. We have

$$\begin{bmatrix} P_1(\langle 1 \rangle) & \cdots & P_1(\langle n \rangle) \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix} = \begin{bmatrix} P_1(\langle 1 \rangle) & \cdots & P_1(\langle n \rangle) \end{bmatrix}$$

and

$$\sum_{i=1}^{n} P_1(\langle i \rangle) = 1.$$

Thus we can compute $P_1(\langle j_i \rangle)$ for any j_i, where $1 \le j_i \le n$.

In the state $\langle j_1 \cdots j_m \rangle$ the probability that a page fault will occur in the next page reference is $1 - \sum_{i=1}^{m} p_{j_m j_i}$. Thus the expected page-fault rate $F^1_{\text{LRU}}(m)$ for the LRU algorithm that uses m buffers with an n-page order-1 program is

$$\begin{aligned}
F^1_{\text{LRU}}(m) &= \sum_{\langle j_1 j_2 \cdots j_m \rangle \in S_m} \Big(P_m(\langle j_1 j_2 \cdots j_m \rangle)(1 - \sum_{i=1}^{m} p_{j_m j_i}) \Big) \\
&= \sum_{\langle j_1 j_2 \cdots j_m \rangle \in S_m} \Big(P_1(\langle j_1 \rangle) \prod_{i=2}^{m} \left[(I - A_{j_{i-1} j_i \cdots j_m})^{-1}\right]_{12} (1 - \sum_{i=1}^{m} p_{j_m j_i}) \Big).
\end{aligned} \tag{4}$$

We can see this formula is more complex than the formula for the order-0 programs. It makes sense to use this approach only if the accuracy of the estimated page-fault rate is sufficiently improved. An example is presented in the next section.

5. An example. We will consider an example where we store random items into a binary storage tree and study the expected page-fault rate both analytically and empirically by assuming that each item is equally likely to be retrieved. We consider both an order-0 model and an order-1 model for this example in order to estimate the expected page-fault rate. To our knowledge, this is the first partially successful caching analysis for a real storage-and-retrieval algorithm of practical interest.

Note that a binary-storage-tree retrieval program is not *truly* an order-1 program because the probability that we will reference a page depends on the currently referenced *item*; for a program to be an order-1 program, the probability that we will read a page should depend only on the currently referenced *page*. Nevertheless, experiments have empirically shown that a binary-storage-tree retrieval program can be well approximated as an order-1 page-reference string.

We will use the following simple example to illustrate how the experiments were conducted. Assume there are two buffers in the cache. Suppose a page can accommodate two nodes of a binary storage tree. After storing six items, assume we end up with the binary storage tree shown in Fig. 1.

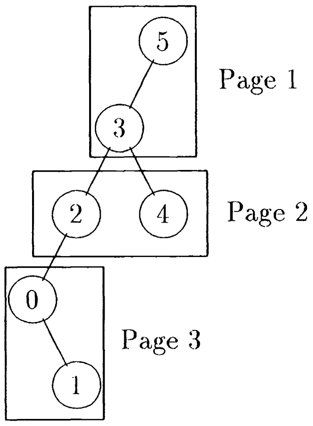


FIG. 1. *A binary storage tree with* 3 *pages.*

When data item 0 is referenced, the sequence of the probes that occur in order to retrieve data item 0 is ⟨5320⟩. This item-probe-sequence string generates the associated page-reference string $\langle P_1 P_1 P_2 P_3 \rangle$, where P_i is a page reference to page i for $1 \leq i \leq 3$. Table 1 shows the probe sequences and the associated page-reference strings with respect to each distinct data item that is selected to be retrieved.

TABLE 1
Page reference strings.

Retrieved item	Probe sequence	Page-reference string
0	⟨5320⟩	$\langle P_1 P_1 P_2 P_3 \rangle$
1	⟨53201⟩	$\langle P_1 P_1 P_2 P_3 P_3 \rangle$
2	⟨532⟩	$\langle P_1 P_1 P_2 \rangle$
3	⟨53⟩	$\langle P_1 P_1 \rangle$
4	⟨534⟩	$\langle P_1 P_1 P_2 \rangle$
5	⟨5⟩	$\langle P_1 \rangle$

Since we assume each item is equally likely to be retrieved, it follows that each page-reference string in Table 1 will occur with the same probability. There are a total of 18 page references in the six page-reference strings, where 11 references are made to page 1, 4 references are made to page 2, and 3 references are made to page 3. In an order-0 model for the tree shown in Fig. 1, we thus estimate that $p_1 = 11/18$, $p_2 = 4/18$, and $p_3 = 3/18$, where p_i is the probability that page i is referenced. We then plug these page-reference probability values, p_1, p_2, and p_3, into formula (2) to obtain the order-0 expected LRU page-fault rate.

Next we switch to an order-1 model to estimate the expected page-fault rate. Note that the last page reference of each page-reference string is followed by P_1 because the next retrieval will reference page 1 first. From Table 1 we find that P_1 is followed by P_1 seven times, is followed by P_2 four times, and is never followed by P_3. Therefore, we conclude that in an order-1 model we should choose $p_{11} = 7/11$, $p_{12} = 4/11$, and $p_{13} = 0$, where p_{ij} is the probability that page j is to be referenced given that page i is just previously referenced. Similarly, we can compute p_{21}, p_{22}, p_{23}, p_{31}, p_{32}, and p_{33}. We can then estimate the expected page-fault rate for order-1 model by the formula (4) obtained earlier.

We also recorded the page-fault rate seen when we performed a sequence of retrieval operations in this binary storage tree. We randomly retrieved data items, which were, in turn, translated into a page-reference string, which was used as the input to the LRU page-replacement algorithm with 2 buffers.

Our experiments consisted of 100 repetitions of the following procedure:

1. randomly store 1000 items in a binary storage tree, which occupies 10 pages;
2. estimate order-0 probabilities and order-1 probabilities;
3. for m, the number of buffers, varying from 2 to 5, perform random retrievals (100,000 page references) from this binary tree and observe the page-fault rate;
4. from the probabilities associated with the observed reference strings obtained in step 3, compute the order-0 and order-1 estimated page-fault rates.

Two typical instances are shown graphically in Fig. 2.

The average relative errors of the order-0 estimations and the order-1 estimations for different cache sizes (numbers of buffers) are shown in Table 2.

TABLE 2
Average relative errors.

Number of buffers	Order-0	Order-1
2	10.78%	0.43%
3	10.18%	0.59%
4	14.02%	0.94%
5	10.53%	1.23%

Not surprisingly, it turns out that the order-0 model yields a poor approximation, whereas the order-1 model accurately estimates the observed page-fault rate.

6. Conclusion.

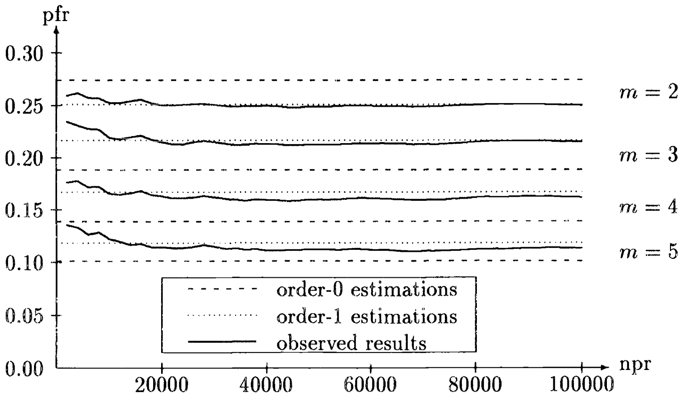
We have presented order-1 models as a generalization of order-0 models. The order-1 models are more powerful in describing the caching behavior of programs. We have also developed a new analysis method,which we call the regular-expression method, that is based upon describing a class of page reference strings as a regular expression.

With the help of the regular-expression method, we can rederive the expected page-fault rate for the LRU caching algorithm with an order-0 model. Using the regular-expression method, we have also been able to derive original results about the expected page-fault rates for several page-replacement algorithms with an order-1 model.

It is not likely that our result for an order-1 model can be further simplified. It would be interesting to find a simpler description of the asymptotic behavior of equation (4).

Although the FIFO algorithm looks simpler than the LRU algorithm in the sense that no page rearrangement is needed when the referenced page is already in the cache, nevertheless, we are not able to compute the expected order-1 page-fault rate for the FIFO algorithm. This remains an open problem.

Acknowledgments. We would like to thank the anonymous referees for careful reading, for constructive criticism, and for pointing out several helpful references.



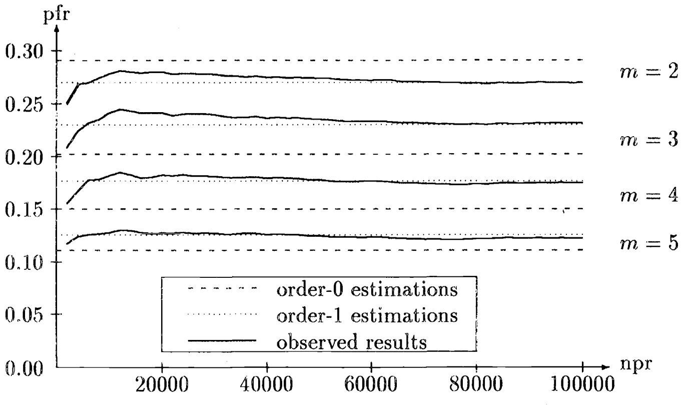


FIG. 2. *Page-fault rate* (pfr) *versus number of page references* (npr).

REFERENCES

[1] A. V. AHO, P. J. DENNING, AND J. D. ULLMAN, *Principles of optimal page replacement*, J. Assoc. Comput. Mach., 18 (1971), pp. 80–93.

[2] L. A. BELADY, *A study of replacement algorithms for a virtual-storage computer*, IBM Syst. J., 5 (1966), pp. 78–101.

[3] J.-H. CHU, *An Analysis of Caching with an Application to Binary Storage Trees*, Ph.D. thesis, Computer Science Department, University of Maryland, College Park, MD, 1989.

[4] P. J. DENNING, Y. C. CHEN, AND G. S. SHEDLER, *A model for program behavior under demand paging*, Tech. Report RC-2301, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1968.

[5] W. EFFELSBERG AND T. HAERDER, *Principles of database buffer management*, ACM Trans. Database Systems, 9 (1984), pp. 560–595.

[6] K. ELHARDT AND R. BAYER, *A database cache for high performance and fast restart in database systems*, ACM Trans. Database Systems, 9 (1984), pp. 503–525.

[7] R. FAGIN, *Asymptotic miss ratios over independent references*, J. Comput. System Sci., 14 (1977), pp. 222–250.

[8] P. FLAJOLET, D. GARDY, AND L. THIMONIER, *Random allocations and probabilistic languages*, in Proc. 15th International Colloquium on Automata, Languages and Programming, Tampere, Finland, 1988, pp. 239–253.

[9] G. R. GRIMMET AND D. R. STIRZAKER, *Probability and Random Processes*, Clarendon Press, Oxford, 1985.

[10] A. R. KARLIN, M. S. MANASSE, L. RUDOLPH, AND D. D. SLEATOR, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[11] W. F. KING, *Analysis of demand paging algorithms*, in Proc. International Federation for Information Processing Congress, Ljubljana, Yugoslavia, 1971, pp. TA-3-155–TA-3-159.

[12] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, *Caching in the sprite network file system*, ACM Trans. Comput. Systems, 6 (1988), pp. 134–154.

[13] P. Palvia, *The effect of buffer size on pages accessed in random files*, Inform. Systems, 13 (1988), pp. 187–191.

[14] A. J. Smith, *Bibliography on paging and related topics*, Operating Systems Rev., 12 (1978), pp. 39–56.

[15] ———, *Disk cache-miss ratio analysis and design considerations*, ACM Trans. Comput. Systems, 3 (1985), pp. 161–203.

[16] H. S. Stone, J. L. Wolf, and J. Turek, *Optimal partitioning of cache memories*, Tech. Report RC-14444, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1989.

[17] R. Turner and B. Strecker, *Use of the lru stack depth distribution for simulation of paging behavior*, Comm. ACM, 20 (1977), pp. 795–798.

SIAM J. COMPUT.
Vol. 22, No. 6, pp. 1331–1348, December 1993

LEARNING DECISION TREES USING THE FOURIER SPECTRUM*

EYAL KUSHILEVITZ† AND YISHAY MANSOUR‡

Abstract. This work gives a *polynomial* time algorithm for learning *decision trees* with respect to the uniform distribution. (This algorithm uses *membership queries.*) The decision tree model that is considered is an extension of the traditional boolean decision tree model that allows linear operations in each node (i.e., summation of a subset of the input variables over $GF(2)$).

This paper shows how to learn in polynomial time any function that can be approximated (in norm L_2) by a polynomially sparse function (i.e., a function with only polynomially many nonzero Fourier coefficients). The authors demonstrate that any function f whose L_1-norm (i.e., the sum of absolute value of the Fourier coefficients) is polynomial can be approximated by a polynomially sparse function, and prove that boolean decision trees with linear operations are a subset of this class of functions. Moreover, it is shown that the functions with polynomial L_1-norm can be learned *deterministically*.

The algorithm can also *exactly identify* a decision tree of depth d in time polynomial in 2^d and n. This result implies that trees of logarithmic depth can be identified in polynomial time.

Key words. machine learning, decision trees, Fourier transform

AMS subject classifications. 42A16, 68Q20, 68T05

1. Introduction.

In recent years much effort has been devoted to providing a theoretical basis for machine learning. These efforts involved formalization of learning models and algorithms, with a special emphasis on polynomial running time algorithms (see [Val84], [Ang87]). This work further extends our understanding of the learning tasks that can be performed in polynomial time.

Recent work by [LMN89] has established the connection between the Fourier spectrum and learnability. They presented a quasi-polynomial-time (i.e. $O(n^{poly-log(n)})$) algorithm for learning the class AC^0 (polynomial size constant depth circuits), where the quality of the approximation is judged with respect to the uniform distribution (and n is the number of variables). Their main result is an interesting property of the representation of the Fourier transform of AC^0 circuits. Using this property, they derive the learning algorithm for this class of functions. [FJS91] has extended the result to apply also to mutually independent distributions (i.e., product distributions) with a similar running time (i.e. quasi-polynomial time). In [AM91] polynomial time algorithms are given for learning both decision *lists* and μ-decision trees (a boolean decision tree in which each variable appears only *once*) with respect to the uniform distribution. As in [LMN89] these algorithms make use of special properties of the Fourier coefficients and approximate the target function by observing examples drawn according to the uniform distribution. More information about Fourier transform over finite groups is found in [Dia88].

In this work we show another interesting application of the Fourier representation that is applied to achieve learnability. The learning model allows membership queries, where the learner can query the (unknown) function on any input. Our main result is a polynomial-time algorithm for learning functions computed by *boolean decision trees* with *linear operations* (over $GF(2)$). In these trees each node computes a summation (modulo 2) of a subset of the n boolean input variables, and branches according to whether the sum is zero or one. Clearly,

*Received by the editors August 16, 1991; accepted for publication (in revised form) September 1, 1992. A preliminary version appeared in the Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, pages 455-464, May 1991.

†Department of Computer Science, Technion, Haifa 32000, Israel. Present address, Aiken Computation Laboratory, Harvard University, Cambridge Massachusetts 02138.

‡Aiken Computation Laboratory, Harvard University, Cambridge Massachusetts 02138. This author was partially supported by Office of Naval Research grant N00014-85-K-0445.

this is an extension of the traditional boolean decision-tree model, since we can still test single variables. On the other hand, we can test in a single operation the parity of all the input variables, compared with a lower bound of 2^n nodes in the traditional model (see [BHO90]).

An interesting consequence of our construction is that one can *exactly* find the Fourier transform representation of boolean decision trees with linear operations in time $poly(n, 2^d)$, where d is the depth of the tree. This implies that we find a function that is *identical* to the tree for any boolean input. A corollary of this result is that decision trees with logarithmic depth can be exactly identified in polynomial time. (Note that enumeration, even of constant depth trees, would require exponential time (due to the linear operation); even eliminating the linear operations and constraining each node to contain a single variable, the number of trees of depth d is $\Omega(n^d)$.)

Our main result—the learning algorithm for decision trees—is achieved by combining the following three results:

- *The algorithmic tool*—We present a randomized polynomial time algorithm that performs the following task. The algorithm receives as an input a boolean function f that can be approximated by a polynomially sparse function g (a function with a polynomial number of nonzero Fourier coefficients) such that the expected error square (i.e., $E(f - g)^2$) is bounded by ε. The algorithm finds some polynomially sparse function h that approximates f, such that $E(f - h)^2 = O(\varepsilon)$. The algorithm we develop here is based on the ideas of [GL89].
- We consider the class of functions $\mathcal{F} = \{f : L_1(f) \leq poly(n)\}$, where $L_1(f)$ is the L_1-norm of the coefficients (i.e., the sum of the absolute value of the coefficients). We show that in order to achieve an approximation of a function $f \in \mathcal{F}$ within ε, it is sufficient to consider only coefficients larger than $\varepsilon/L_1(f)$ (there are at most $(L_1(f)/\varepsilon)^2$ such coefficients). Therefore, every function in the class $\mathcal{F}$ can be approximated by a polynomially sparse function and therefore can be learned in polynomial time by our algorithm.
- We prove that the L_1-norm of the coefficients of a decision tree is bounded by the number of nodes in the tree. Therefore, polynomial size decision trees are in the class $\mathcal{F}$. It follows that every polynomial size decision tree with linear operations can be learned in polynomial time.

Furthermore, for functions in the class $\mathcal{F}$ we show how to derandomize the learning algorithm. The derandomization uses constructions of "small," "almost unbiased" probability spaces, called λ-bias distributions [NN90], [AGHP90]. (For a formal definition of λ-bias probability distributions see §4.1.) Thus, we derive a *deterministic* polynomial time algorithm for learning decision trees.

Our technique sheds a new light on the possibilities of using λ-bias distributions for derandomization. We show that the deviation of the expected value of a function f with respect to the uniform distribution and a λ-bias distribution is bounded by $\lambda \cdot L_1(f)$. One nice example where this bound comes in handy is for showing that the deviation of the AND of a subset of the n variables is bounded by 3λ. (This is since $L_1(AND) \leq 3$, independent of the subset of variables or its size.)

1.1. Relations to other works. Our result could be contrasted with the result of [EH89], where an $O(n^{\log m})$ algorithm is given for learning decision trees in the PAC model, where n is the number of variables and m is the number of nodes in the tree. Their algorithm learns traditional boolean decision trees with respect to an arbitrary distribution, and uses only examples drawn from that distribution. Therefore, it learns in a weaker model. On the other hand, it runs in time $O(n^{\log m})$ compared to the polynomial time of our algorithm. Also, our algorithm handles a stronger model of boolean decision trees, which include linear operations, while the algorithm of [EH89] does not seem to extend to such a model. In [Han90] a polynomial-time algorithm was presented for learning μ-decision trees using membership queries and equivalence queries, and in [Han91] a polynomial time algorithm was presented for learning decision trees in which each variable appears at most a constant number of times. (Again, these results do not address linear operations.)

Recently, Bellare [Bel92] was able to extend a few of our results concerning decision trees and show how to derive an upper bound on the sum of the Fourier coefficients as a function of the predicates in the nodes. He also extends the learning algorithm to the case of product distributions and shows that if the L_1-norm of f (with respect to a product distribution μ) is polynomially bounded, then it can be learned (with respect to μ) in polynomial time. Unfortunately, this result falls short of showing that decision trees are learnable with respect to product distributions, since there are functions (e.g., the AND function) that have a small size decision tree but their L_1-norm is exponential with respect to some product distributions.

Following our work, it has been shown [Man92] how to learn DNF formulas, with respect to the uniform distribution, in $O(n^{\log \log n})$ time. The main contribution of that work is made by bounding the number of "large" coefficients in the Fourier expansion of such a function by $O(n^{\log \log n})$. Then the algorithm of this paper is used to recover them.

In the work of [RB91] the same learning model was considered (i.e., using membership queries and testing the hypothesis with respect to the uniform distribution). They show that any polynomial over $GF(2)$ with polynomial number of terms can be learned in polynomial time in such a model. The class of polynomials with polynomial number of terms (considered in [RB91]) and the class of boolean decision trees with linear operations (considered in our work) are incomparable. On the one hand, the inner-product function has a small polynomial but does not have a small decision tree. On the other hand, consider a boolean decision list with $\log n$ nodes, where each node computes the sum of $\Omega(n)$ variables. Representing such a decision list by a polynomial may require $\Omega(n^{\log n})$ terms.

The power of polynomial size boolean decision trees with linear operations is also incomparable to AC^0 circuits (which are the target of the learning algorithm of [LMN89]). Such trees can compute parity, which cannot be approximated by AC^0 circuits (see [FSS84], [Ajt83], [Yao85], [Has86]). We show that for boolean decision trees with linear operations the L_1-norm is bounded by the number of nodes; therefore, computing a polynomial-size DNF that has an exponential L_1-norm would require an exponential number of nodes (see [BS90] for a construction of such a DNF).

The class $\mathcal{F}$ of boolean functions whose L_1-norm is polynomially bounded was also studied in [Bru90], [BS90], [SB91]. They showed that any such function f can be approximated by a sparse polynomial of a certain form. Note, however, that their notion of approximation is different than ours. Another type of approximation for boolean functions was recently suggested in [ABFR91] (and then studied by others). In that work, boolean functions are approximated by the sign of a low-degree polynomial over the integers.

1.2. Organization. The rest of this paper is organized as follows. Section 2 has the definitions of Fourier transform, decision trees, and the learning model. Section 3 includes the procedure that finds the approximating sparse function. In §4 we prove the properties of functions with small L_1-norm. In §5 we prove the results about boolean decision trees with linear operations. Finally, in §6 we discuss some extensions and mention some open problems.

2. Preliminaries.

In this section we give the definition of Fourier transform and recall some known properties of it (§2.1). Then, we formally define the model of decision trees, which is used in this work (§2.2). We end by describing the membership-queries learning model, which is used in this work (§2.3).

2.1. Fourier transform. Let $f : \{0,1\}^n \rightarrow \Re$ be a real function. Denote by $E[f]$ the expected value of $f(x)$ with respect to the uniform distribution on x, i.e., $E[f] = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)$. The set of all real functions on the cube $\mathbf{Z}_2^n$ is a 2^n-dimensional real vector space with an inner product defined by

$$< g, f > \overset{\triangle}{=} \frac{1}{2^n}\sum_{x\in\{0,1\}^n} g(x)f(x) = E[gf].$$

The *norm* of a function f is defined by $\|f\|_2 \overset{\triangle}{=} \sqrt{< f, f >} = E[f^2]$.

Define a basis for the linear space of real functions on the cube $\mathbf{Z}_2^n$, using the *characters* of $\mathbf{Z}_2^n$ as follows: For each $z \in \{0,1\}^n$, define the basis function χ_z:

$$\chi_z(x_1,\cdots,x_n) \overset{\triangle}{=} \begin{cases} +1 & \text{if } \sum_i z_i x_i \bmod 2 = 0 \\ -1 & \text{if } \sum_i z_i x_i \bmod 2 = 1. \end{cases}$$

The following properties of these functions can be verified easily:

- For every two vectors $z_1, z_2 \in \{0,1\}^n$: $\chi_{z_1}\chi_{z_2} = \chi_{z_1\oplus z_2}$, where $\oplus$ denotes bitwise exclusive-or.
- The family of functions $\{\chi_z : z \in \{0,1\}^n\}$ forms an orthonormal basis. That is, (1) any function $f(x)$ on the cube $\mathbf{Z}_2^n$ can be uniquely expressed as $\sum_z \hat{f}(z)\chi_z(x)$, where $\hat{f}(z)$ are real constants; and (2) if $z_1 \neq z_2$, then $< \chi_{z_1}, \chi_{z_2} >= 0$, and for every z, $< \chi_z, \chi_z >= 1$.

The Fourier transform of f is just the expansion of f as a linear combination of the χ_z's. Since the χ_z's are an orthonormal basis, Fourier coefficients are found via

$$\hat{f}(z) =< f, \chi_z >= E[f\chi_z].$$

The orthonormality of the basis implies *Parseval's identity*:

$$E[f^2] = \|f\|_2^2 = \sum_{z\in\{0,1\}^n} \hat{f}^2(z).$$

Note that if for every $x \in \mathbf{Z}_2^n$, $|f(x)| \leq 1$, then $\|f\|_2 \leq 1$ and therefore for every $z \in \{0,1\}^n$, $|\hat{f}(z)| \leq 1$. Finally, we define $L_1(f)$ as the L_1-norm of the coefficients of f, i.e., $L_1(f) \overset{\triangle}{=} \sum_z |\hat{f}(z)|$.

2.2. Boolean decision trees. In this section we give a precise definition of the *decision tree* model used in this work. This model is much stronger than the traditional decision tree model.

A *boolean decision tree* T consists of a labeled binary tree. Each inner node v of the tree is labeled by a set $S_v \subseteq \{1,\ldots,n\}$ and has two outgoing edges. Every leaf of the tree is labeled by either $+1$ or -1. (Throughout this paper a function is called *boolean* if its range is $\{+1,-1\}$.)

Given an input, $x = (x_1,\ldots,x_n)$, the decision tree defines a *computation.* The computation traverses a path from the root to a leaf and assigns values to the nodes on the path in the following way. The computation starts at the root of the tree T. When the computation arrives at an inner node v, labeled by S_v, it assigns the node v the value $\sum_{i\in S_v} x_i \bmod 2$, which we denote by $val(v)$. If $val(v) = 1$, then the computation continues to the right son of v, otherwise it continues to the left son. The computation terminates at a leaf u and outputs the label of u (which is also the value of the leaf). The value of the tree on an input is the value of the output of the computation.

Note that if, for example, $|S_v| = 1$, then the meaning of the operation is testing the value of a single variable which is the only permitted operation in the traditional decision tree model. If, for example, $|S_v| = 2$, then the meaning of the operation is testing whether the two corresponding variables are equal, and if $|S_v| = n$, then in a single operation we have a test for the parity of all variables. In the traditional decision tree model computing the parity of all the variables requires 2^n nodes.

2.3. Learning model. The learner in our model uses only membership queries. That is, it can query the unknown function f on any input $x \in \{0,1\}^n$ and receive $f(x)$. After performing a finite number of membership queries, the learner outputs an hypothesis h. The error of an hypothesis h, with respect to the function f, is defined to be $error(f,h) \overset{\Delta}{=} Prob[f(x) \neq h(x)]$, where x is distributed uniformly over $\{0,1\}^n$.

A randomized algorithm A *learns* a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ and $\varepsilon, \delta > 0$ the algorithm outputs an hypothesis $h = A(f, \varepsilon, \delta)$ such that

$$\text{Prob}[\text{error}(f,h) \geq \varepsilon] \leq \delta.$$

The algorithm A learns in *polynomial time* if its running time is polynomial in n, $1/\varepsilon$, and $\log 1/\delta$.

We also discuss deterministic learning algorithms. An algorithm A *deterministically learns* a class of functions $\mathcal{F}$ if for every $f \in \mathcal{F}$ and $\varepsilon > 0$ the algorithm outputs an hypothesis $h = A(f, \varepsilon)$ such that

$$\text{error}(f,h) \leq \varepsilon.$$

The algorithm A learns in *deterministic polynomial time* if its running time is polynomial in n and $1/\varepsilon$. Note that in a deterministic algorithm we do not have a parameter δ. That is, the algorithm always succeeds in finding a "good" hypothesis.

A (real) function g *ε-approximates* f (in norm L_2) if $E[(f(x) - g(x))^2] \leq \varepsilon$. In the case that f is a boolean function, we can convert a real prediction function g to a boolean prediction by predicting the sign of g. In such a case, if $f(x) \neq \text{sign}(g(x))$ then $|f(x) - g(x)| \geq 1$, which implies

$$\text{Prob}[f(x) \neq \text{sign}(g(x))] \leq E[(f(x) - g(x))^2] \leq \varepsilon.$$

Thus, we have

CLAIM 2.1. *If g ε-approximates a boolean function f, then*

$$\text{Prob}[f(x) \neq \text{sign}(g(x))] \leq \varepsilon.$$

3. Approximation by sparse functions.

In this section we show how to find an approximation by a sparse function. The main result in this section is that if f can be ε-approximated by some polynomially sparse function g, then there is a randomized polynomial time procedure that finds some function h that $O(\varepsilon)$-approximates f. (A function g is *t-sparse* if it has at most t Fourier coefficients that are not 0.)

The first step is to show that if f can be approximated by a polynomially sparse function g, it can be approximated by a polynomially sparse function that has only "large" coefficients. We remark that we do not make a "direct" use of g (e.g., by approximating g instead of approximating f) but only use its existence in the analysis.

LEMMA 3.1. *If f can be approximated by a t-sparse function g such that $E[(f-g)^2] \leq \varepsilon$, then there exists a t-sparse function h such that $E[(f-h)^2] \leq \varepsilon + O(\varepsilon^2/t)$ and all the nonzero coefficients of h are at least ε/t.*

Proof. Let $g(x) = \sum_{i=1}^{t} \hat{g}(z_i)\chi_{z_i}(x)$. Note that the Fourier coefficients of the function $f - g$ are exactly $\hat{f}(z) - \hat{g}(z)$. Therefore, by Parseval's equality,

$$E[(f-g)^2] = \sum_z (\hat{f}(z) - \hat{g}(z))^2.$$

Thus, requiring that $\hat{g}(z_i) = \hat{f}(z_i)$ can only reduce the expected error squared. Therefore, without loss of generality, the nonzero coefficients of g are the coefficients of f, i.e., $g(x) = \sum_{i=1}^{t} \hat{f}(z_i)\chi_{z_i}(x)$. Let h be the function obtained from g by replacing the "small" coefficients by 0. Namely,

$$h(x) = \sum_{\hat{f}(z_i)\geq\varepsilon/t} \hat{f}(z_i)\chi_{z_i}(x).$$

We now show that $E[(f-h)^2] \leq \varepsilon + O(\varepsilon^2/t)$. Consider the expression,

$$E[(f-h)^2] - E[(f-g)^2].$$

By the above arguments, this is equal to

$$\begin{aligned}\sum_z (\hat{f}(z) - \hat{h}(z))^2 - \sum_z (\hat{f}(z) - \hat{g}(z))^2 \\ &= \sum_{\hat{f}(z_i)<\varepsilon/t} (\hat{f}(z_i) - \hat{h}(z_i))^2 \\ &= \sum_{\hat{f}(z_i)<\varepsilon/t} \hat{f}^2(z_i) < \left(\frac{\varepsilon}{t}\right)^2 t = \varepsilon^2/t.\end{aligned}$$

Since $E[(f-g)^2] \leq \varepsilon$, the lemma follows. □

The above lemma has reduced the problem of approximating f by a t-sparse function to the problem of finding all the coefficients of f that are greater than a threshold of ε/t. Note that the function h defined above does not necessarily contain all the coefficients of f that are greater than ε/t, but only those that appear also in g. However, adding these coefficients to h will clearly make h a better approximation for f. In any case, the number of these coefficients, as follows from Lemma 3.4, cannot be too high.

In the remainder of this section we show a randomized polynomial time procedure that given a function f and a threshold θ outputs (with prob. $1-\delta$) all the coefficients for which $|\hat{f}(z)| \geq \theta$. The procedure runs in polynomial time in n, $1/\theta$ and $\log 1/\delta$. This procedure is based on the ideas of [GL89], although the context is completely different.

Let $f(x) = \sum_{z\in\{0,1\}^n} \hat{f}(z)\chi_z(x)$. For every $\alpha \in \{0,1\}^k$, we define the function $f_\alpha : \{0,1\}^{n-k} \to \Re$ as follows:

$$f_\alpha(x) \stackrel{\Delta}{=} \sum_{\beta\in\{0,1\}^{n-k}} \hat{f}(\alpha\beta)\chi_\beta(x).$$

In other words, the function $f_\alpha(x)$ includes all the coefficients $\hat{f}(z)$ of f such that z starts with α (and all the other coefficients are 0). This immediately gives the key idea for how to find the large coefficients of f: find (recursively) the large coefficients of f_0 and f_1. Note that during the learning process we can only query for the value of the target function f in certain points. Therefore, we first have to show that $f_\alpha(x)$ can be efficiently computed using such queries to f. Actually, we need not compute the exact value of $f_\alpha(x)$ but just need to approximate it. The following lemma gives an equivalent formulation of f_α, which is computationally much more appealing:

LEMMA 3.2. *For any function f, any $1 \le k < n$, any $\alpha \in \{0,1\}^k$, and any $x \in \{0,1\}^{n-k}$,*

$$f_\alpha(x) = E_{y\in\{0,1\}^k}[f(yx)\chi_\alpha(y)].$$

This formulation implies that even though we do not know how to compute the value of $f_\alpha(x)$ we can approximate it, by approximating the above expectation.

Proof. Let $f(yx) = \sum_z \hat{f}(z)\chi_z(yx)$. Note that if $z = z_1z_2$, where $z_1 \in \{0,1\}^k$, then $\chi_z(yx) = \chi_{z_1}(y)\chi_{z_2}(x)$. Therefore,

$$\begin{aligned} E_y[f(yx)\chi_\alpha(y)] &= E_y\left[\left(\sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_1}(y)\chi_{z_2}(x)\right)\chi_\alpha(y)\right] \\ &= \sum_{z_1}\sum_{z_2}\hat{f}(z_1z_2)\chi_{z_2}(x)E_y[\chi_{z_1}(y)\chi_\alpha(y)], \end{aligned}$$

where y and z_1 denote strings in $\{0,1\}^k$ and z_2 denotes strings in $\{0,1\}^{n-k}$. By the orthonormality of the basis, (see §2.1) it follows that $E_y[\chi_{z_1}(y)\chi_\alpha(y)]$ (which is the same as $<\chi_{z_1}, \chi_\alpha>$) equals 0 if $z_1 \ne \alpha$, and equals 1 if $z_1 = \alpha$. Therefore, only the terms with $z_1 = \alpha$ contributes in the sum. Thus, the last term equals

$$\sum_{z_2\in\{0,1\}^{n-k}} \hat{f}(\alpha z_2)\chi_{z_2}(x) = f_\alpha(x). \qquad \square$$

Since both $|f(x)| = 1$ and $|\chi_\alpha(y)| = 1$ we derive the following corollary on the value of $f_\alpha(x)$.

COROLLARY 3.3. *For any boolean function f, any $1 \le k < n$, any $\alpha \in \{0,1\}^k$, and any $x \in \{0,1\}^{n-k}$,*

$$|f_\alpha(x)| \le 1.$$

We showed how to decompose a function f into functions f_α, $\alpha \in \{0,1\}^k$, such that each coefficient of f appears in a unique f_α. Recall that our aim is to find the coefficients $\hat{f}(z)$ such that $|\hat{f}(z)| \ge \theta$. The next lemma claims that this cannot hold for "too many" values of z, and that the property $E[f_\alpha^2] \ge \theta^2$ cannot hold for "many" α (of length k) simultaneously.

LEMMA 3.4. *Let f be a boolean function, and $\theta > 0$. Then,*

1. *At most $1/\theta^2$ values of z satisfy $|\hat{f}(z)| \ge \theta$.*
2. *For any $1 \le k < n$, at most $1/\theta^2$ functions f_α with $\alpha \in \{0,1\}^k$ satisfy $E[f_\alpha^2] \ge \theta^2$.*

Proof. By the assumption that f is a boolean function combined with Parseval's equality, we get

$$\sum_{z\in\{0,1\}^n} \hat{f}^2(z) = E[f^2] = 1.$$

Therefore, (1) immediately follows. Similarly, using the definition of f_α,

$$E[f_\alpha^2] = \sum_{\beta\in\{0,1\}^{n-k}} \hat{f}^2(\alpha\beta).$$

Thus, if $|\hat{f}(\alpha\beta)| \ge \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \ge \theta^2$. By the above two equalities, the following holds.

$$\sum_{\alpha\in\{0,1\}^k} E[f_\alpha^2] = E[f^2] = 1.$$

Therefore, at most $1/\theta^2$ functions f_α have $E[f_\alpha^2] \ge \theta^2$, which completes the proof of (2). $\square$

The algorithm. By now the algorithm for finding the large coefficients of a function f should be rather obvious. It is described by the recursive subroutine `Coef`, which appears in Fig. 1. We start the algorithm by calling `Coef`(λ), where λ is the empty string.

```
SUBROUTINE Coef($\alpha$)
        IF $E[f_\alpha^2] \geq \theta^2$ THEN
                IF $|\alpha| = n$ THEN OUTPUT $\alpha$
                        ELSE Coef($\alpha 0$); Coef($\alpha 1$);
```

FIG. 1. *Subroutine* `Coef`.

As mentioned earlier in this section, we know that each coefficient of f_α appears in exactly one of $f_{\alpha 0}$ and $f_{\alpha 1}$, and also that if $|\hat{f}(\alpha\beta)| \geq \theta$, for some $\beta \in \{0,1\}^{n-k}$, then $E[f_\alpha^2] \geq \theta^2$ (note that when $|\alpha| = n$, then $E[f_\alpha^2] = \hat{f}^2(\alpha)$). Therefore the correctness of the algorithm follows; namely, the algorithm outputs all the required coefficients.

By Lemma 3.4, we also know that the number of α's for which $E[f_\alpha^2] \geq \theta^2$ is bounded by $1/\theta^2$, for each length of α. Thus, the total number of recursive calls is bounded by $O(n/\theta^2)$.

Detailed analysis. We are still not done, since this algorithm assumes that we can compute $E[f_\alpha^2]$ exactly, something that is not achievable in polynomial time. On the other hand, we can approximate $E[f_\alpha^2]$ very accurately in polynomial time. Therefore we modify Subroutine `Coef`: instead of testing whether $E[f_\alpha^2] \geq \theta^2$ we approximate $E[f_\alpha^2]$ and test whether the approximated value is greater than $\theta^2/2$ (see Fig. 2).

```
SUBROUTINE Coef($\alpha$)
        $B_\alpha$ = Approx($\alpha$) /* $B_\alpha$ is approximating $E[f_\alpha^2]$.*/
        IF $B_\alpha \geq \theta^2/2$ THEN
                IF $|\alpha| = n$ THEN OUTPUT $\alpha$
                        ELSE Coef($\alpha 0$); Coef($\alpha 1$);
```

FIG. 2. *The modification of subroutine* `Coef`.

The approximation of $E[f_\alpha^2]$ is such that with very high probability the error in the approximation is small. That is, with high probability, every coefficient satisfying $|\hat{f}(z)| \geq \theta$ will be output, which guarantees the correctness condition of the algorithm. Also, this approximation guarantees that with high probability, no coefficient satisfying $|\hat{f}(z)| \leq \theta/2$ will be output, which bounds (by Lem. 3.4) the number of coefficients the algorithm outputs to at most $4/\theta^2$. Moreover, it implies that for every k at most $4/\theta^2$ strings $\alpha \in \{0,1\}^k$ of length k will pass the test of the subroutine, which bounds the number of calls to the recursive subroutine by $O(n/\theta^2)$. What we are left with is to bound the computation required to approximate $E[f_\alpha^2]$.

Let m_1, m_2 be parameters (to be fixed later). We approximate $E[f_\alpha^2(x)]$ as shown in Fig. 3.

The value of B_α is the approximation to $E[f_\alpha^2(x)]$. We now need to find the "right" values of m_1 and m_2, such that B_α will be a "good" approximation to $E[f_\alpha^2(x)]$. That is, with high probability, if $E[f_\alpha^2] \geq \theta^2$ then $B_\alpha \geq \theta^2/2$ and if $E[f_\alpha^2] \leq \theta^2/4$, then $B_\alpha < \theta^2/2$.

To prove that B_α is a "good" approximation of $E_x[f_\alpha^2(x)]$, we first prove that B_α would be a "good" approximation of $E_x[f_\alpha^2(x)]$ if we compute it with the real values of $f_\alpha(x_i)$. Then we show that the A_i's are "good" approximations for $f_\alpha(x_i)$. Finally, we show that even if we

```
SUBROUTINE Approx($\alpha$)
Choose at random $x_i \in \{0,1\}^{n-k}$, for $1 \le i \le m_1$.
For each $x_i$
        Choose at random $y_{i,j} \in \{0,1\}^k$, for $1 \le j \le m_2$.
        Let $A_i = \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_{i,j}x_i)\chi_\alpha(y_{i,j})$.
                                        /* $A_i$ is approximating */
                                        /* $f_\alpha(x_i) = E_y[f(yx_i)\chi_\alpha(y)]$.*/
Let $B_\alpha = \frac{1}{m_1}\sum_{j=1}^{m_1} A_i^2$.
                                        /* $B_\alpha$ is approximating $E_x[f_\alpha^2(x)]$.*/
RETURN $B_\alpha$.
```

FIG. 3. *Approximation of* $E[f_\alpha^2(x)]$.

compute B_α with the A_i's (instead of $f_\alpha(x_i)$) it is still a "good" approximation of $E_x[f_\alpha^2(x)]$. For the proof we use Chernoff bounds (see [HR89]):

LEMMA 3.5 (CHERNOFF). *Let $X_1, \ldots, X_m$ be independent, identically distributed random variables, such that $E[X_i] = p$ and $S_m = \sum_{i=1}^m X_i$.*

- *If $X_i \in [0,1]$, then*

$$Prob\left[(1-\varepsilon)\cdot p \le \frac{S_m}{m} \le (1+\varepsilon)\cdot p\right] \ge 1 - 2e^{-\varepsilon^2 mp/3}$$

- *If $X_i \in [-1,+1]$, then*

$$Prob\left[|\frac{S_m}{m} - p| \ge \lambda \cdot \frac{1}{\sqrt{m}}\right] \le 2e^{-\lambda^2/2}.$$

Using this bound, we claim that by choosing at random m_1 values x_i and computing the average $f_\alpha^2(x_i)$, we get a value that is very close to $E[f_\alpha^2]$.

LEMMA 3.6. *Let $B'_\alpha = \frac{1}{m_1}\sum_{j=1}^{m_1} f_\alpha^2(x_i)$, where $x_i \in \{0,1\}^{n-k}$, $1 \le i \le m_1$, are chosen uniformly at random. Then,*

$$Prob\left[\frac{3}{4}E[f_\alpha^2] \le B'_\alpha \le \frac{5}{4}E[f_\alpha^2]\right] \ge 1 - 2e^{-\frac{m_1}{48}\cdot E[f_\alpha^2]}.$$

Proof. Follows immediately from the first part of Lemma 3.5 with $\varepsilon = \frac{1}{4}$ (and $p = E[f_\alpha^2]$). □

The next lemma claims that A_i is a "good" approximation for $f_\alpha(x_i)$. It is based on the identity of Lemma 3.2 (i.e., $f_\alpha(x_i) = E_y[f(yx_i)\chi_\alpha(y)]$).

LEMMA 3.7. *For any value of x_i,*

$$Prob\left[|A_i - f_\alpha(x_i)| \ge \theta^2/16\right] \le 2e^{-\theta^4 m_2/2^9}.$$

Proof. The proof of Lemma 3.7 follows immediately from the second part of Lemma 3.5 with $\lambda = \frac{\theta^2\sqrt{m_2}}{16}$. □

Intuitively, if we approximate each $f_\alpha(x_i)$ well, the difference between B_α (which uses the approximate values) and B'_α (which uses the true values) should be small. The following lemma formalizes this intuition.

LEMMA 3.8. *If $|f_\alpha(x_i) - A_i| \le \frac{\theta^2}{16}$, for $1 \le i \le m_1$, then $|B_\alpha - B'_\alpha| \le \frac{2\theta^2}{16}$.*

Proof. By Corollary 3.3 it follows that $|f_\alpha(x_i)| \le 1$. By the definition of the A_i's it follows that $|A_i| \le 1$. Therefore,

$$|B_\alpha - B'_\alpha| = |\frac{1}{m_1}\sum_{i=1}^{m_1}(f_\alpha^2(x_i) - A_i^2)| \le \frac{1}{m_1}\sum_{i=1}^{m_1}|f_\alpha(x_i) - A_i| \cdot |f_\alpha(x_i) + A_i|$$
$$\le \frac{1}{m_1}\sum_{i=1}^{m_1}\frac{\theta^2}{16}\cdot 2 = \frac{2\theta^2}{16}. \qquad \square$$

Using the above lemmas, we now fix the values of m_1 and m_2 so that B_α will be a "good" approximation for $E[f_\alpha^2]$.

LEMMA 3.9. *Let* $m_1 = \Theta(\frac{1}{\theta^2}\log\frac{n}{\delta\theta^2})$ *and* $m_2 = \Theta(\frac{1}{\theta^4}\log\frac{n\cdot m_1}{\delta\theta^2})$. *With probability* $1-\delta$ *the procedure* `Coef` *outputs all the coefficients z such that* $|\hat{f}(z)| \ge \theta$, *and does not output any coefficient z such that* $|\hat{f}(z)| \le \theta/2$.

Proof. As shown above we have $O(n/\theta^2)$ calls to the subroutine `Coef` (and the same number of calls to the subroutine `Approx`). To guarantee a total error probability of δ we choose m_1 and m_2 so that the probability of error in each of the calls is no more than $\frac{\delta\theta^2}{n}$. Also recall that if $z = \alpha\beta$, for some $\alpha \in \{0,1\}^k$ and $\beta \in \{0,1\}^{n-k}$, and if $|\hat{f}(z)| \ge \theta$, then $E[f_\alpha^2] \ge \theta^2$.

Consider an α such that $E[f_\alpha^2] \ge \theta^2$. By Lemma 3.6, with probability at least $1 - 2e^{-m_1\theta^2/48}$, the value of $B'_\alpha \ge \frac{3}{4}\theta^2$. By Lemma 3.7, with probability $1 - 2m_1 e^{-\theta^4 m_2/2^9}$, all the values of A_i $(1 \le i \le m_1)$ satisfy $|f_\alpha(x_i) - A_i| \le \theta^2/16$. In this case, using Lemma 3.8, $B_\alpha \ge \frac{10}{16}\theta^2 > \frac{\theta^2}{2}$.

Consider an α such that $E[f_\alpha^2] \le \theta^2/4$. Note that B_α is monotonic in $E[f_\alpha^2]$, and therefore it is enough to consider the case $E[f_\alpha^2] = \theta^2/4$. By Lemma 3.6, with probability $1 - 2e^{-m_1\theta^2/192}$, the value of $B'_\alpha \le \frac{5}{16}\theta^2$. By Lemma 3.7, with probability $1 - 2m_1 e^{-\theta^4 m_2/2^9}$, all the values of A_i $(1 \le i \le m_1)$ satisfy $|f_\alpha(x_i) - A_i| \le \theta^2/16$. In this case, using Lemma 3.8, $B_\alpha \le \frac{7}{16}\theta^2 < \frac{\theta^2}{2}$.

So far we have shown that the algorithm performs the "right" recursive calls. This implies that, with probability $1-\delta$, the number of recursive calls is at most $4n/\theta^2$. It also implies that all the required coefficients will be output. Now we need to show that in such a case no coefficient z, such that $|\hat{f}(z)| \le \theta/2$ is output. Note that the probability of outputting such a coefficient is at least the probability that we made one "wrong" recursive call, and this probability is bounded by δ. $\square$

Once the procedure outputs the list of vectors, $z_1, \ldots, z_\ell$, we can approximate each coefficient $\hat{f}(z_i)$. Let the approximate value be γ_i. (Since by definition, $\hat{f}(z_i) = E[f\chi_{z_i}]$, then Lemma 3.5 guarantees that a "small" sample will give with a high probability a "good" approximation for all these coefficients.) The prediction hypothesis is $h(x) = \sum_{i=1}^{\ell}\gamma_i\chi_{z_i}(x)$. To conclude, the algorithm has the following performance:

THEOREM 3.10. *There is a randomized algorithm, that for any boolean function f, any* $\delta > 0$, *and any* $\theta > 0$ *outputs a list of vectors* $\alpha_i \in \{0,1\}^n$ *such that*

- *with probability* $\ge 1-\delta$ *the list contains every vector* α *for which* $|\hat{f}(\alpha)| \ge \theta$ *and does not contain any vector* α *for which* $|\hat{f}(\alpha)| \le \theta/2$. (*This implies that the list may contain at most* $4/\theta^2$ *vectors.*)
- *the algorithm runs in time polynomial in n,* $1/\theta$, *and* $\log 1/\delta$.

To summarize, in this section we have shown that if f can be ε-approximated by a t-sparse function, then it is sufficient to find all its coefficients larger than $\varepsilon/t = \theta$. Therefore we have established the following theorem.

THEOREM 3.11. *Let f be a boolean function such that there exists a t-sparse function g that* ε*-approximates f* (*in norm* L_2). *Then there exists a randomized algorithm that on* *input f and $\delta > 0$ outputs a function h such that with probability $1-\delta$ the function h $O(\varepsilon)$-approximates, in norm L_2, the input function f. The algorithm runs in time polynomial in n, t, $1/\varepsilon$, and $\log 1/\delta$.*

4. Functions with small L_1-norm. In this section we show that a function whose sum of the absolute value of the coefficients is "small" has a "small" number of "large" coefficients that "almost" determine the function. Therefore, in order to get a good approximation of the function, it is sufficient to approximate those coefficients. Saying it differently, we show that functions with "small" L_1-norm can be approximated by sparse functions.

Let f be a boolean function, and recall that $L_1(f) \overset{\triangle}{=} \sum_z |\hat{f}(z)|$. The following lemma shows that it is sufficient to approximate a small number of the (large) coefficients of f.

LEMMA 4.1. *Let $\varepsilon > 0$. Let $S = \{z \ : \ |\hat{f}(z)| \geq \varepsilon/L_1(f)\}$, and let $g(x) = \sum_{z\in S} \hat{f}(z)\chi_z(x)$. Then*

$$E[(f-g)^2] \leq \varepsilon.$$

Proof. By the definition of g, we have

$$(f-g)(x) = \sum_z (\hat{f}(z) - \hat{g}(z))\chi_z(x) = \sum_{z\notin S} \hat{f}(z)\chi_z(x).$$

Therefore, using Parseval's identity, we have

$$E[(f-g)^2] = \sum_{z\notin S} \hat{f}^2(z).$$

This is clearly bounded above by

$$\left(\max_{z\notin S} |\hat{f}(z)|\right)\cdot\left(\sum_{z\in\{0,1\}^n} |\hat{f}(z)|\right) \leq \frac{\varepsilon}{L_1(f)}\cdot L_1(f) = \varepsilon. \qquad \Box$$

This implies that if we can find all the coefficients that are greater, in absolute value, than $\varepsilon/L_1(f)$, we can approximate f. The procedure in the previous section gives a way to find all such coefficients in time poly(n, $L_1(f)$, $1/\varepsilon$, $\log 1/\delta$). Note that in order to use subroutine `Coef` we need to know the value of $L_1(f)$. If this is not the case, we can search for an upper bound on it. This will add a multiplicative factor of $O(\log L_1(f))$ to the time complexity. We have established the following theorem.

THEOREM 4.2. *There is a randomized algorithm, that for any boolean function f, and $\varepsilon, \delta > 0$, outputs a function g such that* $\text{Prob}[E[(f-g)^2] \leq \varepsilon] \geq 1-\delta$, *and the algorithm runs in time polynomial in n, $L_1(f)$, $1/\varepsilon$, and* $\log 1/\delta$.

4.1. Derandomization. For functions with a "small" L_1-norm we can efficiently derandomize the algorithm. One drawback of the derandomization is that it requires that we have a bound on the L_1-norm, since we cannot test hypotheses using randomization as before. The main idea in the derandomization is the usage of λ-bias distributions. The notion of a λ-bias distribution was first suggested by [NN90], and other constructions were given later by [AGHP90]. One way to formalize the notion of λ-bias is the following.

DEFINITION 4.1. Every distribution μ over $\{0,1\}^n$ can be considered as a real function $\mu(x) = \sum_z \hat{\mu}(z)\chi_z(x)$. A distribution $\mu(x)$ is λ-bias if for any $z \neq \vec{0}$, $|\hat{\mu}(z)| \leq \lambda 2^{-n}$.

Note that the uniform distribution $u(x) = \frac{1}{2^n}$ has $\hat{u}(z) = 0$, for $z \neq \vec{0}$, and therefore it is 0-bias. Also for any distribution μ,

$$\hat{\mu}(\vec{0}) = <\mu, \chi_0> = E[\mu] = \frac{1}{2^n}\sum_x \mu(x) = \frac{1}{2^n}.$$

One of the applications of λ-bias distributions is to derandomize algorithms. The derandomization of an algorithm is done by showing that the output of the algorithm when its coin tosses are chosen from a uniform distribution, and the output of the algorithm when its coin tosses are chosen from a λ-bias distribution are very similar. If this holds, then the deterministic algorithm is the following: (1) enumerate all the strings in the λ-bias distribution, (2) for each such string compute the value of the randomized algorithm, and (3) output the average of the replies in step (2). To have an *efficient* derandomization we would like that the sample space of the λ-bias probability distribution would be enumerable "efficiently" (in particular, it has to be "small").

THEOREM 4.3 ([NN90, AGHP90]). *There are λ-bias distributions whose sample spaces are of size $(\frac{n}{\lambda})^2$ and are constructible in polynomial time.*

Using the definition (and basic properties) of the Fourier transform we show the following identity.

LEMMA 4.4. *For any function f and any distribution μ,*

$$E_\mu[f] = \hat{f}(\vec{0}) + 2^n \sum_{z \neq \vec{0}} \hat{\mu}(z)\hat{f}(z).$$

Proof. By the definitions,

$$E_\mu[f] = \sum_x \mu(x) f(x) = \sum_x \left(\sum_z \hat{\mu}(z)\chi_z(x) \sum_{z'} \hat{f}(z')\chi_{z'}(x) \right).$$

Clearly, if $z = z'$ then $\sum_x \chi_z(x)\chi_{z'}(x) = 2^n$, and if $z \neq z'$ then $\sum_x \chi_z(x)\chi_{z'}(x) = \sum_x \chi_{z \oplus z'}(x) = 0$. Therefore the above sum equals

$$2^n \sum_z \hat{\mu}(z)\hat{f}(z).$$

As $\hat{\mu}(0) = \frac{1}{2^n}$, the lemma follows. □

Our goal now is to show that the algorithm behaves "similarly" when its coin tosses are chosen from the uniform distribution, u, or from a λ-bias distribution, μ. We show it by proving that the A_i's and the B_α computed by Subroutine `Approx` are "similar." The main tool for this is the following lemma.

LEMMA 4.5. *Let f be any function, u be the uniform distribution, and μ be a λ-bias distribution, then*

$$|E_\mu[f] - E_u[f]| \leq \lambda L_1(f).$$

Proof. By definition, $E_u[f] = \hat{f}(\vec{0})$. From Lemma 4.4 we have

$$E_\mu[f] = \hat{f}(\vec{0}) + 2^n \sum_{z \neq \vec{0}} \hat{\mu}(z)\hat{f}(z).$$

The definition of λ-bias distributions ensures that $|\hat{\mu}(z)| \leq \lambda/2^n$, therefore we get

$$|E_\mu[f] - E_u[f]| \leq |\hat{f}(\vec{0}) + 2^n \sum_{z \neq \vec{0}} \hat{\mu}(z)\hat{f}(z) - \hat{f}(\vec{0})| \leq 2^n \sum_{z \neq \vec{0}} \frac{\lambda}{2^n} |\hat{f}(z)| = \lambda L_1(f),$$

which completes the proof. □

LEMMA 4.6. *Let $h(y) = f(yx)$, for some fixed $x \in \{0, 1\}^k$. Then, $L_1(h) \leq L_1(f)$.*

Proof. The proof is by induction k; let $k = 1$. Then $h(y) = f(yb)$, where $b \in \{0, 1\}$. One can verify easily that if $b = 0$, then $\hat{h}(z) = \hat{f}(z0) + \hat{f}(z1)$, and if $b = 1$, then $\hat{h}(z) = \hat{f}(z0) - \hat{f}(z1)$. In both cases $L_1(h) \leq L_1(f)$.

The induction step follows from the fact that we can restrict the function bit after bit. □

This implies that we can compute the A_i's (the inner loop of subroutine Approx) with λ-bias distributions.

LEMMA 4.7. *Let u be the uniform distribution, and μ be a λ-bias distribution on $\{0, 1\}^k$. For any function f, $k \leq n$, $\alpha \in \{0, 1\}^k$, and $x \in \{0, 1\}^{n-k}$,*

$$|E_{y\in\mu}[f(yx)\chi_\alpha(y)] - E_{y\in u}[f(yx)\chi_\alpha(y)]| \leq \lambda L_1(f).$$

Proof. Let $h_x(y) \triangleq f(yx)$ and $g_x(y) \triangleq h_x(y)\chi_\alpha(y)$. By Lemma 4.6 $L_1(h_x) \leq L_1(f)$. First, we show that $L_1(g_x) = L_1(h_x)$ by showing that they have the same set of coefficients:

$$\hat{g_x}(z) =< g_x, \chi_z >=< h_x\chi_\alpha, \chi_z >=< h_x, \chi_\alpha\chi_z >=< h_x, \chi_{\alpha\oplus z} >= \hat{h_x}(\alpha \oplus z).$$

This implies that $L_1(g_x) \leq L_1(f)$. By Lemma 4.5,

$$|E_\mu[g_x] - E_u[g_x]| \leq \lambda L_1(g_x) \leq \lambda L_1(f). \qquad \square$$

We now show a few basic relations about the L_1-norm of the coefficients of a function, so that we can show that $L_1(f_\alpha^2) \leq L_1^2(f)$.

CLAIM 4.8. *For any function f and $\alpha \in \{0, 1\}^k$, $k \leq n$, then $L_1(f_\alpha) \leq L_1(f)$.*

This is because f_α, by definition, includes only a subset of the coefficients of f. The second claim establishes a relation between the L_1-norm of two functions and the L_1-norm of their product.

CLAIM 4.9. *For any functions g and h, $L_1(gh) \leq L_1(g)L_1(h)$.*

Proof. Note that

$$g(x)h(x) = \sum_{z_1,z_2\in\{0,1\}^n} \hat{g}(z_1)\hat{h}(z_2)\chi_{z_1}(x)\chi_{z_2}(x) = \sum_{z_3}\left(\sum_{z_1}\hat{g}(z_1)\hat{h}(z_3 \oplus z_1)\right)\chi_{z_3}(x).$$

(To see the last transformation, take z_3 to be $z_1 \oplus z_2$.) Therefore,

$$L_1(gh) = \sum_{z_3}|\sum_{z_1}\hat{g}(z_1)\hat{h}(z_3 \oplus z_1)| \leq \sum_{z_1,z_2}|\hat{g}(z_1)||\hat{h}(z_2)| = L_1(g)L_1(h). \qquad \square$$

We use the above two claims to bound $L_1(f_\alpha^2)$.

CLAIM 4.10. *For any function f and $\alpha \in \{0, 1\}^k$, $k \leq n$, then,*

$$L_1(f_\alpha^2) \leq L_1^2(f_\alpha) \leq L_1^2(f).$$

This implies that we can compute B_α (the outer loop of subroutine Approx) using only λ-bias distributions.

LEMMA 4.11. *For any function f, and $\alpha \in \{0, 1\}^k$, $k \leq n$,*

$$|E_\mu[f_\alpha^2] - E_u[f_\alpha^2]| \leq \lambda L_1^2(f).$$

Proof. Combine Lemma 4.5 with Claim 4.10. □

Lemma 4.11 can be used to derandomize the outer loop by choosing $\lambda = \varepsilon/L_1^2(f)$. Lemma 4.7 can be used to derandomize the inner loop by choosing $\lambda = \varepsilon/L_1(f)$. This implies that we have established the following theorem.

THEOREM 4.12. *There is a deterministic algorithm that receives as an input a boolean function f, $L_1(f)$, and $\varepsilon > 0$, and outputs a function g such that $E[(f - g)^2] \leq \varepsilon$, and the algorithm runs in time polynomial in n, $L_1(f)$, and $1/\varepsilon$.*

5. Decision trees. We consider decision trees, whose input is n boolean variables, and the branching in each node is based on a linear combination (over $GF(2)$) of a subset of the variables (as described in §2.2). In this section we show that, for any function corresponding to such a boolean decision tree, the sum of the absolute values of its coefficients (i.e., the L_1-norm) is bounded by m, the number of nodes in the tree. This implies, using the result in the previous section, that such decision trees can be approximated in polynomial time.

LEMMA 5.1. *Let f be computed by a decision tree with m nodes, then $L_1(f) \leq m$.*

Proof. A nonredundant decision tree is a tree in which for every leaf there is some input that ends up in that leaf. By the claim of the lemma, f has a decision tree with m nodes, therefore there is a nonredundant decision tree T_f with at most m nodes that computes f. Denote by $\texttt{leaf}(T_f)$ the set of leaves in the tree T_f, and by $d(v)$ the depth of node v. That is, $d(v)$ is the number of nodes on the path from the root to node v, not including v.

We claim that every node at depth d has exactly a 2^{-d} fraction of the inputs reaching it. The inputs that reach a node at depth d pass through d internal nodes; therefore, they satisfy a set of d linear constraints over $GF(2)$. Each such linear constraint is satisfied by exactly $\frac{1}{2}$ of the inputs. Since T_f is nonredundant, the linear constraints are linearly independent. Therefore, the fraction of the inputs that satisfy all the d constraints is 2^{-d}.

By the definition of a decision tree each input reaches a unique leaf. Let $I(v)$ be the set of all the inputs that reach leaf v. Then for every z,

$$\hat{f}(z) =< f, \chi_z >= E[f\chi_z] \quad = \sum_{v \in \texttt{leaf}(T_f)} 2^{-d(v)} val(v) E_{x\in I(v)}[\chi_z(x)].$$

In the following we show that $|E_{x\in I(v)}[\chi_z(x)]| = 1$ for exactly $2^{d(v)}$ values of z, and zero for the rest. This implies that each leaf can contribute at most one to the value of $L_1(f)$.

Consider a leaf v. Any input $x \in I(v)$ satisfies $d(v)$ linear constraints, i.e.,

$$\begin{aligned} x \odot y_1 &= b_1 \\ x \odot y_2 &= b_2 \\ &\vdots \\ x \odot y_{d(v)} &= b_{d(v)}, \end{aligned}$$

where $\odot$ denotes the inner product of two n-bit vectors $x, y \in \{0,1\}^n$, i.e., $x \odot y = \sum_i x_i y_i \bmod 2$.

The argument has two parts, depending on whether or not z is a linear combination of the y_i's. If z is a linear combination of the y_i's, then clearly the value of $x \odot z$ is fixed, for every $x \in I(v)$. Since the value of $x \odot z$ is fixed, by definition, the value of $\chi_z(x)$ is fixed to either $+1$ or -1, hence $|E_{x\in I(v)}[\chi_z(x)]| = 1$. Since the tree is nonredundant, there are exactly $2^{d(v)}$ vectors that are a linear combination of the y_i's. (Note that we consider $z = \vec{0}$ as a linear combination of the y_i's.) On the other hand, if $z \neq \vec{0}$ is not a linear combination of the y_i's then the number of $x \in I(v)$ satisfying $x \odot z = 0$ is the same as the number of $x \in I(v)$ satisfying $x \odot z = 1$. Therefore, in this case $E_{x\in I(v)}[\chi_z(x)] = 0$. Combining the two claims, we have that $\sum_z |E_{x\in I(v)}[\chi_z]| = 2^{d(v)}$.

Intuitively, each leaf v contributes to at most $2^{d(v)}$ coefficients, and to each coefficient it contributes $2^{-d(v)}$. This implies that leaf v contributes at most one to the sum of the absolute value of all the coefficients. Therefore, $L_1(f)$ is bounded by the number of leaves, which is at most m. The following calculations shows this formally:

$$\begin{aligned}
L_1(f) = \sum_{z\in\{0,1\}^n} \left|\hat{f}(z)\right| &= \sum_{z\in\{0,1\}^n} |E[f\chi_z]| \\
&= \sum_{z\in\{0,1\}^n} \left| \sum_{v\in \mathtt{leaf}(T_f)} 2^{-d(v)} val(v) E_{x\in I(v)}[\chi_z] \right| \\
&\le \sum_{z\in\{0,1\}^n} \sum_{v\in \mathtt{leaf}(T_f)} 2^{-d(v)} \left|E_{x\in I(v)}[\chi_z]\right| \\
&= \sum_{v\in \mathtt{leaf}(T_f)} 2^{-d(v)} \sum_{z\in\{0,1\}^n} \left|E_{x\in I(v)}[\chi_z]\right| \\
&= \sum_{v\in \mathtt{leaf}(T_f)} 2^{-d(v)} 2^{d(v)} \\
&= |\mathtt{leaf}(T_f)| \le m. \qquad \square
\end{aligned}$$

Combining Lemma 5.1 and Claim 2.1 with Theorem 4.12, we get the following theorem:

THEOREM 5.2. *There is a polynomial time (deterministic) algorithm, that for any boolean function f that can be represented by an m node decision tree with linear operations, and for any $\varepsilon > 0$, outputs a function g such that*

$$\mathrm{Prob}[f \neq \mathrm{sign}(g)] \le \varepsilon,$$

and the algorithm runs in time polynomial in n, m, and $1/\varepsilon$.

We now show that the bound given in Lemma 5.1 is tight. Consider the inner product function on inputs of $n = 2\ell$ variables:

$$f(x_1, \ldots, x_{2\ell}) = (-1)^{x_1 x_{\ell+1} \oplus \cdots \oplus x_\ell x_{2\ell}} = \prod_{i=1}^{\ell} (-1)^{x_i x_{\ell+i}}.$$

Let $h_i(x_1, \ldots, x_{2\ell}) = (-1)^{x_i x_{\ell+i}}$, for $1 \le i \le \ell$. Clearly, $f = \prod_{i=1}^{\ell} h_i$. The Fourier transform of h_i is

$$h_i(x) = \frac{1}{2} + \frac{1}{2}\chi_{I(i)}(x) + \frac{1}{2}\chi_{I(i+\ell)}(x) - \frac{1}{2}\chi_{I(i,i+\ell)}(x),$$

where $I(S)$ is the indicator vector of the set S, e.g., $I(\{j\})$ is equal to the vector who has the jth coordinate one and all the other coordinates zero. From the expansion of h_i it is clear that $|\hat{f}(z)| = 2^{-\ell}$, for any z, and therefore $L_1(f) = 2^{2\ell} 2^{-\ell} = 2^{\ell}$. We will show that there is a decision tree with linear operations of size $O(2^\ell)$ that computes f.

The following is a description of the decision tree that computes f. The first ℓ levels of the decision tree form a complete binary tree. In each node of level i $(1 \le i \le \ell)$ we test $x_i \oplus x_{\ell+i}$. For every leaf v of the tree, let $b_1^v, \ldots, b_\ell^v$ be the sequence of the replies to the queries $x_i \oplus x_{\ell+i}$ along the path from the root of the tree to v. Let, $S_v = \{i | b_i^v = 0\}$. We now test for the parity of all x_i's with $i \in S_v$. Let the value of the computation be the value of the parity. The tree has only depth $\ell + 2$, and hence only $O(2^\ell)$ nodes. The reason that it computes the inner product correctly is the following. If $b_i^v = 1$, then exactly one of $x_i, x_{\ell+i}$ is 0 and in particular $x_i x_{\ell+i} = 0$. This implies that the ith term in the inner product is zero, and therefore we can ignore it. If $b_i^v = 0$, then either both $x_i, x_{\ell+i}$ are 0, or both $x_i, x_{\ell+i}$ are 1. In both cases, $x_i x_{\ell+i} = x_i$. Therefore, instead of considering the value of the ith term (i.e., $x_i x_{\ell+i}$), we can consider the variable x_i. Therefore the parity of S_v is the parity of all the relevant terms.

Exact reconstruction. We show that a boolean decision trees with linear operations can be recovered exactly in time polynomial in n and 2^d, where d is the depth of the tree.

It follows from the proof of Lemma 5.1 that all the coefficients of a tree of depth d can be written as $k/2^d$, where k is an *integer* in the range $[-2^d, +2^d]$. The idea is to first find a good approximation of *all* the nonzero coefficients and then, using the above fact, to compute them *exactly*.

By Theorem 4.12, we have a *deterministic* algorithm that for every function f and $\varepsilon > 0$ outputs a function g such that

$$\sum_z (\hat{f}(z) - \hat{g}(z))^2 = E[(f-g)^2] \le \varepsilon$$

in time polynomial in n, $L_1(f)$, and $1/\varepsilon$. In particular, it follows that $|\hat{f}(z) - \hat{g}(z)| \le \sqrt{\varepsilon}$, for every z. We use this algorithm, with $\varepsilon < (\frac{1}{2}\frac{1}{2^d})^2$, which ensures that g satisfies $|\hat{f}(z) - \hat{g}(z)| < \frac{1}{2}\frac{1}{2^d}$, for every z. Since the real coefficient is of the form $k/2^d$, where k is *integer*, the difference between possible values that a coefficient can have is $1/2^d$; since the error is smaller than $\frac{1}{2}\frac{1}{2^d}$, by rounding we find the exact coefficient.

This implies that we recovered all the Fourier coefficients of the function *exactly*. Therefore, we found a function whose Fourier transform is *identical* to the tree's Fourier transform, this implies that the two functions are identical. By the choice of ε and as $L_1(f) \le m \le 2^{d+1}$, the running time of the algorithm is polynomial in n and 2^d. Thus, we have established the following theorem,

THEOREM 5.3. *There is a (deterministic) polynomial time algorithm, that for any boolean function f that can be represented by a depth d decision tree with linear operations, outputs a function g such that*

$$\forall x \in Z_2^n \qquad g(x) = f(x),$$

and the algorithm runs in time polynomial in n and 2^d.

An interesting special case is when the depth of the tree is logarithmic in n. In such a case, the algorithm will run in *polynomial* time.

6. Extensions and open problems.

The characterization of the decision trees can be extended easily to boolean functions of the form $f : \{0, 1, \dots, k-1\}^n \to \{0, 1\}$ that can be computed by a polynomial-size k-ary decision tree, namely, a tree in which each inner node v has k outgoing edges. When the computation reaches the node v, labeled by $S_v \in \{0, 1, \dots, k-1\}^n$, it assigns this node the value $\sum_{i=1}^n S_i \cdot x_i \bmod k$, and the computation continues to the appropriate child of v. For extending the results to such functions and decision trees we have to define the appropriate characters and modify the proofs accordingly. For each $z \in \{0, 1, \dots, k-1\}^n$, define the basis function χ_z:

$$\chi_z(x_1, \cdots, x_n) \overset{\Delta}{=} w^{z_1 \cdot x_1 + \dots + z_n \cdot x_n},$$

where $w = e^{\frac{2\pi i}{k}}$ is the root of unity of order k. In this case, a straightforward extension of our proof for $k = 2$ shows that the sum of the magnitudes of the coefficients is bounded by the number of leaves.

Another issue is decision trees with real outputs, where the leaves have real values from the interval $[0, M]$, i.e., $f : \{0, 1\}^n \to [0, M]$. In a similar way to the boolean case, one can show that any function f that has a real decision tree with m leaves then $L_1(f) \leq mM$. In this case the running time of the learning algorithm would be polynomial in M.

An open problem related to this work is to find other classes of functions that can be learned in polynomial time. In particular, it is very interesting whether functions that can be represented by a polynomial-size DNF formula can be learned in polynomial time. One possible direction to resolve this open problem is to show that for any polynomial-size DNF there is a polynomially sparse function that approximates it in L_2. So far we have not found any counter examples to this claim.

While our algorithm can be derandomized in the case of functions with polynomial L_1-norm, it is an open problem to derandomize it in the more general case of functions that can be approximated by polynomially sparse functions.

REFERENCES

[ABFR91] J. Aspnes, R. Beigel, M. Furst, and S. Rudich, *The expressive power of voting polynomials*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 1991, pp. 402–409.

[AGHP90] N. Alon, O. Goldreich, J. Hastad, and R. Peralta, *Simple constructions of almost k-wisw independent random variables*, in 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, October 1990, pp. 544–553.

[Ajt83] M. Ajtai, $\sum_1^1$*-formulae on finite structure*, Ann. Pure and Appl. Logic, 24 (1983), pp. 1–48.

[AM91] W. Aiello and M. Mihail, *Learning the fourier spectrum of probabilistic lists and trees*, in Proceedings SODA 91, ACM, January 1991.

[Ang87] D. Angluin, *Learning regular sets from queries and counterexamples*, Information and Computation, 75 (November 1987), pp. 87–106.

[Bel92] M. Bellare, *A technique for upper bounding the spectral norm with applications to learning*, in 5th Annual Workshop on Computational Learning Theory, July 1992, pp. 62–70.

[BHO90] Y. Brandman, J. Hennessy, and A. Orlitsky, *A spectral lower bound technique for the size of decision trees and two level circuits*, IEEE Trans. on Computers, 39(2) (1990), pp. 282–287.

[Bru90] J. Bruck, *Harmonic analysis of polynomial threshold functions*, SIAM J. Disc. Math., 3(2) (May 1990), pp. 168–177.

[BS90] J. Bruck and R. Smolensky, *Polynomial threshold functions, AC^0 functions and spectral norms*, in 31st Annual Symposium on Foundations of Computer Science, St. Louis, October 1990, pp. 632–641.

[Dia88] P. Diaconis, *The use of Group representation in probability and statistics*, in Lecture Notes Monograph Series, Vol. II, 1988.

[EH89] A. Ehrenfeucht and D. Haussler, *Learning decision trees from random examples*, Inform. and Comput., 82(3) (September 1989), pp. 231–246.

[FJS91] M. L. Furst, J. C. Jackson, and S. W. Smith, *Improved learning of AC^0 functions*, in 4th Annual Workshop on Computational Learning Theory, August 1991, pp. 317–325.

[FSS84] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[GL89] O. Goldreich and L. Levin, *A hard-core predicate for all one-way functions*, in Proc. 21st ACM Symposium on Theory of Computing, ACM, 1989, pp. 25–32.

[Han90] T. Hancock, *Identifying μ-decision trees with queries*, in 3rd COLT, August 1990, pp. 23–37.

[Han91] ———, *Learning 2μ* DNF *and $k\mu$ decision trees*, in 4th COLT, August 1991, pp. 199–209.

[Has86] J. Hastad, *Computational Limitations for Small Depth Circuits*, MIT Press, Massachusetts Institute of Technology, Boston, 1986, Ph. D. thesis.

[HR89] T. Hagerup and C. Rub, *A guided tour to chernoff bounds*, Info. Proc. Lett., 33 (1989), pp. 305–308.

[LMN89] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, fourier transform and learnability*, in 30th Annual Symposium on Foundations of Computer Science, Reserach Triangle Park, NC, October 1989, pp. 574–579.

[Man92] Y. Mansour, *An $o(n^{\log\log n})$ learning algorihm for* DNF *under the uniform distribution*, in 5th Annual Workshop on Computational Learning Theory, July 1992, pp. 53–61.

[NN90] J. Naor and M. Naor, *Small bias probability spaces: efficient construction and applications*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, May 1990, pp. 213–223.

[RB91] M. R. Roth and G. M. Benedek, *Interpolation and approximation of sparse multivariate polynomials over* GF(2), SIAM J. Comput., 20(2) (April 1991), pp. 291–314.

[SB91] K. Sui and J. Bruck, *On the power of threshold circuits with small weights*, SIAM J. Disc. Math., 4(3) (1991), pp. 423–435.

[Val84] L. G. Valiant, *A theory of the learnable*, Communications of the ACM, 27(11) (November 1984), pp. 1134–1142.

[Yao85] A. C. Yao, *Separating the polynomial-time hierarchy by oracles*, in 26th Annual Symposium on Foundations of Computer Science, Portland, OR, October 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 22, No. 6, p. 1349, December 1993

CORRECTION PARALLEL MERGE SORT*

RICHARD COLE†

This note corrects an error in the CRCW PRAM sorting algorithm described in [Col88]. The error was discovered by Saxena, Bhatt, and Prasad [SBP93]. This algorithm relies on a fast r-way merge, r a parameter, $2 < r \leq n$; in turn, this uses a fast CRCW algorithm for adding a set of r numbers of $O(\log r)$-bits (the latter algorithm runs in $O(\log r/\log\log r)$ time while performing $O(r)$ operations). Unfortunately, the merge, as described, actually seeks to add sets of r $\Theta(\log n)$-bit numbers.

Below, we describe an alternative way of organizing the additions needed for the merge; this alternative method performs additions and multiplications of pairs of numbers, and r-way additions of $O(\log r)$-bit numbers. Thus the bounds claimed in [Col88] do indeed hold.

This correction uses the terminology of the original paper. The following additional information is stored: each item e in $UP(u)$ stores its rank in each of the arrays $OLDSUP(v)$, where v is a child of u. (This information is available, for e determined its rank in $UP(u)$ by computing the sum of its ranks in the arrays $OLDSUP(v)$.)

By means of a proof similar to the one used for Lemma 7, one can show that any k intervals in $SUP(v)$ contain at least $kr - 1$ items in $NEWSUP(v)$. It follows that if item e has rank h in $OLDSUP(v)$, hr is a good estimate of its rank in $SUP(v)$. In fact, this rank is at least $hr - 1$ and at most $(h+1)r + 1$ (for e lies between the hth and the $(h+1)$th items in $OLDSUP(v)$).

We turn to the algorithm. The algorithm is unchanged except as follows. Following the creation of merging subproblems in the last paragraph on page 782 [Col88], each item e in $UP(u)$ knows its rank h' in $SUP(v)$. Let $corr = h' - hr$. Then e can determine its rank in $NEWUP(u)$ by summing the $corr$ terms for each child v of u and adding this sum to the product of r and its current rank. But this only requires the addition of r numbers of $O(\log r)$-bits each, plus some constant time arithmetic operations.

Now, e's rank can be used as an offset for the associated merging subproblem. Each item in $SUP(v)$ involved in the merge has rank in $SUP(v)$ larger than that of e, but larger by at most $r + 1$; call this the *offset rank*. In performing the merge, the offset ranks are treated as ranks. Then the merge will require only the addition of r numbers of $O(\log r)$-bits each. To obtain the actual rank of an item in $NEWUP(u)$, simply add e's rank in $NEWUP(u)$ to the rank determined in the previous sentence.

The above two paragraphs show that the unduly expensive additions of r numbers used in [Col88] can be modified to only require additions of r numbers of $O(\log r)$-bits each, plus constant time operations. Thus the bounds claimed in [Col88] do indeed hold.

REFERENCES

[Col88] R. COLE, *Parallel merge sort*, SIAM J. Comput., 4 (1988), pp. 770–785.

[SBP93] S. SAXENA, P. C. P. BHATT, AND V. C. PRASAD, *Time optimal parallel prefix algorithm*, Unpublished manuscript.

*Received by the editors April 2, 1993; accepted for publication April 12, 1993. The work was supported in part by National Science Foundation grants CCR-8902221, CCR-8906949, and CCR-9202900.

†Courant Institute, New York University, New York, New York 10012.