SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics

ON STOCHASTIC SCHEDULING WITH IN-TREE PRECEDENCE CONSTRAINTS*

CHRISTOS H. PAPADIMITRIOU† AND JOHN N. TSITSIKLIS‡

Abstract. We consider the problem of optimal scheduling of a set of jobs obeying in-tree precedence constraints, when a number M of processors is available. It is assumed that the service times of different jobs are independent identically distributed random variables. Subject to a minor assumption on the service time distribution, we show that policies of the "Highest Level First" type are optimal asymptotically, as the number of jobs tends to infinity.

Key words. scheduling, stochastic, trees

AMS(MOS) subject classification. 90B35

1. Introduction. Scheduling jobs with equal execution times and in-tree precedence constraints to minimize makespan (latest finishing time) is a classical problem in scheduling [Co]. It is known that, for any number of processors, the most natural policy, namely the one that asigns to the next available processor a leaf that has the largest height (distance from the root), is optimal [Hu]. This policy is called *highest level first.*

An interesting twist on this theme is to allow the execution times of the tasks to be independent and identically distributed exponential random variables [CR]. The results here are less complete: It is known that the highest level first policy is optimal *only in the two-processor case* [CR]. In fact, for two processors this policy is optimal under a wide variety of criteria [BR]. Unfortunately, for three processors the highest level first policy is not optimal, and no tractable way to approach this problem is known. It was shown in [Pa] that a generalization of the two-processor problem in another direction (and-or precedences) is PSPACE-complete.

In this paper we show that for any number of processors the highest level first policy, although suboptimal, is not much worse than the optimum. In particular, we show that, for any in-tree, the cost associated with a highest level first policy is no larger than the optimal cost times a factor that goes to one as N increases to infinity. Moreover, this result is shown to be true for a fairly wide class of service time distributions, exponential distributions being a special case.

2. Problem definition. We are given a set of M processors and an in-tree G with N nodes. Each node represents a job that may be processed by any of the processors. We assume that the jobs have service times that are independent and identically distributed random variables, with known distributions. We assume that these random variables are positive, with probability one.

A *scheduling policy* is a rule that, at time $t=0$, assigns L of the processors to L leaves of the tree, where L is the minimum of M and the number of leaves of G. If at time t some processor terminates the processing of a job, we delete the corresponding

* Received by the editors June 24, 1985; accepted for publication (in revised form) April 24, 1986. This research was supported by a National Science Foundation grant, an IBM Faculty Development Award and U.S. Army Research Office contract DAAG-29-84-K-0005.

† Departments of Computer Science and Operations Research, Stanford University, Stanford, California 94305.

‡ Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

leaf. Then, if there still exists a leaf with no processor assigned to it, the scheduling policy reassigns the free processor to one such leaf. The choice of the leaf to which a processor is assigned may depend on all events that have occurred up to that time. For example, it may depend on the amount of time that the other processors have been already processing the jobs to which they have been assigned.

In the above description of scheduling policies we have implicitly imposed the following restrictions:

(a) No processor remains unassigned as long as there exists a leaf with no processor assigned to it;

(b) Once a processor is assigned to a leaf, it remains assigned to it until the processing of the corresponding job is completed. (Thus, preemptive policies are not allowed.)

The above restrictions are only introduced for definiteness and to keep the description of scheduling policies simple. However, our result does not change if these restrictions are altered.

For any policy π and initial in-tree G, we define $T_\pi(G)$ to be the time that the processing of the root of G is completed. We are interested in policies that minimize $T_\pi(G)$, maybe approximately.

A special (and very easily implementable) class of policies is the class of highest level first policies. With such policies, processors are always assigned to leaves of maximum distance from the root. (Notice that there exists more than one highest level first policy; a particular one could be chosen by following a lexicographic tie-breaking rule, for example.)

It is known that highest-level-first policies are not optimal for $M>2$ [CR], even if the service time distribution is exponential. However, we will show that they come within an arbitrarily small percentage of the optimal, as the size of the original tree grows to infinity.

We assume that there exists some $\eta>0$ such that $E[e^{\eta X}]<\infty$, where X is sampled from the service time distribution. Notice that this assumption allows both discrete and continuous service time distributions; exponential distributions are covered in particular. Under the above assumption, we have the following result:

THEOREM. *There exists some function* $\beta:\{1,2,\cdots\}\mapsto[0,\infty)$ *such that* $\lim_{N\to\infty}\beta(N)=0$ *and such that for any in-tree* G *and any highest level first policy* H *we have*

$$E[T_H(G)]\leqq\inf_\pi E[T_\pi(G)](1+\beta(N)),$$

where $N=|G|$ *and where the infimum is taken over all scheduling policies* π.

We have not been able to prove that a bound of this form is tight, and we are tempted to conjecture the stronger result that there exists a constant A such that, for any in-tree G,

$$E[T_H(G)]\leqq\inf_\pi E[T_\pi(G)]+A.$$

3. Proof of the theorem. Let us denote by $\mathcal{U}$ the set of functions $f:[0,\infty)\mapsto[0,\infty)$ with the property that $\lim_{a\to\infty}a^k f(a)=0$, $\forall k\geqq 0$. (Negative exponentials are prototype elements of this class.) We develop a few useful consequences of our assumption on the service time distribution.

We first notice that, by Jensen's inequality, $E[X]\leqq(1/\eta)\log E[e^{\eta X}]<\infty$. We will therefore assume, without any loss of generality, that the mean service time $E[X]$ equals 1. We also have

LEMMA 1. *For any* $\varepsilon>0$ *there exist some* $h_1^\varepsilon,\cdots,h_4^\varepsilon\in\mathcal{U}$ *such that*

(i) $P(|\sum_{i=1}^{N} X_i - N| \geqq \varepsilon D) \leqq h_1^\varepsilon(D) \quad \forall D \geqq N,$

(ii) $E[\sum_{i=1}^{N} X_i; \sum_{i=1}^{N} X_i \geqq N + \varepsilon D] \leqq h_2^\varepsilon(D) \quad \forall D \geqq N,$[1]

(iii) $P(\max_{1 \leqq i \leqq N} X_i \geqq \varepsilon N^{1/2}) \leqq h_3^\varepsilon(N) \quad \forall N,$

(iv) $E[\max_{1 \leqq i \leqq N} X_i] \leqq \varepsilon N + h_4^\varepsilon(N) \quad \forall N,$

where $X_1, \cdots, X_N$ are independent random variables sampled from the service time distribution.

Proof. (i) This a classical result, but for completeness we outline a proof. One first shows that, given any $\varepsilon > 0$, we may choose $\eta > 0$ small enough so that $E[\exp\{\eta(X_i - 1 - (\varepsilon/2))\}] \leqq 1$. (A proof may be found in [Ha], for example.) It then follows that $E[\exp\{\eta(\sum_{i=1}^{N} X_i - N(1+(\varepsilon/2))\}] \leqq 1$ and $P(\sum_{i=1}^{N} X_i \geqq N + \varepsilon D) \leqq P(\sum_{i=1}^{N} X_i \geqq N + (\varepsilon/2)N + (\varepsilon/2)D) \leqq \exp\{-D\eta(\varepsilon/2)\}$. The inequality for the lower bound on $\sum_{i=1}^{N} X_i$ is proved similarly.

(ii) We have that $E[\sum_{i=1}^{N} X_i; \sum_{i=1}^{N} X_i \geqq N + \varepsilon D] \leqq \sum_{k=1}^{\infty} (D + \varepsilon k D + 1) P(\sum_{i=1}^{N} X_i \geqq N + \varepsilon k D) \leqq \sum_{k=1}^{\infty} (2D + \varepsilon k D) h_1^\varepsilon(kD) = h_2^\varepsilon(D)$, where h_1^ε is the function of part (i). We now show that $h_2^\varepsilon \in \mathcal{U}$. For any $m > 0$, we can find some A such that $h_1^\varepsilon(kD) \leqq A/(kD)^{m+3}$, for all k, D, such that kD is large enough, because $h_1^\varepsilon \in \mathcal{U}$. Then, $D^m h_2^\varepsilon(D) \leqq A(1+\varepsilon) \sum_{k=1}^{\infty} (kD)^{-2} = A(2+\varepsilon)\pi^2/(6D^2)$, which goes to zero, as $D \to \infty$. which goes to zero, as $D \to \infty$.

(iii) Let $C = E[\exp\{\eta X_i\}]$. Then, $P(X_i \geqq a) \leqq C e^{-\eta a}$, for all a and

$$P(\max_{1 \leqq i \leqq N} X_i \geqq a) \leqq NC e^{-\eta a}.$$

The result follows by taking $a = \varepsilon N^{1/2}$ and $h_3^\varepsilon(N) = CN \exp\{-\varepsilon\eta N^{1/2}\}$, which belongs to $\mathcal{U}$.

(iv) This follows from part (iii) in the same way that part (ii) followed from part (i). □

To simplify notation, we define $f_1^\varepsilon = \max_{1 \leqq i \leqq 4} \{h_i^\varepsilon\}$, which clearly belongs to $\mathcal{U}$.

We define the *depth* of a leaf of an in-tree to be the number of nodes on the path from the leaf to the root of the in-tree. We then define the depth of an in-tree as the maximum of the depths of all its leaves. We start by establishing the performance of scheduling policies for "thin" trees.

LEMMA 2. *For any $\varepsilon > 0$, there exists some $f_2^\varepsilon \in \mathcal{U}$ with the following property: Let G be any in-tree with M leaves and let $R_1, \cdots, R_M$ be the service times of its leaves. Then, $P(T_\pi(G) \geqq D + \varepsilon D' + \max_{1 \leqq i \leqq M} R_i | R_1, \cdots, R_M) \leqq f_2^\varepsilon(D')$, for all $D' \geqq D$, where D is the depth of G and π is a scheduling policy using M processors. (In fact there exists only a single scheduling policy in this case.) Also, $E[T_\pi(G) | R_1, \cdots, R_M] \leqq \max_{1 \leqq i \leqq M} R_i + (1+\varepsilon)D + f_2^\varepsilon(D)$.*

Proof. If $T_\pi(G) > D + \varepsilon D' + \max_{1 \leqq i \leqq M} R_i$, then there exists a path from some leaf to the root such that the sum of the service times of the jobs on this path (the leaf being excluded) is larger than or equal to $D + \varepsilon D'$. By Lemma 1(i), the probability of this event is bounded by $f_1^\varepsilon(D')$, for any particular path. There is a total of M candidate paths, which proves the first part of the lemma with $f_2^\varepsilon(D') = M f_1^\varepsilon(D')$. The second part of the lemma follows by appealing to Lemma 1(ii) to bound the contribution of the "unlikely" events to $E[T_\pi(G)]$. □

The main idea of the rest of the proof is the following: if the deepest part of an in-tree has very few leaves, then Lemma 2 will imply that a highest level first policy will be reducing its depth at approximately the largest possible rate of one unit per unit time, with high probability. If on the other hand the deepest part of an in-tree is

[1] If X is a random variable with distribution $P(\cdot)$ and A is an event, we use $E[X; A]$ to denote $\int_A X dP(X)$.

"thick," we will show that no policy can gain a substantial advantage over a highest level first policy. In order to put these two arguments together, we need to define a sequence of time intervals during which the deepest part of the remaining in-tree is either "thin" or "thick."

We will be assuming throughout that M is fixed and we introduce some new notation. Let us fix a highest level first policy H and another policy π. Given an initial in-tree G, we let $G_H(t)$, $G_\pi(t)$ be the (random) in-trees remaining at time t, if policy H, π, respectively, is followed. Let $D_H(t)$, $D_\pi(t)$ be the depth of $G_H(t)$, $G_\pi(t)$, respectively. Finally, let $d_H(t)$ be the smallest depth of any leaf to which a processor is assigned by policy H, at time t, in the graph $G_H(t)$. Notice that $G_H(t)$ has fewer than M leaves of depth larger than $d_H(t)$. (To resolve any ambiguity, we assume that all of the above introduced functions are right-continuous in t.) Also, for any processor k that is processing some job at time t, let $R_k(t)$ be the remaining time until the processing of that job is completed. Let $R^* = \max_{k,t} R_k(t)$. Notice that R^* equals the maximum processing time of any job. Thus, by virtue of Lemma 1 (iii)-(iv), we have $P(R^* \geqq \varepsilon N^{1/2}) \leqq f_1^\varepsilon(N)$ and $E[R^*] \leqq \varepsilon N + f_1^\varepsilon(N)$, for all $\varepsilon > 0$.

Given some $\varepsilon > 0$, we define a finite sequence $t_0 \leqq t_1 \leqq \cdots \leqq t_K$ of random times, (where K is a random integer to be defined below), and a corresponding sequence of random variables χ_k (for $k < K$), as follows. Let $t_0 = 0$ and suppose that t_k has been already defined. If $D_H(t_k) < N^{1/2}$ or if $G_H(t_k)$ has no more than M leaves, we let $K = k$ and the construction is completed. Otherwise, we consider two cases: (i) If $D_H(t_k) - d_H(t_k) \geqq N^{1/2}$, we let $\chi_k = 1$ and $t_{k+1} = t_k + (1+\varepsilon)N^{1/2}$. (ii) If $D_H(t_k) - d_H(t_k) < N^{1/2}$, we let $\chi_k = 0$ and we define t_{k+1} as the first time $t > t_k$ that $D_H(t) < N^{1/2}$, or the number of leaves of $G_H(t)$ is no larger than M or $D_H(t) - d_H(t) \geqq N^{1/2}$, whichever comes first. We define, for $k \leqq K$, $\Delta_k = D_H(t_k) - D_\pi(t_k)$. (Notice that the above introduced variables all depend on ε, even though this is not explicit in our notation.) We finally define t^* as the first time that $G_H(t)$ has M or less leaves.

It will be convenient to assume that both stochastic processes $G_H(t)$ and $G_\pi(t)$ are defined on a common probability space as follows. We start with a collection $\mathscr{X} = \{X_n^i : n = 1, 2, \cdots;\ i = 1, \cdots, M\}$ of independent random variables drawn from the service time distribution. We interpret X_n^i as the time spent by processor i to process the nth job ever assigned to that processor. With this definition, each of $G_H(t)$, $G_\pi(t)$ is a well-defined functional of the collection $\mathscr{X}$ of random variables. With this correspondence we have the following useful properties.

LEMMA 3. *If $G_H(s)$ has at least M leaves for all times $s < t$ (and in particular if $t \leqq t^*$) then*

(i) $|G_H(t)| \leqq |G_\pi(t)|$,

(ii) $D_\pi(t) \geqq d_H(t)$.

Proof. (i) Simply notice that each time that a node is deleted from G_π, a node is also deleted from G_H, provided that G_H had at least M leaves just before time t, which we are assuming.

(ii) Let A be the number of nodes of $G(0)$ at depth smaller than $D_H(t)$. If $D_\pi(t) < d_H(t)$, then $|G_\pi(t)| \leqq A$. On the other hand, under policy H, none of the jobs at depth smaller than $d_H(t)$ has been processed and there exists at least one unprocessed job at depth $d_H(t)$. Thus, $|G_H(t)| > A$ and the result follows from part (i). □

LEMMA 4. *There exists some $f_4 \in \mathscr{U}$ such that $P(T_H(G) > 2N) \leqq f_4(N)$.*

Proof. $P(T_H(G) > 2N) \leqq P(\sum_{i=1}^N X_i \geqq 2N) \leqq f_1^1(N)$, where each X_i is sampled independently from the service time distribution and where $f_1^1 \in \mathscr{U}$ is the function of Lemma 1. □

LEMMA 5. *For any $\varepsilon > 0$, there exists some $f_5^\varepsilon \in \mathscr{U}$ such that $P(K > 5N^{1/2}) \leqq f_5^\varepsilon(N)$.*

Proof. Notice that if $K > k+2$, then $t_{k+2} - t_k \geqq N^{1/2}$. Thus, if $K > 5N^{1/2}$, then $T_H(G) \geqq t_K \geqq 2N$, and the result follows from Lemma 4. □

LEMMA 6. *For any $\varepsilon > 0$, there exists some $f_6^\varepsilon \in \mathcal{U}$ such that $P(D_H(t_k) - D_H(t_{k+1}) \leqq N^{1/2}$ and $\chi_k = 1) \leqq f_6^\varepsilon(N)$ for all k.*

Proof. If $\chi_k = 1$, then the portion of $G_H(t_k)$ which lies at depth larger than $D_H(t_k) - N^{1/2}$ has at most M leaves. Therefore, for any time $t \geqq t_k$, a processor is always assigned to each leaf at depth larger than $D_H(t_k) - N^{1/2}$. Hence, by Lemma 2, this portion of $G_H(t_k)$ will be processed until time $t_k + (1 + (\varepsilon/2))N^{1/2} + R^*$, with probability no smaller than $1 - f_2^{\varepsilon/2}(N^{1/2})$. Furthermore, the probability that R^* is larger than $(\varepsilon/2)N^{1/2}$ is bounded by $f_1^{\varepsilon/2}(N)$. The result follows with $f_6^\varepsilon(N) = f_1^{\varepsilon/2}(N) + f_2^{\varepsilon/2}(N^{1/2})$, which belongs to $\mathcal{U}$. □

The following lemma is a straightforward consequence of Lemma 1(i) and the proof is omitted.

LEMMA 7. *For any $\varepsilon > 0$ there exists some $f_7^\varepsilon \in \mathcal{U}$ such that*

(i) $P(D_\pi(t_k) - D_\pi(t_{k+1}) > (1+2\varepsilon)N^{1/2}$ *and* $\chi_k = 1) \leqq f_7^\varepsilon(N) \quad \forall k$;

(ii) $P(D_H(t_k) - D_H(t_{k+1}) > 2N^{1/2}$ *and* $\chi_k = 1) \leqq f_7^\varepsilon(N) \quad \forall k$.

LEMMA 8. *For any $\varepsilon > 0$ there exists some $f_8^\varepsilon \in \mathcal{U}$ such that $P(k \leqq K$ and $\Delta_k \geqq N^{1/2} + 2\varepsilon k N^{1/2} + 1) \leqq k f_8^\varepsilon(N)$ for all k.*

Proof. The proof is by induction on k and with $f_8^\varepsilon = f_6^\varepsilon + f_7^\varepsilon$. The result is true for $k = 0$, since $\Delta_0 = 0$. Suppose it is true up to some k. In order to prove the result for $k+1$, we condition on the event $K > k$. If $\chi_k = 0$ and $k < K$, then, by definition, $D_H(t_{k+1} - \delta) - d_H(t_{k+1} - \delta) < N^{1/2}$, for sufficiently small $\delta > 0$. Moreover, $G_H(t_{k+1} - \delta)$ has more than M leaves. Thus, by Lemma 3(ii), $D_H(t_{k+1} - \delta) - D_\pi(t_{k+1} - \delta) < N^{1/2}$, for sufficiently small δ. At time t_{k+1} each processor may complete at most one job (because we have assumed that the service times are positive with probability one). Thus, $D_\pi(t_{k+1} - \delta) - D_\pi(t_{k+1}) \leqq 1$. Consequently, $D_H(t_{k+1}) - D_\pi(t_{k+1}) < N^{1/2} + 1$.

The above demonstrates that we only need to consider the case $\chi_k = 1$. Notice that $\Delta_{k+1} = \Delta_k + (D_\pi(t_k) - D_\pi(t_{k+1})) - (D_H(t_k) - D_H(t_{k+1}))$. We argue by contradiction. If $\Delta_{k+1} > 1 + N^{1/2} + 2\varepsilon(k+1)N^{1/2}$, then one of the following three must be happening:

(i) $\Delta_k > N^{1/2} + 2\varepsilon k N^{1/2} + 1$, which is the case with probability bounded by $k f_8^\varepsilon(N)$, using the induction hypothesis;

(ii) $D_\pi(t_k) - D_\pi(t_{k+1}) > N^{1/2} + 2\varepsilon N^{1/2}$, which is the case with probability bounded by $f_7^\varepsilon(N)$, because of Lemma 7(i);

(iii) $D_H(t_k) - D_H(t_{k+1}) < N^{1/2}$, which is the case with probability bounded by $f_6^\varepsilon(N)$, because of Lemma 6. Therefore, the induction step goes through with our choice of f_8^ε. □

LEMMA 9. *For any $\varepsilon > 0$ there exists some $f_9^\varepsilon \in \mathcal{U}$ such that $P(\Delta_K \geqq N^{1/2} + 10\varepsilon N + 1) \leqq f_9^\varepsilon(N)$.*

Proof. This is a trivial consequence of Lemmas 5 and 8. □

LEMMA 10. *For any $\varepsilon > 0$ there exists some $f_{10}^\varepsilon \in \mathcal{U}$ such that $P(D_H(t^*) - D_\pi(t^*) > 4N^{1/2} + 10\varepsilon N) \leqq f_{10}^\varepsilon(N)$.*

Proof. We distinguish three cases: (i) If $t^* = t_K$, then $D_H(t^*) - D_\pi(t^*) = \Delta_K$ and we use the bound of Lemma 9. (ii) If $t^* > t_K$, the definition of t_K implies that $D_H(t^*) - D_\pi(t^*) \leqq D_H(t^*) \leqq D_H(t_K) < N^{1/2}$ and there is nothing to prove. (iii) If $t^* < t_K$, then we have $t_{K-1} < t^* \leqq t_{K-1} + (1+\varepsilon)N^{1/2} = t_K$ and $\chi_{K-1} = 1$. Hence, $D_H(t^*) - D_\pi(t^*) \leqq D_H(t_{K-1}) - D_\pi(t_K) = \Delta_K + D_H(t_{K-1}) - D_H(t_K)$. For Δ_K we use the bound of Lemma 9. For $D_H(t_{K-1}) - D_H(t_K)$ we have $P(D_H(t_{K-1}) - D_H(t_K) > 2N^{1/2}$ and $\chi_{K-1} = 1) \leqq P(K > 5N^{1/2}) + \sum_{k \leqq 5N^{1/2}} P(D_H(t_{k-1}) - D_H(t_k) > 2N^{1/2}$ and $\chi_{k-1} = 1) \leqq$

$f_5^\varepsilon(N)+5N^{1/2}f_7^\varepsilon(N)$. By combining the above observations, we obtain the desired result. □

We can now complete the proof of the theorem. We have the obvious bound $E[T_\pi(G)] \geqq E[t^*]+E[D_\pi(t^*)]-1$. Moreover, using Lemma 2, $E[T_H(G)-t^* | G_H(t^*), R_1(t^*), \cdots, R_M(t^*)] \leqq R^*+D_H(t^*)+\varepsilon N+f_2^\varepsilon(N)$. We now use the fact $E[R^*] \leqq \varepsilon N+f_1^\varepsilon(N)$ and combine the above inequalities to obtain $E[T_H(G)-T_\pi(G)] \leqq E[D_H(t^*)-D_\pi(t^*)]+\varepsilon N+f_1^\varepsilon(N)+\varepsilon N+f_2^\varepsilon(N)+1$. We now use Lemma 10, together with the fact $D_H(t^*)-D_\pi(t^*) \leqq N$, to obtain the bound $E[D_H(t^*)-D_\pi(t^*)] \leqq 4N^{1/2}+10\varepsilon N+Nf_{10}^\varepsilon(N)$. Using the obvious fact $E[T_\pi(G)] \geqq N/M$, we finally obtain $E[T_H(G)-T_\pi(G)] \leqq E[T_\pi(G)](12M\varepsilon+\beta^\varepsilon(N))$, where $\beta^\varepsilon(N)$ satisfies $\lim_{N\to\infty}\beta^\varepsilon(N)=0$, for all $\varepsilon>0$. Recall now that ε was arbitrary and the result follows with $\beta(N)=\min_{\varepsilon>0}(11M\varepsilon+\beta^\varepsilon(N))$. □

Acknowledgments. We would like to thank the referees for a variety of suggestions on improving the style of the paper.

REFERENCES

[BR] J. BRUNO, *On scheduling tasks with exponential service times and in-tree precedence constraints*, Acta Inform., 22 (1985), pp. 139–148.

[Co] E. G. COFFMAN, JR., *Computer and Job Shop Scheduling Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Hu] T. C. HU, *Parallel sequencing and assembly-line problems*, Oper. Res., 9 (1961), pp. 841–848.

[Pa] C. H. PAPADIMITRIOU, *Games against nature*, Proc. 24th FOCS, 1983, pp. 446–450.

[CR] K. M. CHANDY AND P. F. REYNOLDS, *Scheduling partially-ordered tasks with probabilistic execution times*, Proc. of the Fifth Symposium on Operating Systems Principles, 1975, pp. 169–177.

[Ha] B. HAJEK, *Hitting-time and occupation-time bounds implied by drift analysis with applications*, Adv. in Appl. Probab., 14 (1982), pp. 502–525.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics

SUBLINEAR PARALLEL ALGORITHM FOR COMPUTING THE GREATEST COMMON DIVISOR OF TWO INTEGERS*

RAVINDRAN KANNAN†, GARY MILLER‡ AND LARRY RUDOLPH§

Abstract. The paper presents a sublinear time parallel algorithm for computing the greatest common divisor of two integers. Its running time on two n bit integers is $O(n \log\log n/\log n)$ using the weak concurrent read concurrent write model.

Key words. parallel algorithm, greatest common divisor

AMS(MOS) subject classification. 68Q99

Introduction. The advent of practical parallel processors has caused a re-examination of many existing algorithms with the hope of discovering a parallel implementation. One of the oldest and best known algorithms is Euclid's algorithm for computing the greatest common divisor (GCD). In this paper we present a parallel algorithm to compute the GCD of two integers.

Although there have been results in the parallel computation of the GCD of polynomials (Borodin, von zur Gathen and Hopcroft [2]), the integer case still appeared to be inherently serial; indeed Brent and Kung [3] achieve a running time of $O(n)$ with n processors arranged in a systolic array, where n is the number of bits required to represent the larger of the two input numbers. Although their method is an improvement on the best known serial integer GCD algorithm $O(n \log^2 n \log\log n)$ by Schönhage [10], it still requires n iterations; the parallelism only reduces the bit operations per iteration.

In this paper we present a sublinear time parallel algorithm to compute the integer GCD of two numbers on a weak CRCW-PRAM model of parallel computation allowing concurrent reads but only concurrent writes of the same value. (This model is often called the *common* model in the literature.) The time bound is $O(n \log\log n/\log n)$ assuming there are $n^2(\log n)^2$ processors working in parallel. This is computed assuming unit time for each elementary *bit* operation. Note that since we require concurrent reads and concurrent writes, our result cannot be used to construct a sublinear depth, polynomially sized boolean circuit for computing the GCD. This remains an interesting open problem.

This is the final journal version of Kannan, Miller and Rudolph [7] which contains essentially the entire contents of this paper. Since the appearance of the earlier version of this paper, Chor and Goldreich [4] have improved the running time getting rid of the $\log\log n$ term.

1. An overview of the algorithm. Throughout we assume that A and B are the two (nonnegative) integers whose GCD we want to find; A is the greater of the two and is an n bit integer. The basic idea of the algorithm is as follows: in the classical Euclidean algorithm we replace A with $A - qB$, where q is the quotient when A is divided by B. Here, instead, we consider in parallel $pA - q_pB$ for $p = 0$ to n, where q_p

* Received by the editors September 23, 1985; accepted for publication (in revised form) May 26, 1986.
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213. The work of this author was supported in part by National Science Foundation grant DCR-8416190.
‡ Department of Computer Science, University of Southern California, Los Angeles, California 90024.
§ Department of Computer Science, Hebrew University, Jerusalem, Israel.

is the quotient when pA is divided by B. All these integers are between 0 and B. By the pigeon-hole principle, there are at least two that agree on leading $\log n$ bits. Thus their difference is a nonnegative integer with at most $(n - \log n)$ bits. If we can replace A by their difference then we would reduce the problem size by $\log n$ bits during each two iterations thus requiring $O(n/\log n)$ iterations. There are two problems with this idea: (i) The GCD of A and B may be changed by the replacement and (ii) it is unclear how to execute the whole process in sublinear number of bit operations—a naive implementation would require $O(\log n)$ time for each iteration! We solve both these problems by a more complicated scheme.

The first problem is fixed (Lemma 1 stated below) as follows: we show that if $\mathrm{GCD}(A, B) = g$ and $\mathrm{GCD}(pA - qB, B) = h$, then g divides h and furthermore, h/g divides p. Since p is at most n, the only extra factors that are introduced into the GCD when we replace A, B by $pA - qB$, B are made up of powers of primes between 0 and n. At the outset of the algorithm, we remove all common prime factors of magnitude at most n between A and B (in $O(\log n)$ time) and then we run the entire algorithm. Now clearly, at the completion of the algorithm, the extra factors introduced in the GCD by the replacement can be removed quickly. Prime factors of magnitude at most n are referred to as "small" factors throughout.

The second problem involves more complications. We want to find p, q_p with p between $-n$ and n such that $pA - q_pB$ is an integer with at most $(n - \log n)$ bits. We thus need $pA - q_pB$ to satisfy

$$0 \leqq pA - q_pB < \min(B, 2^{(n - \log n)}).$$

We make an important observation: we need only to know the $O(\log n)$ most significant bits of A, B to find a p and a q_p that "nearly" satisfy these conditions. Moreover, p and q_p will be no larger than n in absolute value. Thus it appears that we need to deal only with $O(\log n)$ bit numbers at each iteration. But if we update at the end of each iteration, the update will cost $O(\log n)$ parallel time which is too expensive. To avoid this, the algorithm will proceed in $O(n/(\log n)^2)$ phases. Each phase will consist of $O(\log n)$ iterations. During each phase we deal only with the leading $O((\log n)^2)$ bits of A and B. We record in a unimodular 2×2 matrix T the transformations we have performed during the phase on the vector (AB). At the end of the phase the entries of the matrix T are $O((\log n)^2)$ bit numbers. We now perform the transformation given by T on the actual vector (AB) considering all the bits of the integers and then proceed to the next phase. The updates at the end of every phase cost $O(\log n)$ time (since we deal with the whole n bit integers), but since there are only $O(n/(\log n)^2)$ phases this costs only $O(n/\log n)$ time on the whole. Inside each phase, we deal only with $O((\log n)^2)$ bit numbers and thus arithmetic operations take $O(\log \log n)$ time. Thus we spend $O(\log \log n)$ time to chop off each $O(\log n)$ bits, giving the overall time bound of $O(n \log \log n/\log n)$.

2. A sublinear-time, parallel integer GCD algorithm. In this section we present the theoretical foundations as well as some of the algorithmic details of the GCD algorithm. We let A and B denote the two integers whose GCD we wish to compute. Throughout this paper, we ensure that $A \geqq B > 0$. At the outset we set n equal to the number of bits required to represent A, the larger of the two input numbers, and keep it fixed.

This section begins with three lemmas. The first characterizes how the GCD of A and B changes when A is replaced by the value $(pA - qB)$. The second lemma gives the details of our application of the pigeon-hole principle to reduce the number of

bits during an iteration. A third lemma is concerned with the situation in which A is much larger than B.

LEMMA 1. *If* $g = \mathrm{GCD}(A, B)$; $h = \mathrm{GCD}(pA - qB, B)$ *then* g *divides* h *and* (h/g) *divides* p.

Proof. Since g divides A and B, it clearly divides h. Let

$$h' = \mathrm{GCD}\left(\frac{pA}{g} - \frac{qB}{g}, \frac{B}{g}\right),$$

then

$$h' = \mathrm{GCD}\left(\frac{pA}{g}, \frac{B}{g}\right).$$

But A/g and B/g have no common factors. So $h' = \mathrm{GCD}(p, B/g)$. Hence $h' | p$ and further $h' = h/g$. □

LEMMA 2. *If* a, b *and* n *are positive integers and* $a \leqq bn$ *then there exist integers* p *and* q *not both zero such that* $|p| \leqq nb/a$, $|q| \leqq 2n$ *and* $0 \leqq pa - qb \leqq a/n$.

Proof. Consider all the integers p satisfying $0 \leqq p \leqq nb/a$. For each such p, there obviously exists a unique q such that $0 \leqq pa - qb < b$ with $|q| \leqq n$. (q is of course the quotient when pa is divided by b.)

There are $\lfloor bn/a \rfloor + 1$ distinct such pairs (p, q). In addition, take the pair $p = 0$, $q = -1$ which is distinct from the rest and satisfies $0 \leqq pa - qb \leqq b$. This makes a total of $\lfloor bn/a \rfloor + 2$ pairs p, q with $0 \leqq pa - qb \leqq b$. Thus, by the pigeon-hole principle, there are two values $p_1 a - q_1 b$ and $p_2 a - q_2 b$ such that

$$0 \leqq (p_1 a - q_1 b) - (p_2 a - q_2 b) \leqq \frac{b}{\lfloor bn/a \rfloor + 1} \leqq \frac{b}{bn/a} = \frac{a}{n},$$

where $p_1 \neq p_2$. The combination

$$(p_1 - p_2)a - (q_1 - q_2)b$$

satisfies $|p_1 - p_2| \leqq nb/a$ and $|q_1 - q_2| \leqq 2n$ and is the desired result. □

Let $\#X$ indicate the number of significant bits of X, not counting leading zeros.

LEMMA 3. *Suppose* $\#B = l$ *and* $\#A = k + l$ *where* k *is positive. Then the quotient when* A *is divided by* B *can be found approximately (with an error of at most* ± 3*) using only the leading* $\min(2k, k + l)$ *bits of* A *and the leading* $\min(k, l)$ *bits of* B.

Proof. The case when $l \leqq k$ is trivial because in this case the lemma allows us to consider all the bits of A and B. So assume that $l > k$.

Let

$$B = 2^{l-k} B' + B''$$

where $0 \leqq B'' < 2^{l-k}$ and B' comprises the leading k bits of B.

Also let

$$A = 2^{l-k} A' + A''$$

where $0 \leqq A'' < 2^{l-k}$ and A' comprises the leading $2k$ bits of A.

Suppose q' is the quotient and r' the remainder when B' divides A'. Then

$$A' = q'B' + r', \qquad 0 \leqq r' \leqq B' - 1 < 2^k - 1.$$

Then

$$(*) \qquad 2^{l-k} A' = q'(2^{l-k} B') + 2^{l-k} r',$$

$$A' < 2^{2k}, \qquad B' \geqq 2^{k-1}.$$

Thus $q' < 2^{2k-k+1} = 2^{k+1}$. From (*),

$$A = 2^{l-k}A' + A'' = q'(2^{l-k}B' + B'') + 2^{l-k}r' + A'' - q'B'',$$

i.e., $A = q'B + r$ where $-2^{l+1} < r < 2^{l-k}(2^k - 1) + 2^{l-k} = 2^l$.

Since $B \geqq 2^{l-1}$,

$$-4B < r < 2B.$$

Thus $(A - q'B)$ is between $-4B$ and $+2B$. Since we keep all numbers positive, we replace r by $r+4B$ if it is negative. Thus at the end $0 \leqq r \leqq 2B$, i.e., $\#r \leqq \#B + 1$. □

Note that we can find q' and r in the proof of the lemma in parallel time $O(\log^2 k)$ by long division since we have to deal only with $O(k)$ bit integers. In the algorithm, we will apply such a long division whenever $\#A - \#B = k$ exceeds $(\log n)^2 + 1$. Then we replace A, B by B, r in the proof of the lemma. Thus we have chopped off at least $(\log n)^2$ bits off A in $O(\log \log^2 n)$ time which preserves our ratio of time to number of bits chopped off to at most $1/(O(\log n))$.

The remainder of this section presents the algorithmic details. The main program, "IntegerGCD," is first presented and followed by the required procedures. We will use two low-level subroutines: one for normal division and the other for selecting out of a set elements one that has a particular property. Since parallel algorithms for both these problems are well known in the literature, we postpone their explanations to the end.

Given two integers, A, B each at most n bits long, this algorithm computes their GCD *using at most* $(n \log n)^2$ *processors executing in parallel. We make use of the notation*:

- $\#X$ indicates the number of significant bits of X, not counting leading zeros. If X is a matrix, then it indicates the number of bits in the largest entry of the matrix.
- $X[h:l]$ indicates the $h-l+1$ bits of X from the low order lth bit to the high order hth bit. That is, $\lfloor X/2^l \rfloor \bmod 2^{h+1}$. The least significant bit is the zeroth bit.

PROGRAM IntegerGCD(A, B):

The procedure takes as input two **positive** integers A, B, ensures that $A \geqq B$ at the start and maintains these conditions until it has found the GCD of A and B. Once the numbers get small, we run the ordinary (serial) Euclidean algorithm on them which takes time bounded by a polynomial in the number of bits. We assume that such an algorithm is available and it treats the case when the two integers are nonnegative—one or both of them could be zero.

```
1:   If A < B then swap them.
2:   n ← the number of bits of A /*n is never changed*/
3:   If #B ≦ 2(log n)^2 then begin
4:            Find C = A(mod B) by long division
                   *Now C and B are both 2(log n)^2 bit integers*
5:            Find by usual serial Euclidean algorithm the
                GCD of C and B and return this.
6:            Exit
7:            end
8:   Remove common small factors from A and B,
       and call the product of these factors SF
9:   Remove small factors from A and from B.
10:    repeat
11:      DoAPhase (A, B)
```

12: **until** $\#B < 2(\log n)^2$
13: Remove small factors from A and from B.
14: Find $C = A \pmod B$ by usual long division.
15: Run the serial Euclidean algorithm on C, B to get g'.
16: GCD $\leftarrow SF * g'$
END OF GCD

PROCEDURE DoAPhase (A, B)

Given A and B, each with at least $2(\log n)^2$ bits, with $A \geqq B > 0$ this procedure will return two new values for A and B, say A_{new} and B_{new}, with the properties that (i) the sum of the number of bits of A_{new} and B_{new} is at least $(\log n)^2$ less than the sum of those of A and B and that (ii) the GCD of A and B differs from the GCD of A_{new} and B_{new} by at most some small prime factors.

If the magnitudes of A and B differ by a lot then procedure LongDivide is called. Otherwise the procedure DoAnIteration is called at most $\log n$ times with the matrix T used to record the corresponding transformations to A and B.

1: $m \leftarrow \#A$; $s \leftarrow 2(\log n)^2$
2: **if** $\#A - \#B > (\log n)^2 + 1$
3: **then** LongDivide (A, B)
4: **else**
5: $a \leftarrow A[m : m - s + 1]$
6: $b \leftarrow B[m : m - s + 1]$
7: $T \leftarrow$ Identity Matrix
8: $endsize \leftarrow \#a + \#b - (\log n)^2$
9: **repeat**
10: DoAnIteration (a, b, T)
11: **until** $(\#a + \#b < endsize)$
12: $\begin{pmatrix} A \\ B \end{pmatrix} \leftarrow T \begin{pmatrix} A \\ B \end{pmatrix}$
13: Replace A, B by their absolute values.
14: **if** $A < B$ **then** swap their values.
/*Steps 12 through 14 are the only places A, B are changed*/
/*We obviously have $A \geqq B$ at the end of step 14*/
END PROCEDURE

PROCEDURE LongDivide (A, B)

Instead of performing a full division of A by B, we only divide the most significant $2k$ bits of A by the most significant k bits of B (see Lemma 3).

Assume $\#B + (\log n)^2 + 1 < \#A$.

0: $l \leftarrow \#B$
1: $k \leftarrow \#A - \#B$
2: $a \leftarrow$ most significant min $(2k, k + l)$ bits of A
3: $b \leftarrow$ most significant min (k, l) bits of B
4: $q \leftarrow \lfloor a/b \rfloor$ /*by long division*/
5: $C \leftarrow A - qB$
6: **if** $C < 0$ then $C \leftarrow C + 4B$
7: $A \leftarrow B$
8: $B \leftarrow C$
END PROCEDURE

PROCEDURE DoAnIteration (a, b, T)

Given a and b that are both $O((\log n)^2)$ bits and each at least $(\log n)^2$ bits, this procedure returns two new values for a and b whose sum of significant bits is at least $\log n$ less than it was upon entry. The transformations to a and b are recorded in the matrix T.

1: **if** $a/b \geqq n$
   /*NOTE: actual test is $a \geqq nb$*/
2: **then** Find a q such that $q = \lfloor a/b \rfloor$
3: $p \leftarrow 1$
4: **else** Find a pair (p, q), where $|p| \leqq nb/a$
   and $|q| \leqq 2n$, such that
   $0 \leqq pa - qb \leqq a/nm$ /*cf. Lemma 2*/
5: $T \leftarrow \begin{pmatrix} 0 & 1 \\ p & -q \end{pmatrix} T$
6: $\begin{pmatrix} a \\ b \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ p & -q \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$

END PROCEDURE

3. Analysis of the algorithm. The bulk of the algorithm is performed by DoAnIteration. In analyzing both the overall running time of the algorithm and its correctness we need to estimate how much smaller a and b are after each call of DoAnIteration as well as how much larger are the entries of T. Let $\#T$ equal the maximum number of bits in any entry of T. We next state some properties of the functioning of the procedure DoAnIteration.

LEMMA 4. *Let n be the number of bits of the larger of the two positive integers at the start of the algorithm. When each call of DoAnIteration is made, $a \geqq b \geqq 2^{(\log n)^2 - 1}$ and the call*

(i) *decreases the sum of their number of bits, i.e. $\#a + \#b$, by at least $\log n - O(1)$, and*

(ii) *increases the sum $\#T + \#a + \#b$ by at most $O(1)$.*

Proof. Note that if C and D are integers then $\#(CD) \leqq \#C + \#D$ and if they are 2×2 matrices then $\#(CD) \leqq \#C + \#D + 1$.

We divide the proof into two cases depending on whether line (2) or line (4) of DoAnIteration is implemented in this call. Suppose it is line (2) which is implemented, then the decrease in $\#a + \#b$ will be

$$(\#a + \#b) - (\#b + \#(a - qb)) \geqq \#a - \#b \geqq \log n - O(1)$$

which gives (i). Since T will be multiplied by a matrix whose largest entry in absolute value is q we can conclude that $\#T_{\text{new}} \leqq \#T + \#q + O(1)$. Thus, $\#T$ increases by at most $\#q + O(1)$ while $\#a + \#b$ decreases by at least $\#q + O(1)$. This gives (ii) in the case when line (2) is implemented.

Suppose, instead, that line (4) is implemented. By Lemma 2, line (4) will find a $pa - qb$ that satisfies (i) and where p and q are no more than $\log n$ bits. So, looking at step 5 of DoAnIteration, we see that $\#T$ will be increased by at most $\log n + 1$ bits and so (ii) is satisfied. □

We next consider procedure DoAPhase. Due to the if statement in line 2 of DoAPhase, when line (12) in procedure DoAPhase is implemented, we know that

1) it has called DoAnIteration at most $\log n + O(1)$ times,

2) the entries in T are small, i.e.

$$\#T \leqq (\log n)^2,$$

as argued in the proof of Lemma 4,

3) and the sum of the bits in a and b has decreased by at least $(\log n)^2$ bits.

LEMMA 5. *Each execution of the procedure DoAPhase with parameters A and B with* $A \geqq B > 0$ *and* $\#B \geqq 2(\log n)^2$ *will decrease* $\#A + \#B$ *by at least* $(\log n)^2$.

Proof. If $\#A - \#B > (\log n)^2 + 1$ then LongDivide will be called and we get the desired decrease in bits. We may assume that $A \geqq B$ and $\#A - \#B \leqq (\log n)^2$. Note that the latter condition is equivalent to $(\log n)^2 \leqq \#b$ (see line (6) of DoAPhase). Let $m = \#A$ and $s = 2(\log n)^2$. Let a' and b' be the low order $m - s$ bits of A and B that remain after removing a and b, i.e. $a' = A[m-s:0]$ and $b' = B[m-s:0]$. Thus, $A = a2^{m-s} + a'$ and $B = b2^{m-s} + b'$. Let T be the matrix on line (12) and

$$\begin{pmatrix} a_{\text{new}} \\ b_{\text{new}} \end{pmatrix} = T \begin{pmatrix} a \\ b \end{pmatrix}.$$

Thus,

$$T \begin{pmatrix} A \\ B \end{pmatrix} = (2^{m-s}) \begin{pmatrix} a_{\text{new}} \\ b_{\text{new}} \end{pmatrix} + T \begin{pmatrix} a' \\ b' \end{pmatrix}.$$

Since $\#T \leqq (\log n)^2$ and $\#a' = \#b' = m - s$, we have that

$$\#\left(T \begin{pmatrix} a' \\ b' \end{pmatrix} \right) \leqq m - s + (\log n)^2 + 1.$$

Thus, a' and b' only affect the lower half, $(\log n)^2 + 1$ bits, of the most significant $2(\log n)^2$ bits of A and B. In order to finish the proof we need to show that

$$(**) \qquad (\#a + \#b) - (m_a + m_b) \geqq (\log n)^2$$

where

$$m_a = \max(\#a_{\text{new}}, (\log n)^2),$$
$$m_b = \max(\#b_{\text{new}}, (\log n)^2).$$

We must use the maximum in the equation (**) since decreasing either $\#a_{\text{new}}$ or $\#b_{\text{new}}$ below $(\log n)^2$ will not in general decrease the number of bits in A_{new} or B_{new} because of the carry from the low order terms. Initially $\#a = 2(\log n)^2$ and $\#b$ may be smaller. But after the first call of DoAnIteration a will be replaced by an integer with $\leqq \#b$ bits. Thus $\#a_{\text{new}}, \#b_{\text{new}} \leqq \#b$. There are three cases to consider:

Case $a_{\text{new}} \leqq (\log n)^2$. Since $\#a = 2(\log n)^2$, we get:

$$(\#a + \#b) - (m_a + m_b) \geqq (\log n)^2 + (\#b - m_b)$$

using the fact that $\#b \geqq m_b$,

$$\geqq (\log n)^2.$$

Case $b_{\text{new}} \leqq (\log n)^2$. Similar to previous case.

Case $\#a_{\text{new}}$ *and* $\#b_{\text{new}} \geqq (\log n)^2$. Thus the maximums in equation (**) can be replaced with $\#a_{\text{new}}$ and $\#b_{\text{new}}$ reducing equation (**) to fact 3 (stated before Lemma 5).

4. Analysis of running time. We now show that the algorithm requires parallel time $O(n \log\log n / \log n)$. It is clear that the repeat loop of the main program is executed at most $n/(\log n)^2$ times, since each iteration removes at least $(\log n)^2$ bits from the sum of the bits in A and B (Lemma 5). We must show that each execution of the repeat loop takes no more than $O(\log n \log\log n)$ parallel time. This is done

by starting with the analysis of the innermost operations of DoAnIteration and then working our way out. The proofs of the following lemmas also indicate where the parallelism is needed.

We first review some of the known results for integer arithmetic operations performed in parallel. Integer addition of two n bit numbers can be performed in $O(\log n)$ time using n processors (Ladner and Fischer [8]). Integer multiplication also requires only $O(\log n)$ parallel time with $n \log\log n$ processors (Schönhage and Strassen [11]). Integer division too can be done in parallel time $O(\log n)$ with a polynomial number of processors, but the result is much harder (Beame, Cook and Hoover [1]). In fact, we do not need such fancy parellel division algorithms, we just require an $O((\log n)^2)$ algorithm using n^2 processors. This was earlier obtained by Cook [5].

At many points in the algorithm, we are required to choose a value or a pair of values from a set that satisfies a certain condition. It is easy to see that with enough processors, all the values in the set can be tested in parallel; however, in our weak model of parallel processing, simultaneous writes are allowed provided that all the processors write the same value. Thus if the set contains $O(n)$ items and many of the elements may satisfy some desired condition, $O(\log n)$ time may be required to choose any one such element. Fortunately, if there are n^2 processors then these selection processes can be done in the time it takes for a comparison as follows.

Briefly, the maximum of n can be chosen with n^2 processors by recording the results of all pairwise comparisons in a matrix: $M[i, j]$ is set to *true* if $a_i > a_j$ or if ($a_i = a_j$ and $i > j$). The row containing all *trues* corresponds to the maximum element. Since there will be only one such row, this row can be identified in constant time. In our GCD algorithm, we wish to select one of a set of elements in constant time. If a value satisfies our requirement, we tag it with a 1 otherwise with a 0. An element can then be chosen in constant time since the comparison will take only constant time (if $a_i = a_j$ then the processor "knows" if $i > j$). (Frieze and Rudolph [6] give a parallel algorithm that implements such a selection in constant time, with a high probability, with only n processors.)

LEMMA 6. *Each call to Procedure DoAnIteration can be executed in* $O(\log\log n)$ *parallel time using* $n^2(\log n)^2$ *processors.*

Proof. We assume that the two integers, a, b, contain at most $2(\log n)^2$ bits. Line (1) is accomplished by a multiplication of two numbers of no more than $2(\log n)^2$ bits, thus requiring no more than $O(\log\log n)$ time. Lines (2) or (4) can also be done in this time bound by assigning $(\log n)^2$ processors to each of the at most n^2 equations "$pa - qb$" in the case when line (4) is executed and to each of the n equations "$a - bq$" in the other case. The multiplication requires at most $O((\log n)^2)$ processors and can be computed in logarithmic time in the number of bits, i.e. $O(\log\log n)$ time. Only one (p, q) pair is needed; however, there may be many such pairs. For each p there will be only one q that satisfies the condition that $pa - qb < b$ and thus there are at most only $O(n)$ pairs to choose from. The selection can be done in constant time by the above outline with n^2 processors.

Lines (5) and (6) can be computed in $O(\log\log n)$ time since the entries in the matrices are no more than $O((\log n)^2)$ bits. □

LEMMA 7. *Procedure LongDivide can be executed in* $O(\log n)$ *time using no more than* $O(n)$ *processors.*

Proof. We show that each line can be executed within the required time bounds:

Line (1) can easily be computed using a binary fan-in tree and n processors in $O(\log n)$ time.

Lines (2) and (3) take constant time to identify the appropriate bits.

Line (4), the division of a $2k$ bit number by a k bit number can be done within $O((\log k)^2)$ time with k processors.

Line (5) is simply a multiplication of a k bit number by an n bit number and this takes no more than $O(\log n)$ parallel time with n processors. The subtraction is also done within this time bound. □

LEMMA 8. *An execution of the procedure DoAPhase requires* $O(\log n \log\log n)$ *parallel time, using* $n^2(\log n)^2$ *processors.*

Proof. This procedure may invoke the procedure DoAnIteration or the procedure LongDivide and thus depends on the previous two lemmas. In a manner similar to the explanation in the previous proofs, line (2) requires only $O(\log n)$ time using n processors.

By Lemma 3 it is clear that procedure DoAnIteration is not called more than $\log n$ times.

Executing line (10) is also within the time bound since A and B have at most n bits each and the entries of T each are at most $(\log n)^2$ and thus the multiplication takes at most $O(\log n)$ parallel time using n processors. □

THEOREM 1. *The* GCD *of two integers, each representable in at most n bits requires parallel time* $O(n \log\log n/\log n)$ *using* $n^2(\log n)^2$ *processors.*

Proof. The time bound follows from the previous lemmas provided we can remove the common small factors in $O(n/\log n)$ parallel time. All small primes, i.e. less than n, can be identified quickly as follows: For each $p \leqq n$ assign $n \log n$ processors to check that no number up to $\sqrt{p}$ divides p. After all the small primes have been identified, all their powers, p^i for $i \leqq n$, are computed in parallel. Finally, for each power for each prime, p_j^i, we check to see if p_j^i is a factor. For each j the maximum i is then chosen as described above.

Since we can identify all small prime factors of an n bit number in $O((\log n)^2)$ time, the theorem follows. □

5. Conclusion. To summarize, the two salient features of the algorithm are: the observation based on the pigeon-hole principle that we can easily find an integer combination of the two integers A and B which has fewer bits than n and the idea of working in phases so as to perform arithmetics on n bit integers only once every phase, the more frequent operations being performed on $O((\log n)^2)$ bit integers. It appears that yet another approach is needed if the GCD is to be computed in poly-log parallel time.

REFERENCES

[1] P. W. BEAME, S. A. COOK AND H. J. HOOVER, *Log depth for division and related problems*, 25th Annual Symposium on the Foundations of Computer Science, October 1984, pp. 1-6.

[2] A. BORODIN, J. VON ZUR GATHEN AND J. HOPCROFT, *Fast parallel matrix and* GCD *computations*, 23rd Annual Symposium on Foundations of Computer Science, November 1982, pp. 65-71.

[3] R. P. BRENT AND H. T. KUNG, *Systolic* VLSI *arrays for linear time* GCD *computation*, VLSI 83, International Federation of Information Processing, 1983.

[4] B. CHOR AND O. GOLDREICH, *An Improved Parallel Algorithm for Integer* GCD, MIT Laboratory for Computer Science, Cambridge, MA, April 1985, to appear.

[5] S. A. COOK, *The classification of problems which have fast parallel algorithms*, Lecture Notes in Computer Science, Vol. 158, Springer-Verlag, New York-Berlin-Heidelberg, 1983.

[6] A. FRIEZE AND L. RUDOLPH, *A parallel algorithm for all pairs shortest paths in a random graph*, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1984.

[7] R. KANNAN, G. MILLER AND L. RUDOLPH, *Sublinear parallel algorithm for computing the greatest common divisor of two integers*, 25th Annual Symposium on Foundation of Computer Science, October 1984, pp. 7-11. © IEEE.

[8] R. E. LADNER AND M. J. FISCHER, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980), pp. 831-838.

[9] J. REIF, *Logarithmic depth circuits for algebraic functions*, 24th Annual Symposium on Foundation of Computer Science, November 1983, pp. 138-145.

[10] A. SCHÖNHAGE, *Schnelle Berechnung von Kettenbruchententwicklungen*, Acta Inform., 1 (1971), pp. 139-144.

[11] A. SCHÖNHAGE AND V. STRASSEN, *Schnelle Multiplikation grosser Zahlen*, Computing, 7 (1971), pp. 281-292.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
003

COMPLEXITY OF VIEWS: TREE AND CYCLIC SCHEMAS*

ODED SHMUELI† AND ALON ITAI†

Abstract. In relational databases a *view definition* is a query against the database, and a *view materialization* is the result of applying the view definition to the current database. A view materialization over a database may change as relations in the database undergo modifications.

Several problems concerning views are considered, many of which are shown to be hard (NP-complete or even $\sum_2^p$-complete). Each problem was treated for general databases and for the much simpler *tree* databases (also called *acyclic* databases).

View related problems over fixed schemas, in which only the data is allowed to vary, were examined. Methods to handle this case were presented; their complexity is polynomial: for tree schemas the degree of the polynomial is independent of the schema structure while for cyclic schemas the degree depends on the schema structure. These methods may present a practical possibility for dynamic view maintenance.

Key words. database, view, acyclic schema, trees, complexity, maintenance, amortized cost

AMS(MOS) subject classifications. 68P15, 68Q15, 68Q25

1. Introduction. A relational database [8], [27] is a collection of tables called *relations*, each containing a set of data rows called *tuples*. We differentiate between the structure of the database (the *schema*) and the time varying data (the *state*). A database schema $\mathbf{D} = (\mathbf{R}_1, \cdots, \mathbf{R}_n)$ is simply a multiset of finite subsets of a set (of *attributes*) $\mathbf{U} = \bigcup_{i=1}^{n} \mathbf{R}_i$; a schema can be viewed as the edge-set of a hypergraph over $\mathbf{U}$ [2].

One may partition the class of database schemas into *tree* and *cyclic* schemas. A schema is a tree schema if there is a tree whose nodes correspond to the schema's sets, and for each A in $\mathbf{U}$, the subtree induced by nodes containing A is connected. Tree schemas are also called *acyclic schemas*.

The partition above appears to be a good dividing line for database problem analyses. Acyclicity has wide implications in query processing [3], [6], [15], [17], [19], [27] and in dependency theory and schema design [4], [5], [11], [12], [20], [23]. Mathematical properties of acyclicity have also been studied [11], [16], [18], [19], [22], [26].

It has been shown [3], [6], [15], [28] that a class of queries (called *tree queries*), which imply acyclic databases, appears easier to process than queries which imply cyclic databases (called *cyclic queries*); and that the crux of query processing is constructing a tree (actually an "embedded tree") [17], [19]. The above results all hinge on the simple structure of tree schemas. In this paper we examine the relationship between schema structure and view related problems.

A *view definition* is a query against the database. A *view materialization* is the result of applying the view definition to the current database state. A view materialization over a database may change as relations in the database undergo modifications. When views are materialized, they remain valid as long as the underlying database remains unchanged. Usually, views are not materialized until needed. In certain systems views are never materialized; instead queries against the view are *modified* to reflect the view definition (a process called *query modification*).

The main difference between an "ordinary" query and a view definition has to do with the frequency of use. A view is either a query that is often posed or one which

* Received by the editors January 31, 1984; accepted for publication (in revised form) May 19, 1986.
† Computer Science Department, Technion-Israel Institute of Technology, Haifa, Israel.

delimits a relevant portion of the database for a group of users. Hence, maintaining a correct view materialization over time may prove beneficial.

The view definitions we consider are all of a simple form: perform the natural join of all the relations in the database and project the result on a set of attributes $\mathbf{X}$. This simple form actually encodes a much larger class of views [6]. We examine various problems associated with these views and their materialization maintenance over time. We note that view related problems were mainly treated in the past under the guise of query processing [9], [28].

View maintenance includes a variety of problems concerning the tuples in the view, equivalence of views, how changes in the underlying database affect the view and which kind of information is useful in maintaining a view. For example, one of the problems we treat is the following: Given a database D, a view definition $\mathbf{X}$, a tuple t and a relation schema $\mathbf{R}_i$ would the view materialization change when t is added to R_i?

Terminology is presented in § 2; our problem classification scheme is introduced, and a summary of the results is tabulated. Section 3 is devoted to "join problems," i.e., we consider the case $\mathbf{X}=\mathbf{U}$ (the view is the natural join of all the database relations). "Genuine" views, where $\mathbf{X}$ is a proper subset of $\mathbf{U}$, are treated in § 4. In § 5 we consider view complexity over a fixed schema. A preliminary version of the material in this section appeared in [25].

2. Terminology.

2.1. Relational databases. A *universe* $\mathbf{U}$ is a finite set of *attributes*. A *relation schema* $\mathbf{R}_i$ is a subset of $\mathbf{U}$, and a *database schema* $\mathbf{D}$ (or simply *schema*) is a multi-set of relation schemas.[1] Clearly, a database schema may be viewed as the set of edges of a hypergraph over $\mathbf{U}$ [2]. Associated with each $A \in \mathbf{U}$ is a possibly infinite *domain*, $\mathrm{dom}(A)$. The *domain* of a relation schema $\mathbf{R}_i = \{A_{i1}, \cdots, A_{ih_i}\}$ is $\mathrm{dom}(\mathbf{R}_i) \stackrel{\text{def}}{=} \bigtimes_{k=1}^{h_i} \mathrm{dom}(A_{ik})$.

A *relation state* R_i for relation schema $\mathbf{R}_i$ is a finite subset of $\mathrm{dom}(\mathbf{R}_i)$; one can think about the state as a table of data with columns $A_1, \cdots, A_h$. A *database state* for schema $\mathbf{D}$ is an assignment of relation states to $\mathbf{D}$'s relation schemas. We use $\mathbf{D} = (\mathbf{R}_1, \cdots, \mathbf{R}_n)$ to denote a database schema and $D = (R_1, \cdots, R_n)$ for a corresponding state. Let $\cup(\mathbf{D}) = \cup_{i=1}^{n} \mathbf{R}_i$.

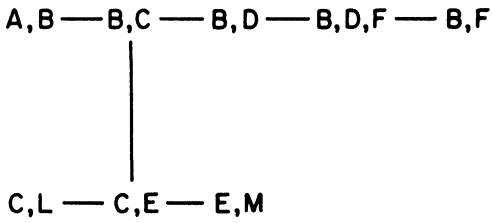
Elements in a relation state are called *tuples*. Tuple t over schema $\mathbf{R}$ *matches* tuple s over schema $\mathbf{S}$ if for all $A \in \mathbf{R} \cap \mathbf{S}$, the values of tuples t and s for attribute A are identical. The *projection* of relation state R over attribute set $\mathbf{X} \subseteq \mathbf{R}$, denoted $R[\mathbf{X}]$, is the maximal subset of $\mathrm{dom}(\mathbf{X})$ containing tuples that match some tuple in R. The (natural) *join* of relation states R and S, *denoted* $R \bowtie S$, is defined as the maximal subset in $\mathrm{dom}(\mathbf{R} \cup \mathbf{S})$ containing tuples that match a tuple in R and a tuple in S. A relation R is *total* in database D if it contains all its possible tuples composed of values appearing somewhere in the database, i.e. if $R = \bigtimes_{A \in \mathbf{R}} (\cup_{\mathbf{R} \in \mathbf{D}} R[A])$.

For a database D over schema $\mathbf{D}$ define $J(D) = \bowtie_{R \in D} R$; we use J instead of $J(D)$ when D is understood; define $J(R_i + t)$ to be the natural join of all the relations in D except that R_i is augmented with the tuple t. A *view definition* is simply a set $\mathbf{X} \subseteq \cup(\mathbf{D})$ of attributes; a *view materialization* V is $V = J[\mathbf{X}]$; also let $V(R_i + t) = J(R_i + t)[\mathbf{X}]$. Our class of views appears to be quite limited; however as is shown in [6] this class encodes a much larger class—those views defined by equijoin queries.

[1] All structures in this paper are finite.

2.2. Tree schemas. A *qual graph*[2] for **D** is an undirected graph whose nodes are in one-to-one correspondence with the relation schemas of **D**, such that for each attribute A, the subgraph induced by the nodes whose corresponding relation schemas contain A is connected [6]. **D** is a *tree schema* if some qual graph for it is a tree; otherwise **D** is a *cyclic schema.* See Fig. 2.1.

Consider the schema
$\mathbf{D} = (\{A, B\}, \{C, L\}, \{E, M\}, \{C, E\}, \{B, F\}, \{B, D, F\}, \{B, D\}, \{B, C\})$.
D is a tree schema viz.



For example, the subgraph induced by attribute C is:

The following is a cyclic schema:
$\mathbf{D} = (\{A, B\}, \{B, C\}, \{C, A\})$.
The only qual graph for **D** is

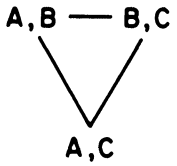


FIG. 2.1. *Tree and cyclic schemas.*

A database is a *tree database* (or an *acyclic database*) if the underlying database schema is acyclic; otherwise it is a *cyclic database.* The following simple procedure, discovered independently by [13] and [29], recognizes tree schemas. The procedure applies the following two steps until neither is applicable.

Step 1. Delete any attribute which appears in exactly one relation schema.

Step 2. Find two relation schemas **R** and **S** in **D** such that $\mathbf{R} \subseteq \mathbf{S}$; delete **R** from **D**.

It can be shown that the original schema is a tree schema iff upon termination of the above procedure the database schema consists of a single (empty) relation schema. (A linear time algorithm for recognizing tree schemas appears in [26].)

2.3. Complexity classes. A problem, or a language, L is in $(NP)P$ if given a string x, determining $x \in L$ can be done by a (non)deterministic Turing machine within time polynomial in the size of the input x. A problem is in Σ_2^p if it can be solved by a nondeterministic Turing machine, which may use an oracle for a set in NP, in time polynomial in the size of the input. (An oracle for a language L can be thought of as a "subroutine" which when given some string x answers in one time unit "yes" if $x \in L$ and "no" otherwise. The subtle point is that the Turing machine can make use of the fact that a string does not belong to the oracle set.) For more details, the reader is referred to [14].

[2] We use traditional graph theory notation.

A problem A is *polynomial-time complete* for the complexity class C, or *C-complete* for short, if for any other problem B in C, there is a polynomial time bounded Turing machine M which transforms a string x into a string $M(x)$, such that $x \in B$ iff $M(x) \in A$. Intuitively, if A is a complete problem in a class, then a polynomial algorithm for solving A will provide an efficient algorithm for *all* problems in the class.

Several known polynomial-time complete problems are:

(1) 3SAT. Given: a boolean formula in 3CNF, i.e. in conjunctive normal form having three literals per clause [1].
Question: Is F satisfiable? That is, is there an assignment to the boolean variables in F which makes it *true*?
3SAT is NP-complete.

(2) Let L be the language

$$L = \{F(X, Y) | \exists X \ \forall Y \ F(X, Y) \text{ is } true\}.$$

L is Σ_2^p-complete (see [14]).

2.4. Problem classification. In the following definition the size of a database is the size of the schema plus the size of the state. We shall classify problems according to the following criteria:

(1) The object in question:
J —A problem concerning the join of a given database D.
V —A problem concerning the view of a given database D on the given attributes **X**.

(2) The type of data supplied (optional).
C —Change: given a tuple t and an index i, consider the new join $J(R_i + t)$ (or new view $V(R_i + t)$).
G—Given: the input consists of the input to C *and* the old join (or old view materialization).

(3) The question:
E —Emptiness: is the join (or the view) *not* empty?
M—Membership: given a tuple t does t belong to the join (view)?
I —Intotality: is the join (view) *not* total?
N—Not equal: is the new join (view) not equal to the old one? (This question is meaningful only if C or G are present.)

For example: the problem JE is defined as follows:

Given a database D is $J \neq \emptyset$?

And the problem VGN is:

Given a database D, a view definition **X**, the view materialization V, an index i and the tuple t is $V \neq V(R_i + t)$?

Thus a total of 22 different problems are defined, results concerning these problems are shown in Table 2.1. (NA appears when the problem is not defined.) If **D** is a tree schema the above problems are sometimes easier as seen in Table 2.2.

3. Join problems.

3.1. Polynomial problems.

JM, JCM *and* JGM. Checking $t \in J$ amounts to checking that for all database relations R_i, $i = 1, \cdots, n$, $t[\mathbf{R}_i] \in R_i$.

TABLE 2.1

	E	M	I	N
J	NP-C[14], [21]	P	P	NA
JC	NP-C	P	P	NP-C
JG	NP-C	P	P	NP-C
V	NP-C[14], [21]	NP-C[14], [28]	Σ_2^p-C	NA
VC	NP-C	NP-C	Σ_2^p-C	Σ_2^p-C
VG	NP-C	NP-C	Σ_2^p-C	NP-C

TABLE 2.2

	E	M	I	N
J	P*	P	P	NA
JC	P*	P	P	P*
JG	P*	P	P	P*
V	P	P[28]	NP-C	NA
VC	P	P	NP-C	NP-C
VG	P	P	NP-C	P

* The results of [3], [6] imply that these problems are polynomial.

Remark. For tree databases all the problems are polynomial if the view definition **X** is contained in any one of the relations $\mathbf{R}_i$.

JI, JCI *and* JGI. If the join is total then each R_i must be total. Conversely, if for all i R_i is total then the join is total. Therefore, the join is total iff each R_i is total. The latter condition can be checked by counting the number of tuples in each R_i.

3.2. NP-complete problems.

JE. This problem was first shown to be NP-complete by Chandra and Merlin [7]. The problem is in NP since a nondeterministic Turing machine can guess a tuple s and check whether $s \in J$ in polynomial time. We show completeness by using the following construction.

Standard database construction. Given a boolean formula $F(X)$ in 3CNF, we show how to construct a database D such that $J \neq \emptyset$ iff F is satisfiable. With each clause of F we associate a relation schema, whose attributes are the variables appearing in the clause. W.l.o.g. we may assume that each relation schema thus constructed consists of three attributes. Each relation consists of the seven boolean assignments which make the original clause evaluate to *true.* It can be easily seen that each tuple in the natural join of all the relations in the database "spells out" an assignment to F's variables satisfying F. Hence, $J \neq \emptyset$ iff F is satisfiable. Observe that the size of the database constructed above is linear in the size of the formula.

JCE. The problem is in NP since it suffices to guess a tuple s and check whether it belongs to $J(R_i + t)$. The completeness follows by a reduction from JE: Let $D = (R_1, \cdots, R_k)$ be an instance of JE. Let $\mathbf{D}' = (\mathbf{R}_0', \mathbf{R}_1', \cdots, \mathbf{R}_k')$ where $\mathbf{R}_0'$ consists of a single attribute C, $R_0' = \{\langle b \rangle\}$ and the attributes of $\mathbf{R}_i'$ are those of $\mathbf{R}_i$ and the additional attribute C. Construct $\mathbf{D}'$ with

$$R_i' = \{\langle a_1, \cdots, a_{h_i}, a \rangle | \langle a_1, \cdots, a_{hi} \rangle \in R_i\}.$$

By construction $J(D')=\varnothing$. We augment R_0' by the tuple $t=\langle a\rangle$, clearly

$$J(R_0'+t)\neq\varnothing \text{ iff } J(D)\neq\varnothing.$$

JGE. In the previous construction the old join was empty; thus we may assume it was given.

JCN *and* JGN. The problem is in NP since a nondeterministic Turing machine can guess a tuple s such that $s\notin J$ and $s\in J(R_i+t)$. (Recall that *JM* is polynomial.) Completeness follows by a reduction from JE (as in the completeness proof of JCE—we add a new column C and a new relation $\mathbf{R}_0$ such that $J(D')=\varnothing$: however after adding a new tuple t $J(R_0+t)=\varnothing$ iff $J(D)=\varnothing$).

4. View problems.

4.1. The VI problem over general databases. Let $F(X)$ be a boolean formula, and t a global truth assignment, i.e., for each variable y_k, $t(y_k)=$ *true* or $t(y_k)=$ *false*. Define $t(F)=F(t(X))$ to be the value of F under the assignment t, where $t((x_1,\cdots,x_n))=(t(x_1),\cdots,t(x_n))$.

The structure of F defines a tree T_F, the root of which is the main connective of F and whose children are the subtrees associated with the arguments of the connective. Each leaf of the tree is associated with a variable $x_j\in X$. Let T_F have m nodes, numbered $1,\cdots,m$. A distinct variable z_i is associated with each node. Let $Z=\{z_1.\cdots,z_m\}$. Let F_i be the subformula associated with the node i.

We define a formula G_i associated with each node: For a leaf x_j, $G_i=z_i\equiv x_j$, expressed in CNF (conjunctive normal form)

$$G_i=(x_j+\bar{z}_i)\cdot(\bar{x}_j+z_i).$$

For a "not" node whose child is z_j, $G_i=z_i\equiv\bar{x}_j$, or in CNF

$$G_i=(x_j+z_i)\cdot(\bar{x}_j+\bar{z}_i).$$

For an "and" node whose children are z_L and z_R, $G_i=z_i\equiv z_L\cdot z_R$, or in CNF

$$G_i=(z_i+\bar{z}_L+\bar{z}_R)\cdot(\bar{z}_i+z_L)\cdot(\bar{z}_i+z_R).$$

Finally, for an "or" node whose children are z_L and z_R, $G_i=z_i\equiv z_L+z_R$, or in CNF

$$G_i=(\bar{z}_i+z_L+z_R)\cdot(z_i+\bar{z}_L)\cdot(z_i+\bar{z}_R).$$

Let $G_F(X,Z)=\Lambda_{i\in T_F}G_i$.

LEMMA 4.1. *Let t be a truth assignment; then $t(G_F(X,Z))$ is true iff for all i $t(z_i)=t(F_i(X))$.*

Proof. By induction on the number of nodes of T_F.

Basis. If T_F has size 1, then $F=x_j$. Therefore, $Z=\{z_i\}$ and $G_F=z_i\equiv x_j$. Because of the structure of G_F:

$$G_F(t(X,Z))=\textit{true} \quad \text{iff } t(z_i)=t(x_j).$$

Since $t(x_j)=t(F)$:

$$G_F(t(X),Z)=\textit{true} \quad \text{iff } t(z_i)=t(F).$$

Induction step. Suppose, for instance, that $F(X)=F_L+F_R$. Therefore, $G_F=(\Lambda_{i\in F_{T_L}}G_i\Lambda_{i\in F_{T_R}}G_i)\cdot(z_F\equiv(z_L+z_R))$. Since $G_L=\Lambda_{i\in F_{T_L}}G_i$ and $G_R=\Lambda_{i\in F_{T_R}}G_i$, $G_F=G_L\cdot G_R\cdot(z_F\equiv(z_L+z_R))$.

For $t(G_F)$ to be *true*, $t(G_L)=$ *true*, $t(G_R)=$ *true* and $t(z_F\equiv(z_L+z_R))=$ *true*. By induction, $t(z_k)=t(F_k)$ for all variables z_k in either the left or the right subtree. Obviously, $t(z_F)=$ *true* iff at least one of z_L or z_R has the value *true*. W.l.o.g., z_L is *true*

and by the induction hypothesis, $t(F_L) = \textit{true}$, which implies that $t(F) = \textit{true}$. Hence, $t(z_F) = t(F)$. The case $t(z_F) = \textit{false}$ is handled similarly.

Conversely, if $t(G_F)$ is *false* then $t(G_L) = \textit{false}$ or $t(G_R) = \textit{false}$ or $t(z_F \equiv z_L + z_R) = \textit{false}$. If one of the first two cases holds, then by the induction hypothesis there exists a z_i such that $t(z_i) \neq t(F_i)$. Otherwise, $t(z_F) \neq t(z_L + z_R)$ and since $t(G_L) = t(G_R) = \textit{true}$, $t(z_L) = t(F_L) = \textit{true}$ and $t(z_R) = t(F_R) = \textit{true}$. Consequently, $\textit{false} = t(z_F) \neq t(F) = \textit{true}$.

The other cases are similar. □

The above lemma can also be proved by using the techniques of Cook's theorem [1].

LEMMA 4.2. *For every Boolean formula $F(X)$ over the variables $X = \{x_1, \cdots, x_n\}$ there exists a* 3CNF *formula $H_F(X, Z)$ $(Z = \{z_1, \cdots, z_m\})$ such that*

(i) *For all assignments t to the variables X*

$$F(t(X)) = \textit{true} \quad \textit{iff } H_F(t(X, Z)) \textit{ is satisfiable};$$

(ii) *The size of H_F is linearly bounded by the size of F.*

Proof. Construct G_F as in Lemma 4.1. Using the above notation, let $H_F(X, Z) = G_F(X, Z) \cdot z_F$.

Let t satisfy H_F, i.e., $H_F(t(X), t(Z)) = \textit{true}$. Therefore, both $t(z_F) = \textit{true}$ and $t(G_F(X, Z)) = G_F(t(X), t(Z)) = \textit{true}$. By Lemma 4.1, if $G_F(t(X), t(Z)) = \textit{true}$, $t(z_F) = F(t(X))$. Since $t(z_F) = \textit{true}$, $F(t(X)) = \textit{true}$.

Conversely, suppose $F(t(X)) = \textit{true}$. Let t assign the value $F_i(t(X))$ to z_i. By Lemma 4.1 $G_F(t(X), t(Z)) = \textit{true}$, and since by the assignment $t(z_F) = F(t(X)) = \textit{true}$, $H_F(t(X), t(Z)) = \textit{true}$ and hence $H_F(t(X), Z)$ is satisfiable. □

THEOREM 4.1. VI *is Σ_2^p-complete.*

Proof. The fact that VI is in Σ_2^p is straightforward: a nondeterministic Turing machine M can guess v and then consult an oracle for $v \notin V$, the oracle set is in NP since determining whether $v \in V$ is NP-complete.

Let L be the language

$$L = \{F(X, Y) | \exists X \forall Y F(X, Y) \text{ is } \textit{true}\}.$$

L is complete in Σ_2^p (see [14]). Given a string of the form $F(X, Y)$ we show how to construct a database D and view definition X, such that $\exists X \forall Y F(X, Y)$ is *true* iff $(\exists v v \notin V)$.

By Lemma 4.2, given a boolean formula $F(X, Y)$ we construct, in linear time in the size of F, a boolean formula $H_{\neg F}(X, Y, Z)$ in 3CNF such that for all boolean vectors t_X, t_Y,

$$[\exists Z H_{\neg F}(t_X, t_Y, Z)] \leftrightarrow [\neg F(t_X, t_Y)] \text{ is } \textit{true},$$

which implies that for any boolean vector t_Z,

$$[H_{\neg F}(t_X, t_Y, t_Z)] \rightarrow \neg F(t_X, t_Y) \text{ is } \textit{true}.$$

Build a standard database D over $\mathbf{U} = X \cup Y \cup Z$ for $H_{\neg F}$ as described in § 3. Let X be the view definition on D.

CLAIM. $[\exists v v \notin V]$ *iff the formula* $[\exists X \forall Y F(X, Y)]$ *is true.*

Proof of claim. Assume $[\exists v v \notin V]$: Each tuple t over $\mathbf{U}$ "spells out" an assignment t_X, t_Y, t_Z to the boolean variables in $H_{\neg F}(X, Y, Z)$. Suppose $v \notin V$, then for all tuples t such that $t_X = v$ $H_{\neg F}(t_X, t_Y, t_Z) = \textit{false}$ (otherwise, if $H_{\neg F}(t_X, t_Y, t_Z) = \textit{true}$ then t would be in J and v in V). Let v spell out the assignment t_X to the X variables in $H_{\neg F}(X, Y, Z)$. It follows that

$$\forall Y \forall Z \quad [H_{\neg F}(t_X, Y, Z) = \textit{false}].$$

In other words

$$\forall Y \forall Z \quad \neg H_{\neg F}(t_X, Y, Z).$$

This gives

$$\forall Y \quad \neg[\exists Z H_{\neg F}(t_X, Y, Z)].$$

Consider any boolean vector t_Y. By the above we have

$$\neg[\exists Z H_{\neg F}(t_X, t_Y, Z)].$$

But, by Lemma 4.1

$$[(\exists Z) H_{\neg F}(t_X, t_Y, Z)] \leftrightarrow [\neg F(t_X, t_Y)].$$

We conclude that for any boolean vector t_Y

$$\neg\neg F(t_X, t_Y) \quad \text{or equivalently } F(t_Y, t_X),$$

which means that indeed $\exists X$ (namely t_X) such that for all Y, $F(X, Y)$.

We now assume $[\exists X \forall Y F(X, Y)]$:

Let t_X be a boolean vector such that $\forall Y F(t_X, Y)$. Choosing any t_Y, we have $F(t_X, t_Y)$. By Lemma 4.2,

$$\neg\exists Z \quad H_{\neg F}(t_X, t_Y, Z)$$

i.e.,

$$\forall Z \quad \neg H_{\neg F}(t_X, t_Y, Z).$$

Since t_Y was chosen arbitrarily it follows that

$$[\forall Y \forall Z \neg H_{\neg F}(t_X, Y, Z)] = \textit{true}. \tag{4.1}$$

We now show that (4.1) implies $\exists vv \notin V$. It suffices to show that

$$[\neg\exists vv \notin V] \rightarrow \neg[\forall Y \forall Z \neg H_{\neg F}(t_X, Y, Z)],$$

or, equivalently, that $[\forall vv \in V] \rightarrow \neg[\forall Y \forall Z \neg H_{\neg F}(t_X, Y, Z)]$. In other words, we have to show that

$$[\forall vv \in V] \rightarrow \exists Y \exists Z H_{\neg F}(t_X, Y, Z).$$

From $\forall vv \in V$ follows the existence of a boolean vector t_X in V. But if t_X is a tuple in the view, then there must exist a "parent" tuple $t = (t_X, t_Y, t_Z)$ such that $t_X = t[X]$. The existence of t implies the existence of t_Y and t_Z such that $H_{\neg F}(t_X, t_Y, t_Z) = \textit{true}$. But this simply states that

$$\exists Y \exists Z \quad H_{\neg F}(t_X, Y, Z) \quad \square$$

4.2. The VI problem over tree databases. The following problems are useful in proving the NP-completeness result:

SET COVER:

Given n sets $C_1, \cdots, C_n$ and an integer k, are there $C_{i_1}, \cdots, C_{i_k}$ whose union equals $\bigcup_{i=1}^{n} C_i$? (NP-complete [14])?

FAMILY COVER:

Given $k > 1$ families $S_1, \cdots, S_k$, where each S_i contains n sets $S_{i1}, \cdots, S_{in}$, are there $S_{1i_1}, \cdots, S_{ki_k}$ such that $\bigcup_{j=1}^{k} S_{ji_j} = \bigcup_{i=1}^{k} \bigcup_{j=1}^{n} S_{ij}$?

NCP (Non-Cartesian Product):

Given families $F_1, \cdots, F_h$, where $F_i = (C_{i1}, \cdots, C_{ik})$, $(k > 1)$, does the NCP inequality condition

$$\cup_{i=1}^{h}(C_{i1} \times \cdots \times C_{ik}) \subsetneq \left(\bigcup_{i=1}^{h} C_{i1}\right) \times \cdots \times \left(\bigcup_{i=1}^{h} C_{ik}\right)$$

hold?

CLAIM 1. FAMILY COVER *is* NP-*complete.*

Proof. Let $C_1, \cdots, C_n$ and $k > 1$ be an instance of SET COVER. Form an instance of FAMILY COVER with families $S_1, \cdots, S_k$, where each S_i contains a copy of $C_1, \cdots, C_n$.

CLAIM 2. NCP *is* NP-*complete.*

Proof. The problem is obviously in NP. Let $S_1, \cdots, S_k$ be an instance of FAMILY COVER, where $S_i = \{S_{i1}, \cdots, S_{in}\}$. W.l.o.g. each S_i has a unique element associated with it that is a member of each set in the S_i family, and is *not* a member of any set in any other family. Let $A = \cup_{i=1}^{k} \cup_{j=1}^{n} S_{ij}$. We form an instance of NCP by associating with each $a \in A$ a family of sets $F_a = \{F_{a1}, \cdots, F_{ak}\}$ where $F_{aj} = \{\langle j, p\rangle | a \notin S_{jp}$ and $1 \leqq p \leqq n\}$.

We claim that there is a FAMILY COVER iff the NCP inequality condition is satisfied.

First, observe that $(\cup_{a\in A} F_{a1}) \times \cdots \times (\cup_{a\in A} F_{ak}) = \times_{i=1}^{k}\{\langle i, 1\rangle, \cdots, \langle i, n\rangle\}$, because there is a unique element associated with each family S_i which appears in no other family S_j.

Suppose there is a family cover $S_{1i_1}, \cdots, S_{ki_k}$. We claim that

$$t = \langle\langle 1, i_1\rangle, \cdots, \langle k, i_k\rangle\rangle \notin \cup_{a\in A}(F_{a1} \times \cdots \times F_{ak}).$$

Otherwise, there must exist $a \in A$ such that $t \in (F_{a1} \times \cdots \times F_{ak})$; i.e. $\langle 1, i_1\rangle \in F_{a1}, \cdots, \langle k, i_k\rangle \in F_{ak}$, which means $a \notin S_{1i_1}, \cdots, a \notin S_{ki_k}$, which in turn implies that $S_{1i_1}, \cdots, S_{ki_k}$ is *not* a FAMILY COVER. This is a contradiction.

Suppose that $t = \langle\langle 1, i_1\rangle, \cdots, \langle k, i_k\rangle\rangle \notin \cup_{a\in A}(F_{a1} \times \cdots \times F_{ak})$. We claim that $S_{1i_1}, \cdots, S_{ki_k}$ is a FAMILY COVER. Consider $a \in A$. Since $t \notin (F_{a1} \times \cdots \times F_{ak})$, it follows that for some $1 \leqq j \leqq k$, $\langle j, i_j\rangle \notin F_{aj}$, i.e. $a \in S_{ji_j}$. This reasoning holds for all $a \in A$ and hence $S_{1i_1}, \cdots, S_{ki_k}$ is a FAMILY COVER. □

THEOREM 4.2. *For tree schemas,* VI *is* NP-*complete.*

Proof. For tree schemas $t \in V$ is in P, hence the problem is in NP. Consider an NCP instance $F_1, \cdots, F_h$ with $F_i = (F_{i1}, \cdots, F_{ik})$. We now show how to build a database and view definition (forming a VI instance) such that the NCP inequality holds for the NCP instance iff there is a tuple which is not in the materialized view of the VI instance.

Construct $k+1$ relations $\mathbf{R}_0, \mathbf{R}_1, \cdots, \mathbf{R}_k$. C is the sole attribute of $\mathbf{R}_0$. For $1 \leqq i \leqq k$, $\mathbf{R}_i$ has two attributes: C and B_i. These relations constitute a tree database, whose root node corresponds to $\mathbf{R}_0$. The database state is constructed as follows:

$$R_0 = \{\langle i\rangle | i = 1, \cdots, k\},$$

$$R_m = \{\langle i, e\rangle | e \in F_{im}\}, \qquad m = 1, \cdots, k, \quad i = 1, \cdots, h.$$

Clearly,

$$V = (R_0 \bowtie \cdots \bowtie R_k)[B_1 \cdots B_k] = \cup_{i=1}^{h}(F_{i1} \times \cdots \times F_{ik}).$$

So, if there is $v \notin V$ then the NCP inequality holds. Conversely, if the NCP inequality holds, then there is a tuple $v \notin V$. □

4.3. The VCI and VGI problems. Given a database D, a view definition $\mathbf{X}$ and a tuple t, the VCI problem is to determine whether there exists $v \notin V(R_i + t)$. The problem is easily seen in Σ_2^p for general databases and in NP for tree databases; we show that it is complete in these respective cases.

The proof is done by reducing a VI instance to a VCI instance. The reduction is by adding a new column, say N, to $\mathbf{R}_1$ and setting each tuple entry in this column to a. Also, add a new relation $\mathbf{R}_N$, whose sole column is N, containing no tuples. Clearly, J on this new database is empty and so is V. However, adding $\langle a \rangle$ to R_N will yield the original view. Thus, there exists $v \notin V$ in the original database iff there exists $v \notin V(R_N + \langle a \rangle)$.

The reduction shows that VCI is Σ_2^p-complete for general databases. Observe that if the original database is a tree database, then so is the new database. Hence, VCI is NP-complete for tree databases.

Since in the previous reductions the view materialization was empty before adding $\langle a \rangle$, the same results hold for VGI.

4.4. The VCN problem. The VCN problem is to determine whether $V(R_i + t) \neq V$. This problem is clearly in Σ_2^p since a nondeterministic Turing machine can guess a "new" join tuple, extract from it a "supposed" new view tuple, and consult an oracle as to whether the new view tuple belongs to the original view. We show that VCN is Σ_2^p-complete by reducing VI, which was previously shown Σ_2^p-complete, to VCN.

The VI completeness proof actually yields that VI over databases with at most three attributes per relation schema, whose attributes all have the domain $\{true, false\}$, is Σ_2^p-complete. Given such a database D and some view definition $\mathbf{X}$, we reduce VI to VCN as follows. We construct a database $\mathbf{D}'$ by adding to each relation schema in $\mathbf{D}$ a new column, say N. We also add a new relation $\mathbf{R}_N$ with the sole attribute N.

The database D' is populated thus: To each of the original tuples in D the N column entry is set to a. We also put the tuple $\langle a \rangle$ in R_N. To each relation in D' we add all the eight $\{true, false\}$ combinations for the original columns, with the N column entry set to b.

Note that, by construction, V over D is identical to V over D'; the a values preserve the original view and the b values add nothing as b does not appear in R_N. We claim that $V \neq V(R_N + \langle b \rangle)$ in D' iff $\exists v(v \notin V)$ in D. Since all $\{true, false\}$ combinations are present in D' in conjunction with the N column value b, the addition of $\langle b \rangle$ to R_N will make the view total. So, if the view with $\langle b \rangle$ added is different than the view without $\langle b \rangle$, we conclude that the original view "missed" a tuple. Conversely, if the view prior to $\langle b \rangle$'s addition "missed" a tuple, certainly $V \neq V(R_N + \langle b \rangle)$ following $\langle b \rangle$'s addition. Hence we have proved the following theorem.

THEOREM 4.3. VCN *is* Σ_2^p*-complete.*

We now treat the VCN problem over tree databases.

THEOREM 4.4. *For tree schemas, the* VCN *problem is* NP*-complete.*

Proof. For tree schemas $t \in V$ is in P; hence VCN is in NP. By Theorem 4.2, VI is NP-complete over tree databases. In addition the reduction from VI to VCN presented above preserves the tree property of the schema because attribute N is uniformly added to each relation. Hence VCN is NP-complete over tree databases. □

4.5. The VCE, VGE and VGN problems. In § 3 it was proved that JGE, JCN and JGN are NP-complete over general databases. The corresponding view problems: VGE, VCN and VGN, are therefore also NP-complete (with $\mathbf{X} \equiv \mathbf{U}$). For tree databases VE

is polynomial, this is because V is empty iff J is empty, and JE is polynomial for tree databases. It follows that VCE and VGE are also polynomial.

THEOREM 4.5. *For tree schemas* VGN *is in* P.

Proof. Yannakakis has shown that the view materialization of a tree database can be computed by an algorithm which is polynomial in the size of the database and the size of the view materialization [28]: Let p be that polynomial.

Let D' be the database resulting from the addition of t to R_i and $V' = V(R_i + t)$. We now start producing V' from D' using Yannakakis' algorithm and abort the algorithm if it does not halt within $p(|D|+|t|, |V|)$ steps. Since $V' \supseteq V$, the views are different iff $|V'| > |V|$. If $V' = V$ then Yannakakis' algorithm must finish within $p(|D'|, |V'|) = p(|D|+|t|, |V|)$ steps. Thus if the algorithm aborted, then the views are different. On the other hand, if the algorithm terminated, then we have produced the materialization V' and can easily check whether $|V| = |V'|$. The entire procedure requires essentially $p(|D|+|t|, |V|)$ steps. □

5. Fixed schemas.[3] In the problems previously analyzed, the database schema was part of the problem instance. This section treats the case in which the database schema is *fixed* over all problem instances, i.e., problem instances differ only in the tuples in the relations.

5.1. Additions into a tree database with the view contained in one of the relations. Let us consider a special case. Suppose for some $r \leqq n$, $\mathbf{X} \subseteq \mathbf{R}_r$. Furthermore, assume that relations $\mathbf{R}_1, \cdots, \mathbf{R}_k$ constitute a tree schema. Let T be a qual tree with $\mathbf{R}_r$ at its root. ($\mathbf{R}_r$ is called the *root* relation and the relations at the leaves are called *leaf* relations.)

Let $\mathbf{R}_i$ be a node in T and $\mathbf{R}_j$ its child. Tuple $t \in R_i$ is *supported* by tuple $s \in R_j$ if t matches s. A tuple $t \in R_i$ is *good* if every child R_j of R_i has a good tuple $s \in R_j$ which supports t. Also, all tuples in a leaf relation are considered good. Tuple $t \in R_i$ is *compatible below* with a child relation R_j if there is a good tuple $s_j \in R_j$ which supports t. Hence, $t \in R_i$ is good iff t is compatible below with all of $\mathbf{R}_i$'s children.

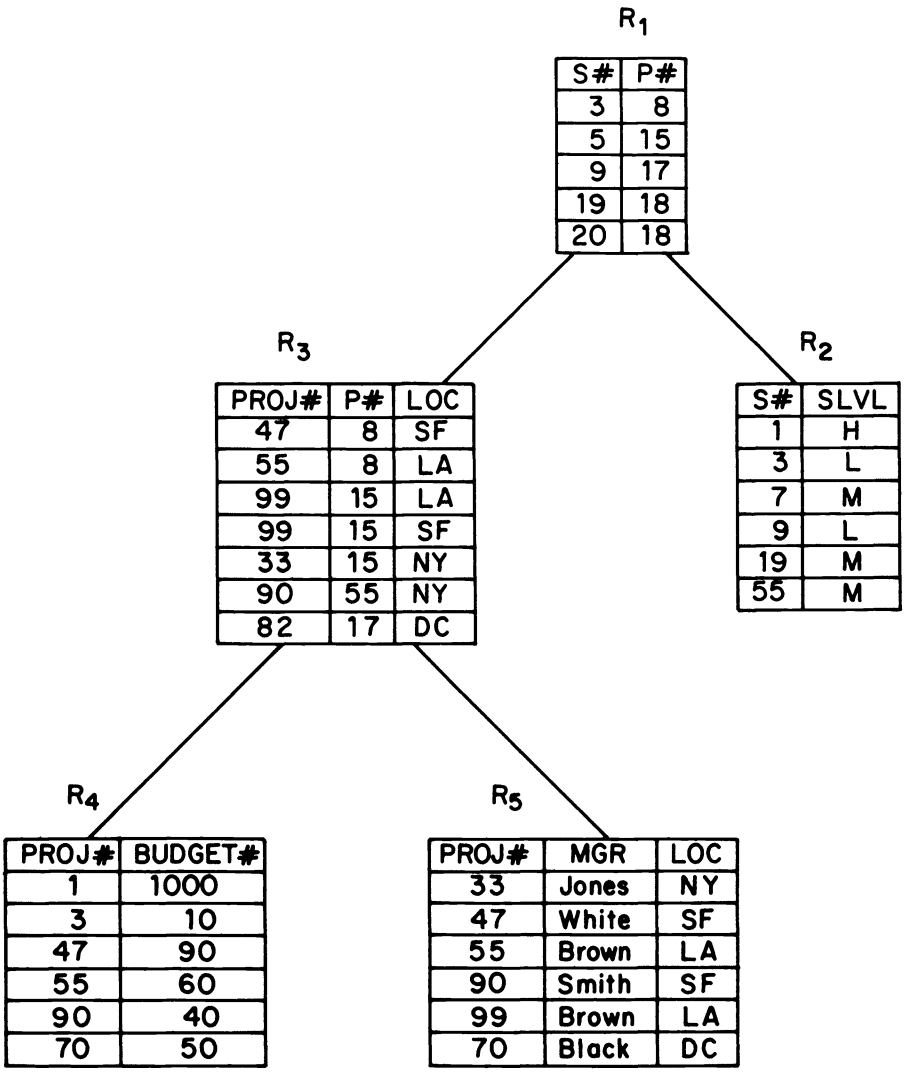
Intuitively, a tuple $t \in R_i$ is good if it is unanimously supported by all its children, its children's children and so on, i.e. t belongs to the projection onto $\mathbf{R}_i$ of all the relations in the subtree rooted at $\mathbf{R}_i$. Observe that $t \in R_i$ may contribute to $J(D)$, and therefore possibly to V, iff t is good. In other words, all nongood tuples, which we call *bad*, will definitely not contribute to $J(D)$ and V.

Consider the database of Fig. 5.1, with $\mathbf{X} \subseteq \mathbf{R}_1$. The relations $\mathbf{R}_2$, $\mathbf{R}_4$ and $\mathbf{R}_5$ are leaf relations and therefore all their tuples are good. Only the first two tuples of R_3 are good (e.g., ⟨47, 8, SF⟩ matches ⟨47, 90⟩ $\in R_4$ and ⟨47, White, SF⟩ $\in R_5$; and ⟨99, 15, LA⟩ matches ⟨99, Brown, LA⟩ $\in R_5$ but no tuple of R_4). Tuple ⟨3, 8⟩ is the only good tuple of R_1, it matches the good tuples ⟨47, 8, SF⟩ $\in R_3$ and ⟨3, L⟩ $\in R_2$. Tuple ⟨9, 17⟩ $\in R_1$ is bad because *all* the R_3 tuples it matches are bad.

The partition of each original relation into a good part and a bad part is helpful when processing updates. We start by discussing tuple addition into the tree database of Fig. 5.1. There are three cases to consider—the relation is a root, a leaf or an internal node.

(i) *Root.* Suppose $t_1 = ⟨9, 8⟩$ is added to R_1 to indicate that supplier number 9 now supplies part 8. Tuple t_1 is good since it is supported by the good tuples ⟨55, 8, LA⟩ $\in R_3$ which indicates that project 55, located at LA, requires part number 5, and ⟨9, L⟩ $\in R_2$ indicating that the service level of supplier number 9 is rated L. On the other hand,

[3] Copyright 1984, Association for Computing Machinery, Inc., reprinted by permission.



R_1: supplier $S\#$ supplies part P#;
R_2: each supplier may provide product support (indicated by SLEVEL);
R_3: project PROJ# may need part P# at location LOC;
R_4: project PROJ# has an allocated BUDGET;
R_5: project PROJ# is managed by MGR at location LOC.
The view is on $S\#$ and $P\#$.

FIG. 5.1. *An example tree database.*

adding the tuple $t_2 = \langle 7, 99\rangle$ to R_1 cannot possibly change the view since it is not supported by any good tuple of R_3. (The fact that it is supported by the good tuple $\langle 7, \text{M}\rangle \in R_2$ is immaterial.) Thus t_2 should be added to bad(R_1).

(ii) *Leaf.* Suppose $t_3 = \langle 99, 30\rangle$ is added to R_4 indicating that project 99 has been assigned a budget 30K. First, leaves only have good parts. Thus t_3 is added to good(R_4). Now, it is possible that the new addition may change the good part of R_3 (which is equivalent to changing an internal node and is discussed below). Namely, $t_4 = \langle 99, 15, \text{LA}\rangle \in R_3$, previously supported only by the good tuple $\langle 99, \text{Brown, LA}\rangle \in R_5$ is now also supported by t_3; thus t_4 should move to the good part of R_3. This effect

might propagate up the tree. On the other hand ⟨99, 15, SF⟩, which also matches t_3, remains in bad(R_3) since even now it is not supported by any R_5-tuple. To summarize, if the new tuple is good, we should check the matching tuples in the bad part of the parent node because now some of them can become good.

(iii) *Internal node.* Suppose $t_5 = \langle 70, 18, \text{DC}\rangle$ is added to R_3. Tuple t_5 is good since it is supported both by $\langle 70, 50\rangle \in R_4$ and by $\langle 70, \text{Black}, \text{DC}\rangle \in R_5$. As mentioned above, changes to an internal node may propagate upwards. We now have to check if t_5 is *compatible above*—i.e. matches with tuples in its parent relation. Indeed, t_5 matches $t_6 = \langle 19, 18\rangle$ and $t_7 = \langle 20, 18\rangle$ of R_1. Hence t_6 becomes good since it is supported by $\langle 19, \text{M}\rangle \in R_2$, while t_7 remains bad since it is not supported by any (good) R_2-tuple.

Consider an empty database over our fixed schema. To this state apply a sequence of n tuple additions (into various relations). Throughout this addition process maintain the database as above—i.e. with good-bad partitions. Compatibility above is checked only when a tuple becomes good. A tuple t is thus compared to all tuples in its parent node, and if we find a matching bad tuple s then s is checked for compatibility since potentially s may have become good. Thus, each time a tuple becomes good it initiates $O(n)$ compatibility checks. Each compatibility check compares a tuple with all the tuples in a parent (or child) node. Thus, in the worst case, each tuple is compared to all other tuples, costing $O(n^2)$ time. Thus, the cost of n additions in this naive scheme is $O(n^3)$.

The following *good-bad marking scheme* reduces the number of times t is checked for compatability below. Consider a tuple t in bad(R_3) (see Fig. 5.1). It may be there because either

(i) t[PROJ#] is not mentioned in R_4, or

(ii) t[PROJ#, LOC] is not mentioned in R_5.

However, we have no information as to which of these cases hold. To remedy this situation, with each tuple in bad(R_3) we associate *marks.* For example, an R_4-mark would indicate that t could find no match in R_4; likewise, for an R_5-mark. As relations change marks may need updating.

Data structures. We now describe the data structures employed and how the insert and delete operations are performed. Consider relation (node) R_i with tree parent R_p and children $R_1, \cdots, R_c$. Define $\mathbf{Z}_{im} = \mathbf{R}_i \cap \mathbf{R}_m$. The following balanced trees[4] are associated with R_i:

(a) For each child R_m, a tree C_{im} containing all tuples of $R_i[\mathbf{Z}_{im}]$. For each $w_m \in C_{im}$ we associate the list of tuples t in R_i with $w_m = t[\mathbf{Z}_{im}]$ and a *good-counter* indicating the number of good tuples (in R_m) that support it.

(b) T_i-containing all the tuples (good and bad) of R_i; each tuple has a *mark-counter,* which counts the number of bad marks it has, and a pointer (called the *up-pointer*) to the tuple $v = t[\mathbf{Z}_{pi}] \in C_{pi}$. ($t$ has an R_m-mark iff w_m's good-counter is equal to zero.)

We should note that in all the appearances of t in these trees, it is the *same t,* i.e., t has a record structure which allows it to concurrently be a part of several lists.

Operations. Consider a tuple t in R_i with $v = t[\mathbf{Z}_{pi}]$ and $w_m = t[\mathbf{Z}_{im}]$.

Insert (t, R_i)

A. First, t is inserted into the tree T_i and its mark-counter is set to zero.

B. For each child R_m treat C_m as follows:

[4] On a set with n elements, the operations insert, delete and member can be performed in $O(\log n)$ when the set is implemented as a balanced tree; examples for balanced tree schemes include AVL trees and 2:3 trees [1].

(a) If w_m does not appear in C_{im} then insert it into C_{im} and set its good-counter to zero. Add t to w_m's list and if w_m's good-counter $= 0$ (i.e. w_m is bad) then add 1 to t's mark-counter, thus counting the number of children of R_i that do not support t.

(b) Once all C_{im}'s have been treated, if t's mark-counter $= 0$ then t is bad and we are done; otherwise t is good and we set t's up-pointer to v's appearance in C_{pi}. (Of course, if v does not appear in C_{pi} then it is inserted.) Finally, v's good-counter in C_{pi} is incremented by 1.

(c) If incrementing v's counter transformed it from 0 to 1 then v's list is scanned and each tuple on this list has its mark-counter decremented. If now some tuple s on v's list has its mark-counter equal to zero (i.e. it became good) then stage (b) above must be (recursively) applied to s.

Delete(t, R_i)

A. Delete t from the tree T_i in R_i.

B. For $1 \leqq m \leqq c$, if t was the only tuple on w_m's list and w_m's good-counter is zero, then delete w_m from C_{im}.

C. If t was good then the good-counter associated with v in R_{pi} is decreased (if it becomes zero and v's list is empty then v is removed from C_{pi}). If v's good-counter becomes zero then v's list is scanned and each tuple has its mark-counter incremented. If some tuple's mark-counter changes from 0 to 1 then the tuple is now bad and stage C of *Delete* has to be (recursively) applied to this tuple and R_p.

Addition analysis. Consider adding a tuple t into relation R_i where the database contains n tuples (we use the same notation as above). Entering t into T_i costs $O(\log n)$. Entering t into w_m's list (recall that $w_m = t[\mathbf{Z}_{im}]$ belongs to C_{im}) costs $O(\log n)$; as there are c such trees, the overall cost is $O(c \log n)$. The analysis of t's interaction (in case t is good) with R_p is a bit more intricate. First, the good-counter of v in C_{pi} has to be incremented at a cost of $O(\log n)$. Now, if as a result of this the counter has changed from 0 to 1, mark-counters for tuples on v's list are updated. This updating may cause some bad tuples in R_p to become good and the effect propagates up the tree.

The crucial point in the analysis is that the effect propagates on the unique path from R_i to the root and that in each relation node R along the way each tuple can lose at most one mark—the one corresponding to the unique child S of R which also lies on the path from R_i to the root. Suppose $t \in R$ becomes good. Using t's up-pointer, the list in the appropriate C-tree in R_i's parent can be accessed in $O(1)$ time. This list is then traversed and marks of tuples in the list are updated. Hence, since there are n tuples in the database, the overall cost of the propagation effect is $O(n)$. Summarizing, the overall cost of inserting t is $O(c \log n + n)$.

Deletion analysis. Finding t and deleting it from T_i and the lists on the C_{im} trees can be done in $O(c + \log n)$ time. However, if t was the only tuple on a list in C_{im} and the value w_m has a zero good-counter then w_m needs to be deleted ($O(\log n)$time). Thus the overall cost of updating T and the c trees is $O(c \log n)$. If t was bad we are done. Otherwise, v's good-counter in C_{pi} is decremented; if it becomes zero then, effectively, an R_i-mark is added to the tuples on v's list. If this transforms some tuples in R_p from good to bad, the effect might propagate up the tree. Again, the number of marks that can be added to all tuples in the database in the course of a single deletion is bounded by n. Hence, the overall cost of a single deletion is $O(c \log n + n)$.

THEOREM 5.1. *Let R be a relation in a tree database with n tuples, and let R have c children. Then a single tuple can be added or deleted from R in $O(n + c \log n)$ time.*

By the above theorem, any sequence of m operations during which the database never contained more than n tuples costs $O(mn)$. Another complexity measure is

amortized cost, the cost of adding n tuples into an initially empty database. The main observation here is that in the course of n additions at most n tuples can become good and each tuple can lose at most all its marks. Thus the amortized cost for n additions (and no deletions) into a node with c children is $O(cn \log n)$. We summarize this by

THEOREM 5.2. *Consider a sequence of n additions to an initially empty database or n deletions and no additions applied to a database with n tuples. This sequence can be performed in* $O(\kappa n \log n)$ *time, where* κ *is the maximum number of children of a node in the qual tree.*

5.2. Additions into a general database. If the view attributes are not contained in any relation schema, or if the database is not a tree database, we transform the database and view to the previous case by adding new relations called *templates*. The problem of finding suitable templates will not be addressed here; see [18], [19]. One can think of templates as including in principle "all possible tuples". One way to achieve this is to let a template be total w.r.t. the database. This is fairly wasteful and we shall see other ways of maintaining templates in which only relevant tuples are maintained. In general, templates contain tuples which are computed in various ways from database relations; i.e., *template tuples* are generated from *original database tuples*.

PROPOSITION 1. *Let* $\mathbf{D} = (\mathbf{R}_1, \cdots, \mathbf{R}_k)$ *be a database and let S be a relation such that* $S \supseteq J(D)[\mathbf{S}]$. *Then for all views* $\mathbf{X}$,

$$\left(\bowtie_{i=1}^{k} R_i\right)[\mathbf{X}] = \left(\left(\bowtie_{i=1}^{k} R_i\right) \bowtie S\right)[\mathbf{X}]$$

(*i.e. a view cannot be affected by adding S*).

Proof. Apply elementary properties of the join operator. □

PROPOSITION 2. *The following example illustrates that populating a template S with less than* $(\bowtie_{i=1}^{k} R_i)[\mathbf{S}]$ *might produce incorrect views*:

Let $\mathbf{D} = (\mathbf{R}_1, \mathbf{R}_2)$, $\mathbf{X} = \mathbf{S} = \{\mathrm{B}\}$.

R_1:

A	B
1	2
3	4

R_2:

B	C
2	5
4	6

S:

B
2

.

Clearly, $(R_1 \bowtie R_2)[\mathbf{B}] \setminus S = \{\langle 4 \rangle\}$; *also,* $(R_1 \bowtie R_2)[\mathbf{X}] = \{\langle 2 \rangle, \langle 4 \rangle\}$; *and* $((R_1 \bowtie R_2) \bowtie S)[\mathbf{X}] = \{\langle 2 \rangle\}$; *therefore,* $((R_1 \bowtie R_2) \bowtie S)[\mathbf{X}]$ *is strictly contained in* $(R_1 \bowtie R_2)[\mathbf{X}]$. □

Consider first the case of a cyclic database in which the view attributes are contained in some relation; the other cases are similar. Assume the database was transformed into a tree database by adding some templates. For the good-bad mechanism to function, by Proposition 1 it is sufficient for each template S to contain $(\bowtie_{i=1}^{k} R_i)[\mathbf{S}]$ and by Proposition 2 it might not be sufficient for a template to contain less than that.

Next, we discuss various schemes for extending the good-bad mechanism to templates. Unlike relations where the "base set" of tuples is fixed, templates may undergo changes when base relation tuples are changed: the template base set may grow as a result of adding a tuple to the good set of a base relation, or shrink when such a tuple is deleted. The problem is parametrized according to the transformed schema structure, according to the following parameters:

τ: the number of templates;

γ: the maximum number of generators (defined below) per template;

κ: the maximum number of children of a node in the resulting qual tree.

Let **D** be a database schema transformed into a tree schema by adding τ templates. Consider the process of adding n tuples to an initially empty database state D. We separate the cost into two parts: that of finding the tuples to be entered into the templates, and that of entering all the tuples into the database; the latter consists of the cost of the addition of the n original tuples and the cost of adding template tuples, both using the good-bad mechanism. We have the following theorem.

THEOREM 5.3. *Adding n tuples into an initially empty transformed database with τ templates requires adding at most $O(\tau \cdot 2^n)$ tuples into templates.*

Proof. An addition of a tuple t into a relation R may introduce a "new value" $t[\mathbf{S}]$ for template **S**. Let $s = t[\mathbf{R} \cap \mathbf{S}]$. To enlarge the template,[5] we simply *duplicate* S and in one of the copies replace the $\mathbf{R} \cap \mathbf{S}$ columns with s. Thus, the addition of a tuple may double the number of tuples in each template.[6] The result follows since there are n original tuples and τ templates. □

COROLLARY. *Adding n tuples to an initially empty database requires at most $O(\kappa \tau n 2^n)$ time.*

Proof. By Theorem 5.3 $N = \tau 2^n$ tuples are added, and by Theorem 5.2 this costs $O(\kappa N \log N)$ time. □

The above result is discouraging since the cost is extremely high even for a small number of tuples. As we shall see, we can substantially improve this result.[7]

The manner in which templates are enlarged determines the cost of extending the good-bad mechanism. Let S be a template over attributes **S**. One way to generate relevant S tuples is to join enough database relations to obtain all of S's attributes. Formally, the relations $\mathbf{R}_1, \cdots, \mathbf{R}_g$ are a *generator set* for S provided $\mathbf{S} \subseteq \bigcup_{i=1}^{g} \mathbf{R}_i$; they generate $S' = (\bowtie_{i=1}^{g} R_i)[\mathbf{S}]$. S' can then be partitioned to good(S') and bad(S') by the usual procedure. In other words, we have described a method for instantiating a candidate for containing both the good and the relevant bad tuples in a template. (See Fig. 5.2(a).)

The cost of tuple additions is dominated by the correct maintenance of templates, i.e. when a tuple is added to the good part of a generator relation, the templates for which it is a generator might have to be enlarged. This means joining the new tuple with all the other generators, a potentially costly procedure ($O(\tau n^{\gamma})$). Since there are at most n such additions, the overall cost is $O(\tau n^{\gamma+1})$. (A closer analysis reveals that the cost is $O(\tau/(\gamma - 1^{\gamma-1})n^{\gamma})$.)

The following refinement reduces this cost. For each template we build a *generator tree* that is a full binary tree; the template is at its root and the generators at its leaves. An internal node consists of the join of its two child relations. (Note that the generator tree is a separate structure which comes in addition to the usual qual tree and the various balanced trees. See Fig. 5.2(b).)

In order to compute the cost of n additions into the generator relations of a template S, we make the following observations:

(1) When a tuple enters a generator relation, it has to be compared to its sibling in the generator tree in order to populate their parent.
(2) Each leaf v contains at most $\beta(v) = n$ tuples.
(3) The parent v of nodes v_1 and v_2 has at most $\beta(v) = \beta(v_1) \cdot \beta(v_2)$ tuples. Consequently, a node at distance h from the leaves has at most n^{2^h} tuples.

---

[5] Initially the template relation contains an arbitrary tuple.

[6] If duplicate tuples are eliminated from the template then its size cannot exceed $n^{|\mathbf{S}|}$ and the term 2^n may be replaced by $n^{|\mathbf{S}|}$.

[7] For $N = \tau \cdot n^{|\mathbf{S}|}$, $O(\kappa \cdot |\mathbf{S}| \cdot \log n \cdot n^{|\mathbf{S}|})$ time is required and we will improve on that as well.

(4) The cost of adding $\beta(v_1)$ tuples to a child and $\beta(v_2)$ tuples to its sibling is exactly $\beta(v_1) \cdot \beta(v_2)$, the maximum size of their parent.
(5) The cost of all the additions into a set of generators is equal to the sum of the sizes of all the internal nodes of the generator tree.

THEOREM 5.4. *Suppose n tuples are added to an initially empty database. The time required to add all template tuples is* $O(\tau(n/\gamma)^\gamma)$.

Proof. First, consider two sibling nodes in the generator tree with a total of m tuples. The number of tuples in their parent node is maximum when each of the siblings has $m/2$ tuples. Therefore, the number of tuples in a generator tree is maximum when all its leaves have the same number of tuples. The worst case occurs when there are γ leaves and exactly n/γ tuples per leaf, in which case the total number of tuples is

$$\sum_{i=1}^{\log \gamma} \left(\frac{n}{\gamma}\right)^{2^i} \frac{\gamma}{2^i} = O\left(\left(\frac{n}{\gamma}\right)^\gamma\right).$$

Since there are τ templates, the total cost is $O(\tau(n/\gamma)^\gamma)$. □

COROLLARY. *Adding or deleting a single tuple to a database containing n original tuples requires at most* $O(\tau(n/\gamma)^\gamma + \kappa\gamma \log n)$ *time.*

COROLLARY. *Adding n tuples to an initially empty treefied database requires at most* $O(\kappa\tau/(\gamma^{\gamma-1})n^\gamma \log n)$ *time.*

This is more encouraging than the corollary to Theorem 5.3 since in many practical applications γ is small.

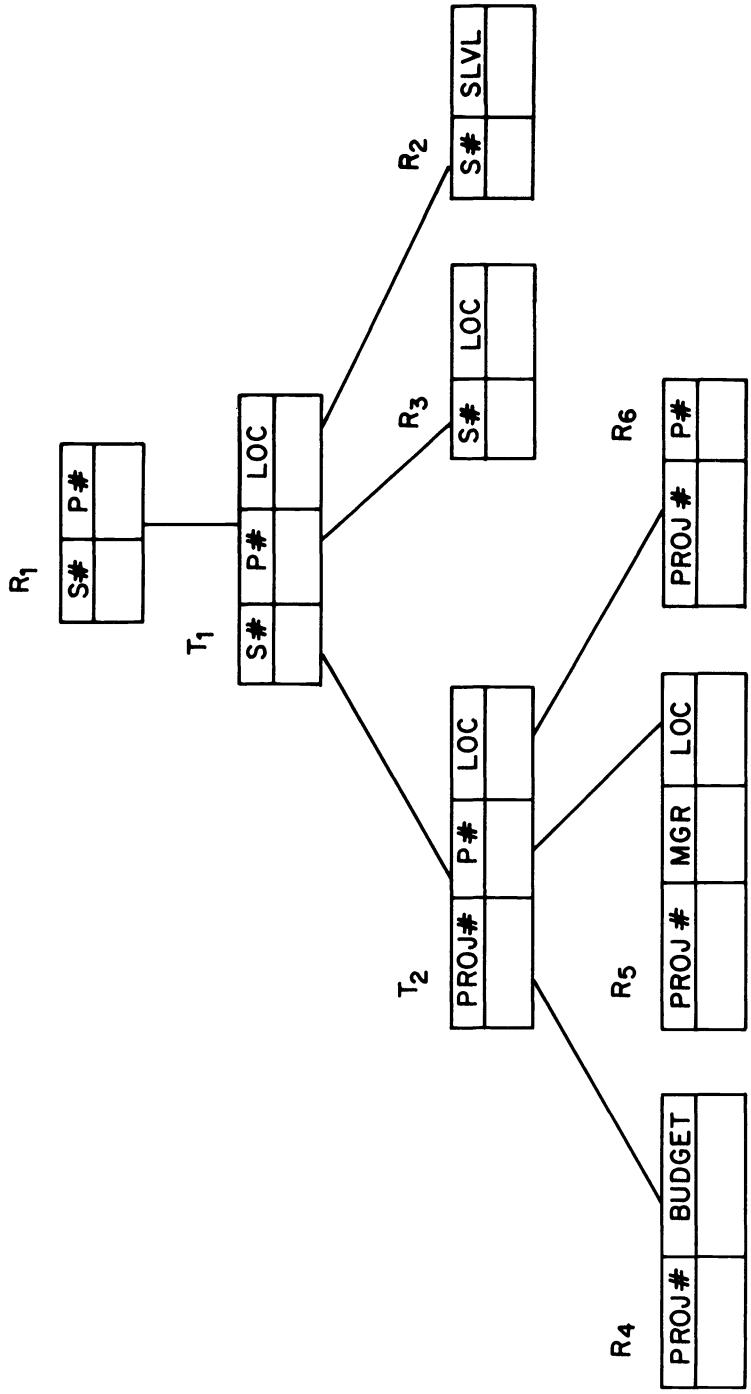
Finally, we note that the cost of a single deletion can be quite high, since it may cause many tuples in templates to become bad, costing the same as n additions. Practically, it seems better to do the following: each time we delete a tuple we also delete all tuples it helped generating (in templates). Thus, at all time, when n original tuples are in the database, there remain at most $O(\tau(n/\gamma)^\gamma)$ tuples in the database.

6. Conclusions. Several problems involving views were considered. It turns out that many view related problems are hard (Σ_2^p-complete) for arbitrary databases. Even when the database structure is relatively simple (tree databases), many problems remain NP-complete.

Each problem was treated for general databases and for the much simpler tree databases. We noticed the following "complexity reduction phenomenon"—NP-complete (Σ_2^p-complete) problems over general schemas become polynomial (NP-complete) over tree schemas. It is also interesting to note that while query processing over tree databases is polynomial, in the sense that intermediate results can be bounded by a polynomial in the input and the final result, such is not the case for view related problems. There seems to be an inherent "information loss" which makes view problems hard even on tree databases.

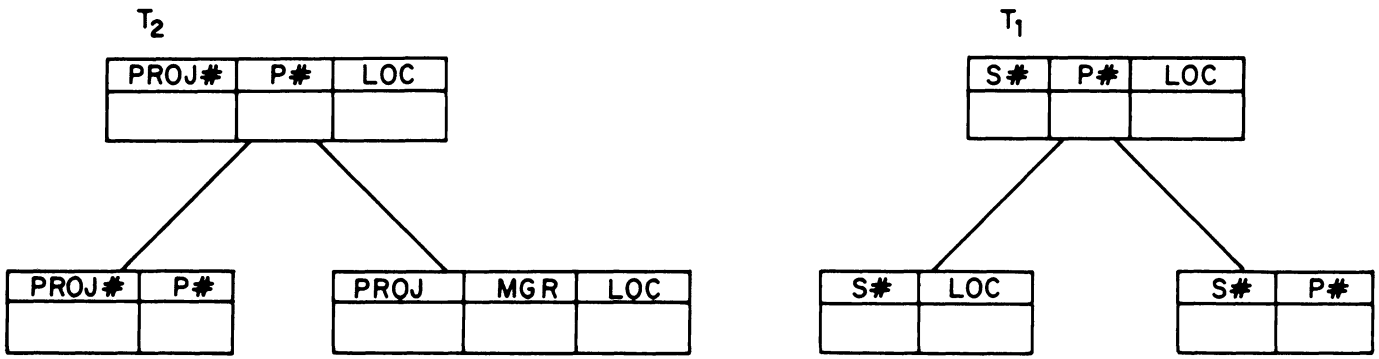
We have also examined view related problems over fixed schemas, in which only the data is allowed to vary. We have presented methods to handle this case. Their complexity is polynomial: for tree schemas the degree of the polynomial is independent of the schema structure, while for cyclic schemas the degree depends on the schema structure. Our results concerning fixed schemas are summarized in Table 6.1.

The $\log n$ factor arises from using balanced trees. We can eliminate it by using hashing, but then the results bound the average behavior, not the worst case. We do not know whether the bounds we found are tight and we leave it as an open problem. This paper also suggests additional problems such as maintaining multiple views, and that of extending the mechanism to an off-line sequence of updates to base relations.



(a) Adding templates T_1 and T_2 to the original schema.

(b) Generator trees.



R_1, R_2, R_4, R_5: are as in Fig. 5.1;
R_3: supplier S# is at location LOC;
R_6: project PROJ# needs part P#;
T_1, T_2: added templates.
The view is on S# and P#.

FIG. 5.2. *Adding templates.*

TABLE 6.1

	A single addition or deletion	A sequence of n additions
Tree databases	$O(\kappa n \log n)$	$O(n \log n)$
Cyclic databases	$O\left(\kappa \frac{\tau}{\gamma^{\gamma-1}} n^{\gamma} \log n\right)$	$O\left(\frac{\tau}{\gamma^{\gamma-1}} n^{\gamma} \log n\right)$

The complexity measure used in the analysis was the number of tuple operations. Thus the analysis is directly applicable to small scale databases whose data, or very large portions thereof, fits into memory. The tuple operations measure is inadequate for large databases in which only a small portion of the data can reside in main memory. Consider a large scale database environment. First, the balanced trees may be implemented as B-trees or replaced by a suitable hashing scheme. Second, recursive add and delete operations should be made to recurse on sets of tuples rather than on single tuples. This reduces the number of relations that are accessed at any one time, a better use of buffers is achieved and therefore secondary storage access performance is improved.

Acknowledgments. We thank R. Pinter for commenting on an earlier version of the manuscript and S. Even for simplifying the proof of Lemma 4.1.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1976.

[2] C. BERGE, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[3] P. A. BERNSTEIN AND D. W. CHIU, *Using semi-joins to solve relational queries*, J. Assoc. Comput. Mach., 28 (1981), pp. 25-40.

[4] C. BEERI, R. FAGIN, D. MAIER, A. MENDELZON, J. D. ULLMAN AND M. YANNAKAKIS, *Properties of acyclic database schemes*, in Proc. 13th Ann. ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, May 1981, pp. 355-362.

[5] C. BEERI, R. FAGIN, D. MAIER AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput. Mach., 30 (1983), pp. 479-513.

[6] P. A. BERNSTEIN AND N. GOODMAN, *The power of natural semijoins*, this Journal, 10 (1981), pp. 751-771.

[7] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational databases*, in Proc. 9th Ann. ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, May 1977, pp. 77-90.

[8] E. F. CODD, *A relational model for large shared data banks*, Comm. ACM, 13 (1970), pp. 377-387.

[9] S. S. COSMADAKIS, *The complexity of evaluating relational queries*, in Proc. ACM SIGACT-SIGMOD Conference on Principles on Database Systems, Atlanta, March 1983, pp. 149-155.

[10] U. DAYAL AND P. A. BERNSTEIN, *On the updatability of relational views*, in Proc. 4th VLDB Conference, West Berlin, September 1978.

[11] R. FAGIN, *Degrees of acyclicity for hypergraphs and relational database systems*, J. Assoc. Comput. Mach., 30 (1983), pp. 514-550.

[12] R. FAGIN, A. O. MENDELZON AND J. D. ULLMAN, *A simplified universal relation assumption and its properties*, ACM Trans. Database Systems, 7 (1982), pp. 343-360.

[13] M. H. GRAHAM, *On the universal relation*, Technical Report, Univ. Toronto, September 1979.

[14] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*completeness*, W. H. Freeman, San Francisco, 1979.

[15] N. GOODMAN AND O. SHMUELI, *Tree queries: a simple class of queries*, ACM Trans. Database Systems, 4 (1982), pp. 653-677.

[16] ———, *Syntactic characterizations of tree database schemas*, J. Assoc. Comput. Mach., 30 (1983), pp. 767-786.

[17] ———, *The tree projection theorem and relational query processing*, J. Comput. System Sci., 28 (1984), pp. 60-79.

[18] ———, *Transforming cyclic schemas into trees*, in Proc. ACM SIGACT-SIGMOD Conference on Principles of Database Systems, Los Angeles, CA, March 1982, pp. 49-54.

[19] N. GOODMAN, O. SHMUELI AND Y. C. TAY, GYO *reductions, canonical connections, tree and cyclic schemas and tree projections*, J. Comput. System Sci., 29 (1983), pp. 331-349.

[20] R. HULL, *Acyclic join dependencies and database projections*, J. Comput. System Sci., 27 (1983), pp. 231-349.

[21] P. HONEYMAN, R. E. LADNER AND M. YANNAKAKIS, *Testing the universal instance assumption*, Inform. Process. Lett., 10 (1980), pp. 14-19.

[22] D. MAIER AND J. D. ULLMAN, *Connections in acyclic hypergraphs*, in Proc. ACM SIGACT-SIGMOD Conference on Principles of Database Systems, Los Angeles, CA, March 1982, pp. 34-39.

[23] ———, *Maximal objects and the semantics of universal relation databases*, ACM Trans. Database Systems, 8 (1983), pp. 1-14.

[24] N. SPYRATOS, *Translation structures of relational views*, in Proc. 6th VLDB Conference, Montreal, 1980.

[25] O. SHMUELI AND A. ITAI, *Maintenance of views*, in Proc. SIGMOD 1984, Boston, MA, June 1984, pp. 240-255, SIGMOD Record Vol. 14, No. 2.

[26] R. E. TARJAN AND M. YANNAKAKIS, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, this Journal, 13 (1984), pp. 566-579.

[27] J. D. ULLMAN, *Principles of Database Systems*, Computer Science Press, Potomac, MD, 2nd ed., 1983.

[28] M. YANNAKAKIS, *Algorithms for acyclic database schemes*, in Proc. 7th VLDB Conference, 82-94, Cannes, September 1981.

[29] C. T. YU AND M. Z. OZSOYOGLU, *An algorithm for tree-query membership of a distributed query*, in Proc. COMPSAC79, IEEE Comp. Society, November 1979.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
004

DATA MOVEMENT TECHNIQUES FOR THE PYRAMID COMPUTER*

RUSS MILLER†‡ AND QUENTIN F. STOUT†§

Abstract. The pyramid computer was initially proposed for performing high-speed low-level image processing. However, its regular geometry can be adapted naturally to many other problems, providing effective solutions to problems more complex than those previously considered. We illustrate this by presenting pyramid computer solutions to problems involving component labeling, minimal spanning forests, nearest neighbors, transitive closure, articulation points, bridge edges, etc. Central to these algorithms is our collection of data movement techniques which exploit the pyramid's mix of tree and mesh connections. Our pyramid algorithms are significantly faster than their mesh-connected computer counterparts. For example, given a black/white square picture with n pixels, we can label the connected components in $\theta(n^{1/4})$ time, as compared with the $\Omega(n^{1/2})$ time required on the mesh-connected computer.

Key words. pyramid computer, graph-theoretic algorithms, image processing, component labeling, mesh-connected computer

AMS(MOS) subject classifications. 68Q25, 68Q20, 68U10

1. Introduction.

Pyramid-like parallel computers have long been proposed for performing high-speed low-level image processing [4], [17], [24], [32], [34]. The pyramid has a simple geometry which adapts naturally to many types of problems, and which may have ties to human vision processing. The pyramid can be projected into a regular pattern in the plane, which makes it ideal for VLSI implementation, providing thousands or millions of processing elements. At least three pyramid computers for image processing are currently being constructed [12], [23], [30].

There is no reason to limit pyramid computers to low-level image processing. They can be adapted to many other problems, and should be considered as alternatives to machines such as the mesh-connected computer. To show this, we present several new fundamental pyramid computer algorithms which are significantly faster than their mesh-connected computer counterparts. These algorithms solve problems in graph theory, image processing, and digital geometry.

The pyramid computer we consider is a combination of tree and mesh structures. Complete definitions appear in § 2, with the essentials being that a pyramid of size n has an $n^{1/2} \times n^{1/2}$ mesh-connected computer as its base, and $\log_4(n)$ levels of mesh-connected computers above. A generic processing element (PE) at level k is connected to 4 siblings at level k, 4 children at level $k-1$, and a parent at level $k+1$. (See Fig. 1.)

To date, the literature on pyramids primarily consists of two classes of algorithms. The first concentrates on the tree structure, using child-parent links. Examples of this are the component labeling in [6], [29], the feature extraction in [20], the median filtering in [31], the selection in [25], the single-figure convexity in [15], and the polygon construction in [21]. These algorithms work efficiently only when the amount of data

* Received by the editors August 23, 1983; accepted for publication (in revised form) Janaury 20, 1986. This research was supported by the National Science Foundation under grant MCS-83-01019.

† Mathematical Sciences, State University of New York, Binghamton, New York 13901.

‡Present address, Department of Computer Science, State University of New York, Buffalo, New York 14260.

§ Present address, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109.

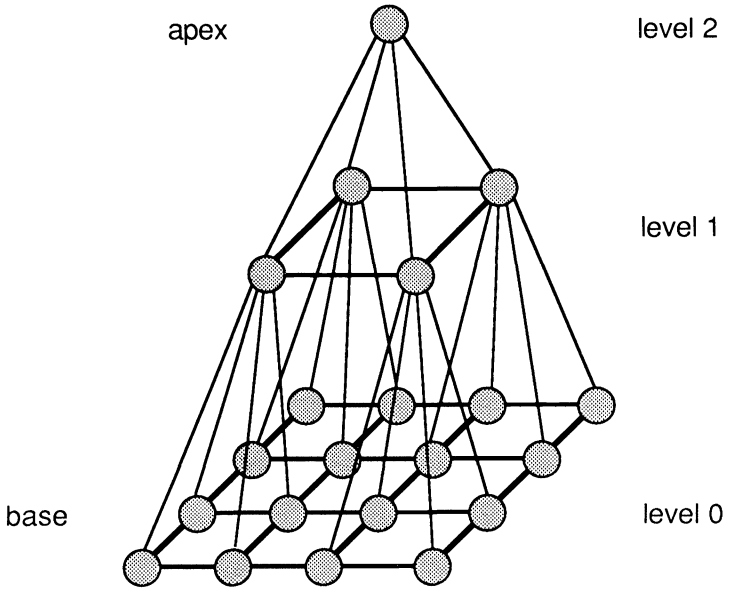


FIG. 1. *A pyramid computer of size* 16.

can be drastically reduced; otherwise too much data must pass through the apex, creating a bottleneck. The second class of algorithms concentrates on the mesh, essentially ignoring everything above the base. Examples of this are the sorting and median filtering in [25], matrix multiplication, and the multiple-label convexity in [15]. Reference [25] shows that the excessive data movement requirements of sorting force any pyramid algorithm to take $\Omega(n^{1/2})$ time. Since the base mesh can sort in $\theta(n^{1/2})$ time, the mesh oriented approach to sorting is within a multiplicative constant of being optimal.

In this paper, we consider a third class of algorithms which utilizes both types of connections. The basic approach is to reduce $O(n)$ pieces of intial data, stored one piece per base PE, down to $O(n^{1/2})$ pieces of data from which the desired result can be obtained. As has been noted for other models [10], [16], [18], this final information should be quickly moved to a region where interprocessor communication is as fast as possible, and once the answer has been obtained the results should be quickly moved to their final locations. For the pyramid this suggests moving the $O(n^{1/2})$ pieces to the middle level, which is an $n^{1/4} \times n^{1/4}$ mesh. The movement to and from the middle level is often the most time-consuming part of the algorithm, so we have developed a collection of efficient operations for performing these data movements, as well as techniques for reducing the amount of movement required.

These new data movement operations are presented for several algorithmic strategies, such as divide-and-conquer, and for various formats of the input data. They are used in several different algorithms, some of which solve various versions of the connected component labeling problem defined in § 2. In § 3, we use the pyramid read and pyramid write operations in an algorithm which labels the components of a graph of $\theta(n^{1/2})$ vertices in $\theta(n^{1/4} \log(n))$ time, where the graph is given as unsorted edges stored one per base PE. In § 4, we show that if the input is organized as an adjacency matrix, then the faster pyramid matrix read and pyramid matrix write operations reduce the time to $\theta(n^{1/4})$. In § 5, we consider input which is a digitized black/white picture,

for which we wish to label the black figures. Since each black pixel is a vertex, there may be $\theta(n)$ vertices, but the geometry of the situation allows us to use the funneled read operation to complete the labeling in $\theta(n^{1/4})$ time. These times are far better than the $\Omega(n^{1/2})$ required on a mesh-connected computer of size n [2], [18], [37].

Section 5 also introduces the operation of reducing a function over a cross-product. This is used to solve a nearest neighbor problem in which for each black component we wish to determine the label of and distance to the nearest black component. This operation is somewhat unusual in that once the relevant data has been collected at the proper level of the pyramid, it is then spread downward to finish the calculations.

In § 6, we give the detailed implementations of the data movement operations and also consider the optimality of our algorithms. In § 7, we extend the operations to pyramids of other dimensions. Throughout the paper we also solve related problems, such as marking minimal weight spanning forests, finding the transitive closure of a symmetric boolean matrix, marking articulation points, and deciding if a graph is bipartite.

2. Definitions. The *mesh-connected computer* (MCC) *of size n* is a single instruction stream-multiple data stream (SIMD) machine in which n processing elements (PEs) are arranged in a square lattice. (We assume that n is a perfect square.) PE (i, j), $1 \leqq i, j \leqq n^{1/2}$, is connected via unit-time communication links to PEs $(i \pm 1, j)$ and PEs $(i, j \pm 1)$, assuming they exist. See [7], [14], [16], [18], [33] for an overview of the MCC.

A *pyramid computer* (PC) *of size n* is an SIMD machine that can be viewed as a full, rooted, 4-ary tree of height $\log_4(n)$, with additional horizontal links so that each horizontal *level* is an MCC. A PC of size n has at its base an MCC of size n, and a total of $(4/3)n - (1/3)$ PEs. The levels are numbered so that the base is level 0 and the apex is level $\log_4(n)$. A PE at level i is connected via bidirectional unit-time communication links to its 9 neighbors (assuming they exist): 4 siblings at level i, 4 children at level $i-1$, and a parent at level $i+1$. (See Fig. 1.) We make the standard assumptions that each PE has a fixed number of words (registers), each of length $\theta(\log(n))$, and that all operations take unit time. Each PE contains registers with its row, column and level coordinates, the concatenation of which provides a unique label for the PE. (These registers can be initialized in $\theta(\log(n))$ time if necessary.)

We will illustrate the use of our data movement techniques by giving solutions to a variety of problems. Each problem involves a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. The graph can be expressed in various forms:

(a) *Unordered edge input.* The edges of the graph are initially distributed in a random fashion throughout the base of the PC, no more than one edge per PE.

(b) *Adjacency matrix input.* PE(i, j) at the base of the PC contains entry $A(i, j)$ of the adjacency matrix A of the graph.

(c) *Digitized picture input.* A digitized black/white picture is initially stored one pixel per PE in the base of the PC. The vertex set consists of the black pixels, where neighboring black pixels have an edge between them.

The problems we solve are the following:

(1) *Component labeling.* The input to the problem is an undirected graph $G = (V, E)$, given in any of the three input formats. We assume that the elements of V have a linear order. The component labeling algorithm assigns to each vertex a component label, where two vertices receive the same component label if and only if there is a connected path between them. In this paper, the component label will be chosen to be the minimum label of any vertex in the component.

(2) *Minimal spanning forest.* Given a weighted undirected graph, mark the edges of a minimal weight spanning tree for each component of the graph. The input format can be unordered edges or a weight matrix.

(3) *Nearest neighbor.* Given a digitized picture at the base of the PC with its components already labeled, calculate for each black component the label of the component nearest to it and its corresponding distance. Any l_p metric can be used to measure the distance.

(4) *Transitive closure.* Compute the transitive closure of a symmetric boolean matrix initially given at the base of the PC.

(5) *Bipartite graphs.* Given an undirected graph $G=(V, E)$, decide if G is bipartite. That is, decide if V can be partitioned into sets $V1$ and $V2$ so that each edge of G joins a member of $V1$ to a member of $V2$.

(6) *Cyclic index.* Compute the cyclic index of an undirected graph $G=(V, E)$, where the cyclic index of G is the largest number s so that V can be partitioned into sets $V(0), \ldots, V(s-1)$, such that for any edge (x, y), if x is in $V(i)$ then y is in $V((i \pm 1) \bmod s)$.

(7) *Bridge edges.* Given an undirected graph, decide which edges are bridge edges, where a bridge edge is an edge whose removal increases the number of components.

(8) *Articulation points.* Given an undirected graph, decide which vertices are articulation points, where a vertex is called an articulation point if its removal (along with its incident edges) increases the number of components.

(9) *Biconnectivity.* Given an undirected graph, decide if all components are biconnected, where a component is said to be biconnected if, for any two points in the component, there are two disjoint paths between them.

3. Graphs as unsorted edges. In this section, we assume that the graphs are given as unsorted edges stored one per PE at the base of the pyramid, where edges may be represented more than once. This format is the most general, including the others as special cases.

3.1. Data movement operations. There are several well-known data movement operations for the MCC. Two of these, the random access read (RAR) and random access write (RAW), will be defined here for the MCC and then extended to the PC. These operations involve two sets of PEs, the *sources* and the *destinations.* Source PEs send zero, one or two records, each consisting of a key and one or more data parts. (The upper limit of two records simplifies our algorithms. Most authors allow only one, in which case the operation needs to be repeated.) Destination PEs receive zero, one or two records sent by the source PEs. We allow the possibility that a PE is both a source and a destination.

MCC *Random Access Write* (RAW): In a RAW the destination PEs specify the number of records they wish to receive. At the end of the RAW, the number of records received by a destination PE is between zero and the number requested. Each record sent by a source PE is received by a destination PE, with the exception that if two or more source PEs send records with the same key then only the minimum such record is received. (In other circumstances the minimum could be replaced by any other commutative, associative, binary operation computable in $\theta(1)$ time.)

MCC *Random Access Read* (RAR): In a RAR no two records sent by source PEs can have the same key. Each destination PE specifies the keys of the records it wishes to receive, or it specifies a null key, in which case it receives nothing. Several destination PEs can request the same key. A destination PE may specify a key for which there is no source record, in which case it receives a null message.

Both the RAW and the RAR can be completed in $\theta(n^{1/2})$ time on an MCC of size n [19]. On the PC, the RAW and RAR extend to the pyramid write and pyramid read, respectively. We now describe the actions of these operations, deferring the details of their implementations to § 6.1.

A *pyramid write* has the source and destination PEs playing the same roles as in the MCC's RAW. Furthermore, all destination PEs must lie on one level and all source PEs must lie on the same level or some level below. (If both levels are the same then a given PE might be both a source and a destination.) As an example, consider the following "sample" call:

```
Pyramid write from level L up to level M,
    For all PEs on level L,
        if test1 then send(A1,B1,C1), send(A2,B2,C2);
    For all PEs on level M,
        if test2 then receive(D,E,F);
```

Since source PEs are descendents of destination PEs it must be that $L \leqq M$ in this example. *test1* and *test2* are arbitrary boolean tests. For a PE on level L, if *test1* is true then the PE creates and sends two records, one whose key is the value of $A1$, with the values of $B1$ and $C1$ as data, and a second with key $A2$, with data $B2$ and $C2$. (The key is always the first component.) If *test1* is false then the PE does not send any records. A PE on level M will not try to receive any record if *test2* is false. If *test2* is true it will try to receive a single record, where the key goes into D and data parts go into E and F. If no record is received then the values of D, E, and F become ∞.

A *pyramid read* parallels the RAR in the same way that the pyramid write parallels the RAW, except now the destination PEs are descendents of the source PEs. As an example, consider

```
Pyramid read at level L from level M,
    For all PEs on level M,
        if test1 then send(A,B)
    For all PEs on level L,
        if test2 then receive(C,D);
```

If a PE on level L requests a key C which has not been sent then D is set equal to ∞.

In § 6.1, we show how to implement the pyramid write and pyramid read. If the top level is an MCC of size m and the bottom level is $i-1$ levels below, then the time for both operations is $\theta(i+(m*i)^{1/2})$.

3.2. Component labeling. Except for obvious differences in the computer model and the data movements operations, our component labeling algorithm is similar to those in [9], [11], [13], [18], to which the reader is referred for proofs and further details. The algorithm proceeds through a series of stages, where at each stage the vertices are partitioned into disjoint *clubs*. Vertices are in the same club only if they are in the same component of the graph. We say that a club is *unstable* if it is not an entire component.

Initially each vertex is its own club. During a single *stage* of the algorithm, unstable clubs are merged together to form larger clubs, and the number of unstable clubs decreases by at least half. This is repeated until no unstable clubs remain. Since each stage reduces the number of unstable clubs by at least half, at most $\lfloor \log_2(v) \rfloor$ stages are needed for a graph with v vertices.

Each club has a unique label, which is the minimum label of any vertex in the club. During the algorithm, let $L(x)$ denote the current label of the club containing vertex x. Initially $L(x) = x$. During a stage, clubs are merged as follows: let u be a label of an unstable club. Compute $M(u) = \min\{L(y): (x, y) \in E, L(x) = u\}$. The graph whose vertices are the labels of unstable clubs and whose edges are of the form $(u, M(u))$, u an unstable club, is called a *min-tree forest.* In this forest each unstable club is connected to at least one other, which guarantees that the number of unstable clubs is at least halved after each stage. For each tree in the min-tree forest, we form a new club which is the union of all the clubs in the tree. For each unstable club u, let $N(u)$ denote the resulting label of the tree containing u. Since $N(u)$ is the minimum label of any club in u's tree, it is the minimum label of any vertex in the new larger club which contains all the vertices originally in u. For each vertex x, the new value of $L(x)$ is $N(L(x))$, and the stage is completed.

Our component labeling algorithm for a PC is given in Fig. 2. It incorporates an integer function count_keys which counts the number of distinct keys in the base. The operation of count_keys is similar to that of the pyramid write, and is given in detail in § 6.1.

An important point of the implementation is moving the data to a place where the min-tree forest relabeling can be done quickly. The forest is essentially upward directed, in that $M(u) \leqq u$, and this makes it easier to label. Reference [18] showed that if the forest has f vertices then the relabeling can be done in $\theta(f^{1/2})$ time on an MCC of size $\theta(f)$. If the forest data remained at the base, then the relabeling would take $\theta(n^{1/2})$ time. However, by first moving the data up the pyramid and then relabeling it, this step will take only $\theta(v^{1/2})$ time. We use forest_level to indicate the level of the PC on which the forest is formed. Initially, this level must have at least v PEs. Each stage reduces the size of the forest by at least half, so after 2 stages forest_level can be increased by 1. Without this upward movement, the time would increase by a factor of $\log(n)$.

THEOREM 1. *Given a pyramid computer of size n, if the base contains the unsorted edges of an undirected graph with v vertices, then the above algorithm labels the components in* $\theta\ (\log(n) + v^{1/2}[1 + \log(n/v)]^{1/2})$ *time.*

Proof. Proposition 3 of § 6.1 shows that count_keys finishes in the time claimed. Within the loop, at the start of an iteration, let k be the number of PEs at level forest_level. The pyramid read and write take $\theta(\text{forest_level} + k^{1/2}[1 + \text{forest_level}]^{1/2})$ time and the min-tree forest relabeling takes $\theta(k^{1/2})$ time. Since $k = n/4^{\text{forest_level}}$, the time for this iteration of the loop is

$$\theta(\text{forest_level} + n^{1/2}[1 + \text{forest_level}]^{1/2}/2^{\text{forest_level}}).$$

The initial value of forest_level is $\lfloor \log_4(n/v) \rfloor$ and forest_level increases every 2 iterations, so the total time of the algorithm is

$$\theta\left(\sum_{i=\lfloor \log_4(n/v) \rfloor}^{\log_4(n)} \frac{i + n^{1/2}[1+i]^{1/2}}{2^i}\right) = \theta\left(\log(n) + v^{1/2}\left[1 + \log\left(\frac{n}{v}\right)\right]^{1/2}\right). \qquad \square$$

In the worst case, when $v = \theta(n)$, our PC algorithm takes $\theta(n^{1/2})$ time, which is better than the $\theta(n^{1/2} \log(n))$ MCC algorithm of [18]. This situation arises, for example, when considering planar graphs, for which $v \leqq 3e - 6$ edges. For smaller values of v our improvement over MCC algorithms becomes even more pronounced. All MCC algorithms must take $\Omega(n^{1/2})$ time, but for a dense graph with $v = \theta(n^{1/2})$ our pyramid algorithm requires only $\theta(n^{1/4} \log^{1/2}(n))$ time.

```
For all base PEs,
    Label1:=Vertex1; Label2:=Vertex2;
    If Vertex1<∞ then  create a record with key Vertex1,
                       create a record with key Vertex2;

v:=count_keys;    {v is the number of vertices}

forest_level:=⌊log4(n/v)⌋;

for stage:=1 to ⌊log2(v)⌋ do

    Pyramid write from the base upto level forest_level,
        For all base PEs,
            send(Label1, Label2), send(Label2, Label1);
        For all PEs at level forest_level,
            receive(Vertex1, Vertex2);

    Relabel the min-tree forest, so that each PE on level forest_level has
        Label1:=N(Vertex1).

    Pyramid read at the base from level forest_level,
        For all PEs on level forest_level,
            if Vertex1<∞ then send(Vertex1, Label1);
        For all base PEs,
            receive(Label1, temp_label1),  receive(Label2, temp_label2);

    For all base PEs,
        if temp_label1<∞ then Label1:=temp_label1,
        if temp_label2<∞ then Label2:=temp_Label2;

    If (stage mod 2)=0  then forest_level:=forest_level + 1;

end {for};
```

FIG. 2. *Component labeling algorithm.*

3.3. Minimal spanning forests. The strong similarities between component labeling algorithms and minimal spanning forest algorithms are well known. In particular, others have noted that small changes to a component labeling algorithm for a parallel computer can give a minimal spanning forest algorithm for the same computer [3], [10], [22]. There are two changes that must be made to our component labeling algorithm. First, one must keep track of which edges are used. Second, when clubs are being merged each club must use an edge of minimal weight, rather than an edge to a club of minimal index. Furthermore, a club may have more than one minimal edge, which may introduce cycles. We prevent this by ordering the edges. We say that weighted edge $(w1, x1, y1)$ is less than weighted edge $(w2, x2, y2)$ if $w1 < w2$, or if

$w1 = w2$ and $\min(x1, y1) < \min(x2, y2)$, or if $w1 = w2$, $\min(x1, y1) = \min(x2, y2)$, and $\max(x1, y1) < \max(x2, y2)$.

Incorporating these changes is quite straightforward, giving the following result:

THEOREM 2. *Given a pyramid computer of size n, if the base contains the unsorted weighted edges of an undirected graph with v vertices, then the above algorithm finds a minimal spanning forest in* $\theta(\log(n) + v^{1/2}[1 + \log(n/v)]^{1/2})$ *time.*

Even if the edges are unweighted, spanning forests can be quite useful. We illustrate this with an example. To decide if an undirected graph $G = (V, E)$ is bipartite, let each edge have weight 1 and use Theorem 2 to select a spanning forest. Using a pyramid write, write the edges of the forest to level $\lfloor \log_4(n/v) \rfloor$. In each tree of the forest, select the vertex of minimum label as the root, and use the MCC algorithm in [27] to determine the depth of each vertex in its rooted tree. (This algorithm takes $\theta(v^{1/2})$ time.) Say that a node is in $V1$ if its depth is even, and is in $V2$ if its depth is odd. It is easy to show that G is bipartite if and only if this particlar choice of $V1$ and $V2$ is such that every edge of E joins a member of $V1$ and a member of $V2$. To check whether this property is true, have the base PEs use pyramid reads to determine the depths of the vertices of the edges they contain. Finally, pass these results to the apex, combining them along the way.

The above algorithm takes $\theta(\log(n) + v^{1/2}[1 + \log(n/v)]^{1/2})$ time. Furthermore, we can solve several graph-theoretic problems by using Theorem 2 to pick a spanning forest, moving the forest to level $\lfloor \log_4(n/v) \rfloor$, using an MCC algorithm at that level, and using pyramid reads and writes to move data up and down. MCC algorithms for several graph-theoretic problems are given in [27], and these can be incorporated in a PC algorithm as described, yielding:

COROLLARY 1. *Given a pyramid computer of size n, if the base contains the unsorted edges of an undirected graph G with v vertices, then in* $\theta(\log(n) + v^{1/2}[1 + \log(n/v)]^{1/2})$ *time one can*

(a) *decide if G is bipartite,*
(b) *determine the cyclic index of G,*
(c) *find all bridge edges of G,*
(d) *find all articulation points of G,*
(e) *decide if G is biconnected.*

We should mention that some of the MCC algorithms of [27] are patterned after MCC algorithms in [2], with the difference that the algorithms in [2] require matrix input while those in [27] use only edge input. The algorithms of [2] are unsuitable because there may not be v^2 PEs to hold the adjacency matrix. More important, the algorithms of [2] are too slow because they use matrix calculations that take $\theta(v)$ time on a PC.

4. Graphs as adjacency matrices. In this section, we consider undirected graphs with $n^{1/2}$ vertices, where the graph is given as an adjacency matrix or weight matrix. We assume that the (i, j) entry of the matrix is stored in base PE (i, j). Because the input is now more structured, we are able to give algorithms which are slightly faster than those of § 3.

4.1. Data movement operations. The algorithms of this section require two new data movement operations, *pyramid matrix write* and *pyramid matrix read.* A pyramid matrix write performs the same basic action as a pyramid write and comes in two versions, one for rows and one for columns. In the row (column) version source PEs lie in the base, and those in the same row (column) send the same key. The pyramid matrix read performs the same basic action as a pyramid read, and also comes in two

versions. For the row (column) version, all destination PEs lie in the base, and those in the same row (column) request the same key.

Detailed implementations of these operations appear in § 6.2, where it is shown that if a pyramid matrix write (read) has its destination (source) PEs at a level which is an MCC of size k, $k \leqq 2n^{1/2}$, then the time used is $\theta(\log(n) + k^{1/2}[1 + \log(n/k^2)]^{1/2})$. (Though we never have more than $n^{1/2}$ keys, we must allow $k = 2n^{1/2}$ since the highest level holding $n^{1/2}$ PEs actually has $2n^{1/2}$ PEs when $n > 256$ is an odd power of 4.)

4.2. Matrix algorithms. Our algorithms for graphs given as adjacency or weight matrices are simple adaptations of our algorithms for unsorted edges. We merely start with the previous algorithms, remove the call to count_keys, replace pyramid read with pyramid matrix read, and replace pyramid write with pyramid matrix write. The resulting algorithms are faster than those of § 3 by a factor of $\log^{1/2}(n)$.

THEOREM 3. *Suppose the adjacency matrix of an undirected graph with $n^{1/2}$ vertices is stored in the base of a pyramid computer of size n. Then the above algorithm labels the connected components in $\theta(n^{1/4})$ time.*

THEOREM 4. *Suppose the weight matrix of a weighted undirected graph with $n^{1/2}$ vertices is stored in the base of a pyramid computer of size n. Then the above algorithm marks a minimal spanning forest in $\theta(n^{1/4})$ time.*

COROLLARY 2. *Suppose the adjacency matrix of an undirected graph G with $n^{1/2}$ vertices is stored in the base of a pyramid computer of size n. Then in $\theta(n^{1/4})$ time one can*

(a) *decide if G is bipartite,*
(b) *determine the cyclic index of G,*
(c) *find all bridge edges of G,*
(d) *find all articulation points of G,*
(e) *decide if G is biconnected.*

Determining the transitive closure of a symmetric boolean matrix stored in the base of a PC is a simple adaptation of component labeling. First perform component labeling for matrix input. For PEs which are storing off-diagonal entries (i.e., for which the row and column are different), the new entry is 1 if the row label equals the column label, while otherwise it remains 0. For PEs on the diagonal, if the original entry was 1 it remains so, while if it was 0 then it becomes 1 only if some other entry in the row is 1. Pyramid matrix reads and writes can be used to determine the proper diagonal entries, giving the following result:

COROLLARY 3. *Suppose an $n^{1/2} \times n^{1/2}$ symmetric boolean matrix is stored in the base of a pyramid computer of size n. Then the transitive closure can be determined in $\theta(n^{1/4})$ time.*

5. Divide-and-conquer algorithms. In this section, we use a divide-and-conquer approach to solve a variety of geometric problems involving black/white pictures stored one pixel per PE at the base of the PC. The use of divide-and-conquer for geometric problems is well known, but a naive use of this strategy on the pyramid computer does not necessarily produce good results. We demonstrate some efficient implementations of this strategy on the PC.

Throughout this section we will often divide the MCC at some level into squares of some size S. What we mean by this is that we will completely partition the MCC into disjoint squares of size S, where S is a power of 4. Using this partitioning, the concept of *the square of size S at level l containing* PE P is well defined (assuming that level 1 is of size S or greater). The term *picture square* will be used to refer to a square in the base.

The computations for our divide-and-conquer solutions will proceed in a bottom-up fashion. The first *stage* will involve analyzing picture squares of size 4^c, for some small constant c which depends upon the particular problem. In general, stage i has analyzed picture squares of size 4^{c+i-1}, and stage $i+1$ combines the results together to analyze picture squares of size 4^{c+i}. An important point of our solution strategy is that at the end of stage i, each picture square of size 4^{c+i-1} has been reduced to $O(2^i)$ records of data, from which stage $i+1$ can produce the analysis for picture squares of size 4^{c+i}. Our algorithms proceed rapidly by moving the data that represents a picture square up through the subpyramid whose base is the picture square.

For a picture square of size 4^{c+i-1}, the square of size $4^{c+\lceil i/2\rceil-1}$ at level $\lfloor \frac{i}{2} \rfloor$ which contains all level $\lfloor \frac{i}{2} \rfloor$ ancestors of the PEs in the picture square is called the *data square corresponding to the picture square.* Notice that when stage i is working on picture squares of size 4^{c+i-1}, the corresponding data squares have enough PEs to contain the results from stage $i-1$, so all the work is performed in the data squares. Further, the data square corresponding to a picture square is either the union of the data squares corresponding to the picture's quadrants, or else it is the union of the parents of the quadrants' data squares. This means that the data from one stage is either already in place, or must move up only one level, for the next stage.

The last stage of the divide-and-conquer algorithm is stage $\log_4(n)-c+1$, during which the entire picture is analyzed. Since the intermediate results are scattered in data squares throughout the pyramid, a final step is needed to move these results back down to the base. This final data movement is accomplished via a funnel read, described in § 5.1. Section 5.1 also introduces a data movement operation called reducing a function. This operation allows data squares to perform some calculations (such as computing the nearest neighbor for each point from a set of points) in time linear in the edgelength of the square, even though we do not know of an MCC algorithm which finishes in this time. The operation of reducing a function uses PEs below the data square to help perform the calculations in the desired time.

5.1. Data movement operations. We now describe data movement operations that will be used to implement divide-and-conquer algorithms on the PC.

Funnel read. Assume each base PE knows the key for data it wishes to read from its stage 1 data square. For a stage i data square which is responsible for supplying the data for a given key, there are three possibilities: either one of its PEs has the data, or it must read the data from its stage $i+1$ data square (where by its stage $i+1$ data square we mean the data square it supplies data to), or one of its PEs has an alias for the key and must read the data for the alias from its stage $i+1$ data square. (If i is the last stage, then the square must have the data.) Further, a data square of size S never receives more than S requests. The funnel read ultimately obtains the data for all of the base PEs in $\theta(S^{1/2})$ time, where S is the size of the data squares at the final stage. Figure 3 is a picture of a funnel read, and its detailed implementation is in § 6.3.

Reducing a function. Given sets Q, R, and S, let g be a function mapping $Q \times R$ into S, and let $*$ be a commutative, associative, binary operation over S. Define f, a map from Q into S, by $f(q) = *\{g(q, r): r \in R\}$. We say f is the *reduction of* g. For example, if Q and R are sets of points in some metric space, if S is the real numbers, if $g(q, r)$ is the distance from q to r, and if $*$ is the minimum, then $f(q)$ is the distance from q to the nearest point in R.

Suppose the elements of Q are stored one per PE in a square of size m at level i, and the elements of R are also stored one per PE in the square. (A PE may contain an element of Q and an element of R.) Suppose g and $*$ can both be computed in

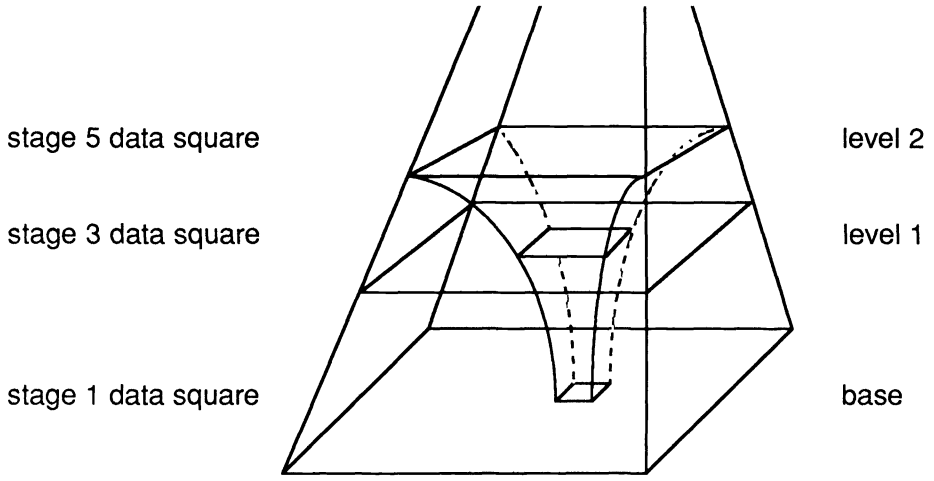


FIG. 3. *A single* PE*'s view of a funnel read.*

$\theta(1)$ time. Then the operation of reducing a function will compute $f(q)$ and store it in the PE containing q, for all $q \in Q$, in $\theta(m^{1/2}+m/4^i)$ time. The detailed explanation of this operation appears in § 6.3.

5.2. Component labeling for digitized pictures. As was noted in § 2, a digitized picture can be viewed as being an undirected graph, where the black pixels are the vertices, and adjacent black pixels have an edge between them. Upon termination of our component labeling algorithm for digitized pictures, every base PE containing a black pixel will also contain the label of the pixel's component.

ALGORITHM. We follow the basic divide-and-conquer strategy outlined above. Our algorithm is similar to the MCC algorithm of [18], but is significantly faster.

First, each base PE P containing a black pixel generates a record (p, q, ∞) for each neighbor Q containing a black pixel, where p is the index of P, q is the index of Q, and the third component of the record will be used to store the component label of p and q generated at this stage. Thus, each PE may generate as many as four records, one for each of its neighbors.

At stage 1, picture squares of size 256 are labeled. (That is, $c=4$ in the generic divide-and-conquer strategy.) This is accomplished by performing the component labeling algorithm for unsorted edge data, as described in § 3.2, simultaneously for every picture square of size 256 by using the edge data just generated in each picture square. That is, for each picture square we can apply the unsorted edge data labeling algorithm only to those records (x, y, ∞) for which both x and y are in the picture square, i.e., we omit those records for which x is on the border of the square and y is in an adjacent picture square. (Since x and y are concatenated coordinates of PEs, in $\theta(1)$ time a PE containing (x, y, ∞) can decide whether or not x and y are in the same picture square.)

When the edge data labeling algorithm is completed, those PEs with a record (x, y, ∞) use a RAR in their picture square to determine the (possibly) new label of x. The label is stored in the third field of this record. (If the third field was ∞ at the end of the edge data labeling algorithm, it must be that y lies outside of x's stage 1 picture square.) These records have all of the information needed for the next stage

of the algorithm, since the only components for which a single label has not yet been assigned are those which lie in two or more stage 1 picture squares. All records generated originally are kept in their stage 1 data squares, and each PE containing a record (x, y, z) with y outside the picture square generates a record (z, y, ∞) for use in the next stage. There are at most 64 such records generated within a single picture square of size 256. (Each corner pixel may generate 2 such records, and all other border pixels may generate 1.) These records are now spread out in their stage 1 data squares so that no PE holds more than one such record. This concludes stage 1.

At stage i, the algorithm labels picture squares of size 4^{3+i}, using data squares of size $4^{3+\lceil i/2 \rceil}$ at level $\lfloor \frac{i}{2} \rfloor$. If i is odd, then all of the data needed (the records generated at the end of the previous stage) is already present, while if i is even, then the data is in the four data squares one level below. In the latter case, the data is moved up and distributed so that no PE has more than one record. (This can be done in $\theta(2^{i/2})$ time.) Next, component labeling is performed for this edge input, again using only edges both of whose vertices lie in the picture squares. When finished, a PE containing a record with a vertex outside of the picture square generates a record for the next stage. Since we are working on picture squares of size 4^{3+i}, at most 2^{5+i} records can be generated for the next stage.

After stage $\log_4(n)-3$, all of the labels have been decided. Notice that if P's component extends outside of P's stage 1 picture square, then the labeling information in the stage 1 data square may be incorrect, and P would need to consult later stages. The component may extend outside of P's picture square for many stages, so in advance P does not know which data square has the needed labeling information. This is where a funnel read is used, moving labels from the data squares of the last stage back down towards the base, taking $\theta(n^{1/4})$ time. This gives the following result:

THEOREM 5. *The component labeling problem for digitized picture input on a pyramid computer of size n can be solved in* $\theta(n^{1/4})$ *time.*

This represents a substantial improvement over the $\theta(n^{1/2})$ MCC algorithm in [18]. Reference [29] recently presented a different PC algorithm for labeling components in a digitized picture. This algorithm is designed to label "convex blobs," but because it uses only child-parent links it takes $\theta(n^{1/2})$ time to label a $D \times n^{1/2}$ rectangle, for any constant D. In contrast, our algorithm will label any digitized picture, and hence all "convex blobs," in $\theta(n^{1/4})$ time.

5.3. Nearest neighbors. The solution to the nearest neighbor problem is quite similar to the solution just presented for the component labeling problem. Therefore, we will describe in detail only those aspects of the algorithm that change.

In the nearest neighbor problem, we wish to find the kin of each component, where the *kin* of a component is the label/distance pair representing the nearest labeled component with a different label. (In case of ties, the component of smallest label is chosen.) The input to the nearest neighbor problem is a digitized picture with its components already labeled, and at the conclusion of the algorithm each black pixel will have the kin information for its component.

Our divide-and-conquer algorithm is based on the following observation: assume that the 4 quadrants within a picture square have been analyzed. When combining the 4 quadrants, the only components whose kin could lie in a quadrant other than their own are those components that have at least one pixel that is an extreme point. An *extreme point* is a black pixel that is, relative to its component, either the northernmost or southernmost black pixel in its column, or the easternmost or westernmost black

pixel in its row. Within a quadrant, components with no extreme points must have determined their kin in earlier stages since they are totally surrounded by other components within their quadrant.

We again analyze squares of size 256 at stage 1. Within each picture square, for each component C we determine the closest component within the square, and store this in a record $(C, \text{kin}(C))$. (This kin information may be incorrect, but the final funnel read will bring the correct information down from a data square above.) For each column i in the picture square, form the records $(1, i, \text{tr}(i), \text{tl}(i))$, $(2, i, \text{br}(i), \text{bl}(i))$, where $\text{tr}(i)$ ($\text{br}(i)$) is the row of the topmost (bottommost) black pixel in the column restricted to the square, and $\text{tl}(i)$ ($\text{bl}(i)$) is the label of this pixel. (If the column has no black pixel then set the coordinates to ∞.) Similarly, for each row j we form records $(3, j, \text{lc}(j), \text{ll}(j))$, $(4, j, \text{rc}(j), \text{rl}(j))$ for the leftmost and rightmost black pixels in the row. These are the records needed for the next stage.

In general, at stage $i+1$ we first find, for each black pixel represented in one of the records passed on from stage i, the nearest black pixel (represented in a record) of a different label. We use the operation of reducing a function to do this, where Q and R are the records, S is the real numbers, $*$ is the minimum, and g is the distance, with the exception that g gives an infinite distance if the two points have the same label. When the operation is finished, we use an RAR to form a record $(C, \text{kin}(C))$ for each component C represented by one or more pixels. To generate the records for the next stage, notice that for each column in the stage $i+1$ picture square there are two type 1 records. The one representing the topmost pixel is passed to the next stage, and similar reductions occur for records of types 2, 3 and 4.

Finally, after the last stage of the algorithm, a funnel read brings the correct kin information back to the base, giving the following result:

THEOREM 6. *The nearest neighbor problem for digitized picture input on a pyramid computer of size n can be solved in* $\theta(n^{1/4})$ *time.*

We note that if one is interested only in determining, for each black pixel, the location of the nearest black pixel, then the PC needs only $\theta(\log(n))$ time [26].

6. Data movement operations. In this section, we describe how to perform the data movement operations used in earlier sections. We also discuss the optimality of our algorithms.

6.1 Pyramid read, pyramid write, and count-keys. A pyramid read starts with records stored at some level i, each with a different key, and moves them down to level j where they can be read. Let $m = n/4^i$ and $S = 4 * \lfloor \log_4 {}^{\lceil m/(i-j+1) \rceil} \rfloor$. In this algorithm, we use the term *square* to mean "square of size S", and we divide levels $j \cdots i$ into squares. The squares on level i are numbered from 1 to m/S using a snake-like ordering. (See Fig. 4.) All of the data starting in square k on level i is called *packet k.*

By a *cycle* we mean $cS^{1/2}$ time units, where the constant c, independent of n and S, is chosen so that in one cycle a square can perform all of the following functions:

1. Exchange packets with the next square on the same level (where next is with respect to the snake-like ordering).
2. Copy a packet to the four squares on the level below.
3. Perform an MCC RAR.

We now describe the pyramid read algorithm. Packets are first passed backwards along level i towards square 1, using the snake-like ordering, one square per cycle. Once at square 1, a packet is moved forwards along level i, again using the snake-like ordering. Each time a square at level i receives a packet moving forwards, it first copies

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

FIG. 4 *Snake-like ordering.*

it to the four squares below before passing it along, all in one cycle. Each square at level $(j+1)\ldots(i-1)$ which receives a packet just copies it to the four squares on the level below. Finally, each time a square on level j receives a packet it does an MCC RAR.

PROPOSITION 1. *In a pyramid computer of size n, a pyramid read from level i to level j takes* $\theta(i-j+1+[m*(i-j+1)]^{1/2})$ *time, where* $m=n/4^i$.

Proof. The operation is finished when packet m/S has moved backwards to square 1, forwards to square m/S, down to level j, and all level j squares beneath square m/S have done a RAR. This takes $2*m/S-1+i-j$ cycles, or $\theta(i-j+1+[m*(i-j+1)]^{1/2}$ time. □

For the pyramid write, assume that the destination PEs are on level i, the source PEs are on level j, and m and S are defined as above. The pyramid write is basically performed by running the pyramid read in reverse. Slight differences arise because several base PEs can send records with the same key, but perhaps different data parts, in which case we need to take a minimum. Also, it is not initially known which packet a given record will end up in.

To accommodate these problems, in general a square Z will have a packet's worth of data from each square feeding into it (either the four squares below, or, for squares at level i, the four squares below and the preceding square in the snake-like ordering). From this, Z has enough to make at least one packet's worth of data. However, since the square to which it is feeding data may have some leftover data from the previous cycle, the square it is feeding informs Z how many records are needed. In one cycle, Z supplies the necessary data and informs each square feeding into it how many of that square's records need to be replaced. Since it takes one cycle to receive the data, and one cycle for Z to pass on data after the new data is received, each step of the pyramid write takes two cycles.

Making these minor changes to the pyramid read, we obtain:

PROPOSITION 2. *In a pyramid computer of size n, a pyramid write from level j to level i can be performed in* $\theta(i-j+1+[m*(i-j+1)]^{1/2})$ *time, where* $m=n/4^i$.

Recall that count_keys is a function responsible for counting the number of distinct keys present in the base of a PC. If each key were represented only once, then count_keys could finish in $\theta(\log(n))$ time. However, keys in general are duplicated, so count_keys uses the pyramid write to eliminate duplicates. It first tries to determine if the number of keys is $\leqq K$, where $K = 4^{\lceil \log_4(\log_4(n)) \rceil}$, by doing a pyramid write to level $L = \log_4(n/K)$, where each destination PE requests one record. When finished, all PEs below level L use a pyramid read to check whether their key was passed all the way up to level L. If this is so for all PEs, then the number of keys is the number of records at level L. Otherwise, count_keys sees if the number of keys is $\leqq 4K$ by doing a new pyramid write to level $L-1$. It continues multiplying the number of keys by 4 at each stage until it reaches a stage where the pyramid write succeeds in moving all the keys to level L. This gives us the following result:

PROPOSITION 3. *If there are k different keys present in the base of a pyramid computer of size n, then in* $\theta(\log(n)+k^{1/2}[1+\log(n/k)]^{1/2})$ *time count_keys will count them.*

6.2. Pyramid matrix read and pyramid matrix write. Assume that a pyramid matrix write has its destination PEs at level i, and let $m = n/4^i$. (Recall that $m \leqq 2n^{1/2}$.) The pyramid matrix write has two steps: the first moves the data to level $j = \log_4(m)$, and the second moves it to level i. (Note: if $m = 2n^{1/2}$ then set $j = i$ instead of $i+1$.)

To perform the first step of the row version of the pyramid matrix write (the column version is similar), we partition the PEs at level j into strings of $k = 2^j$ PEs all in the same row, and call such a string and all PEs beneath it a *prism*. Notice that a prism includes parts of k rows in the base, and hence sits over no more than k different keys. In each prism, at time j the first string PE receives the minimum record sent from any base PE beneath it in the first row of its prism. This PE passes the record on to the next PE in its string. In general, the computations are pipelined so that at time $j+r-1+p-1$ the pth PE in the string of each prism receives the minimum record sent from any PE beneath it in the rth row of the prism, and also receives from the preceding string PE the minimum record sent from any base PE in the rth row beneath any of the preceding string PEs. The pth PE in the string takes the minimum of these two values and passes it to the $p+1$st PE in its string.

At time $j+k-1$ the last PE in each string forms the minimum sent by any base PE in the first row of its prism, and this value is sent back towards the first PE of its string. These reverse messages are passed simultaneously with the previously mentioned ones. Finally, at time $j+2*k-2$ the last string PE (the kth one) finds the minimum record sent by any base PE in the kth row of the prism. Simultaneously, the minimum record sent by any base PE in the first row of a prism has moved back to the first PE of its string, and the first step of the algorithm is finished.

The second step is just a pyramid write from level j to level i. This gives us the following result:

PROPOSITION 4. *In a pyramid computer of size n, a pyramid matrix write to level i,* $i \geqq \lfloor \log_4(n)/2 \rfloor$, *takes* $\theta(\log(n)+m^{1/2}[2+\log(n/m^2)]^{1/2})$ *time, where* $m = 4^i$.

Proof. If $m \leqq n^{1/2}$, then the time for the first step is $\theta(m^{1/2})$, and the time for the second step is $\theta(i-j+1+m^{1/2}[i-j+1]^{1/2})$. Since $j = \log_4(m)$ and $i = \log_4(n/m)$, we have the desired result. Otherwise, if $m = 2n^{1/2}$, then the time is $\theta(m^{1/2})$. In this case, $\log_4(n/m^2) = -1$, which is why there is a 2 instead of the usual 1 inside the brackets. □

For a pyramid matrix read, assume that the source PEs are at level i, and let $m = n/4^i$. Again we describe just the row version, which takes 3 steps. The first step uses prisms of height j, where j is as above. By using the first step of the matrix write,

in $\theta(m^{1/2})$ time the top row (string) of each prism contains the keys needed by the rows beneath. The second step is a pyramid read from level i to level j. The third step reverses the first one, taking the data to the base.

PROPOSITION 5. *In a pyramid computer of size n, a pyramid matrix read from level* i, $i \geqq \lfloor \log_4 (n)/2 \rfloor$, *takes* $\theta(\log (n) + m^{1/2}[2 + \log (n/m^2)]^{1/2})$ *time, where* $m = 4^i$.

6.3. Funnel read and reducing a function. The funnel read is straightforward. If the final stage of an algorithm is stage i, then we first have each stage $i-1$ data square use a pyramid read to obtain the data from its stage i data square. (Notice that this runs in time linear in the size of the stage i data square.) We continue downwards, each stage $j-1$ data square using a pyramid read to obtain its data from its stage j data square. When going down, the squares get smaller by a factor of 4 (see Fig. 3), so we obtain the following:

PROPOSITION 6. *Assume that the final stage of an algorithm is stage* i, *and that the stage* i *data square is an* MCC *of size S. Then a funnel read runs in* $\theta(S^{1/2})$ *time.*

Let g, Q, R, S, $*$, f, m, and l be as in the description of reducing a function in § 5.1. We use the notation $G(A, B)$ to denote the function defined on a subset A of Q whose value at $a \in A$ is given by $G(A, B)\ (a) = *\{g(a, b)\colon b \in B\}$. Notice that f is $G(Q, R)$.

For a set of PEs S, by *computing* $G(A, B)$ *in* S we mean that for each element a in A there is a PE in S which computes and stores the value of $G(A, B)(a)$. Notice that if a set $A \subset Q$ is partitioned into subsets $A1$, $A2$, $A3$, and $A4$, then

$$G(A, B) = G(A1, B) \cup G(A2, B) \cup G(A3, B) \cup G(A4, B),$$

where we view a function as a set of ordered pairs. Also, if a set $B \subset R$ is partitioned into subsets $B1$, $B2$, $B3$, and $B4$, then

$$G(A, B)(a) = G(A, B1)(a) * G(A, B2)(a) * G(A, B3)(a) * G(A, B4)(a)$$

for any $a \in A$.

Using these observations, our operation of reducing a function is also straightforward. If $m = 1$, then obviously the single PE just computes its value in $\theta(1)$ time. If $l = 0$ (i.e., if all of the data is at the base), then we merely circulate all values of R among all PEs holding members of Q, and each such PE calculates the associated f value. This takes time proportional to m, the number of PEs. Otherwise, Q is partitioned into 4 subsets (as equal as possible), $Q1$, $Q2$, $Q3$ and $Q4$, and these are arranged in the four quadrants of level l, as in the left-hand side of Fig. 5. The quadrant holding Qi is responsible for computing $G(Qi, R)$. It does this by first having each PE copy its element of Q into the PE's four children. The quadrant also copies all of R to the square beneath it, creating the situation as in the right-hand side of Fig. 5. A square of size $m/4$ on level $l-1$ holding Qi and Rj is now responsible for computing $G(Qi, Rj)$, which it does recursively. When this is finished, beneath each quadrant of level l the four squares of size $m/4$ at level $l-1$ send up their results. A PE at level l holding an element $q \in Q$ receives $G(Q, R1)(q)$, $G(Q, R2)\ (q)$, $G(Q, R3)\ (q)$ and $G(Q, R4)\ (q)$ from its children, and by taking the $*$ of these computes $G(Q, R)\ (q)$.

PROPOSITION 7. *Suppose the elements of Q are stored one per* PE *in a square of size m at level l, and the elements of R are also stored one per* PE *in this square, and suppose g and* $*$ *can be computed in* $\theta(1)$ *time. Then the reduction of g can be computed in* $\theta(m^{1/2} + m/4^l)$ *time.*

Proof. It takes $\theta(m)$ time to copy the values of Q and R from level l to the appropriate places on level $l-1$. Since the size of the squares reduces by a factor of 4 at each level, it takes $\theta(m)$ time to copy the values all the way down to the base.

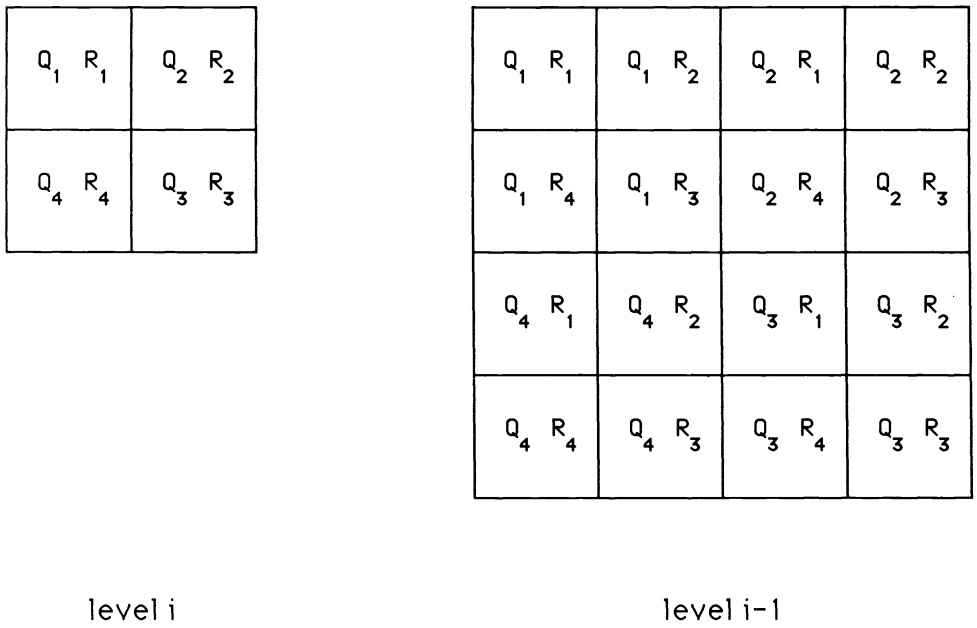


FIG. 5. *Reducing a function.*

(If $m < n^{1/2}$, then the data does not even reach the base, instead only moving down $\log_4(m)$ levels until the problem has been broken into squares of size 1.) The squares on the base have size $m/4^l$, so they take $\theta(m/4^l)$ time to compute their values. It now takes only $\theta(1)$ time per level to combine results and move them up. □

6.4. Optimality. In this section we show that some of our results are quite close to being optimal.

PROPOSITION 8. *In a pyramid computer of size n, the time needed to move* $B \geqq 1$ *bits of data from the first column of the base to the last column of the base is* $\Omega(\log(n) + (B/\log(n))^{1/2})$.

Proof. Assume $B \geqq \log_2(n)$, and let $L = \lfloor \log_4(n * \log_2(n)/B) \rfloor$ and $E = n^{1/2}/2^L$. For each column of PEs at level L, call the entire column and all of its descendents a *prism.* The data initially resides in the leftmost prism and must move to the rightmost one. If a bit only moves along communication links involving PEs at level L or below, then at least $E-1$ communication links must be traversed, since there are E prisms, and each communication link either keeps the bit in the same prism or moves it to an adjacent one.

Figure 6 shows a side view of the PEs at level L and above. The usual way of drawing the pyramid has been slightly altered so that all PEs in the same column and level are represented by a single PE. The time spent traversing any vertically drawn wires (communication links) will be ignored. The weights along all other wires indicate the number of steps that could be saved by using the wire. There are $(E/2) * (E/2-1)$ horizontal wires labeled 1, $(E/4)^2 * 2$ slanted wires labeled 1, $(E/4) * (E/4-1)$ horizontal wires labeled 3, and so forth. Since each wire can carry at most $C * \log_2(n)$ bits per unit time, for some constant C, in 1 unit of time the maximum number of bits moved by the nonvertical wires above level L is

$$C * \log(n) * \left[\sum_{i=1}^{\log_4(n)-L} (2^i - 1)(E^2/4^i - E/2^i) + \sum_{i=1}^{\log_4(n)-L-1} 2(2^i - 1)(E^2/4^{i+1}) \right],$$

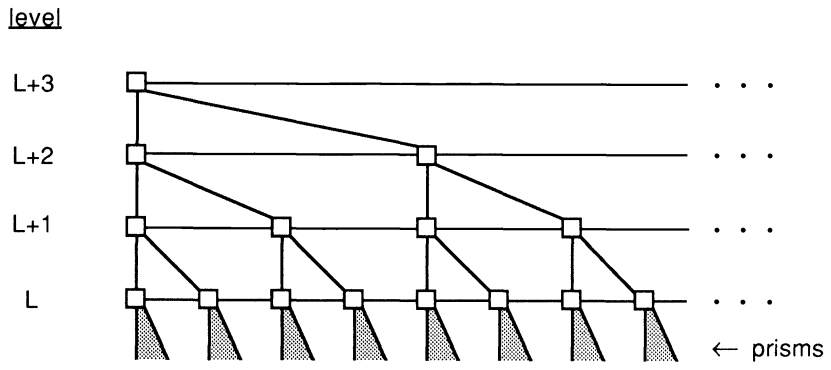


FIG. 6. *Another view of the pyramid computer.*

which is less than $C * E^2 * \log_2(n)$. Therefore, in t units of time the total savings is less than $C * E^2 * t * \log_2(n)$.

On the other hand, a bit of data that reaches the rightmost prism in t units of time must have crossed wires with a total weight of at least $E-1-t$. Furthermore, if all B bits of data reach the rightmost prism in t units of time, the total savings must be at least $B * (E-1-t)$. Therefore, t must be such that

$$B * (E-1-t) < C * E^2 * t * \log_2(n),$$

or

$$t > B * (E-1)/(C * E^2 \log_2(n) + B) = \theta((B/\log(n))^{1/2}).$$

Since the pyramid computer of size n has a diameter (maximum distance between any two vertices) of $2 * \log_4(n)$, we also have $t \geqq \log(n)$. Hence we have the desired result. □

For each of the problems considered in this paper, it is easy to devise inputs to which Proposition 8 applies. For example, suppose one is performing component labeling of digitized pictures, and the input is known to be of the form in Fig. 7, where an X indicates a pixel which may or may not be black and a Y indicates a pixel that

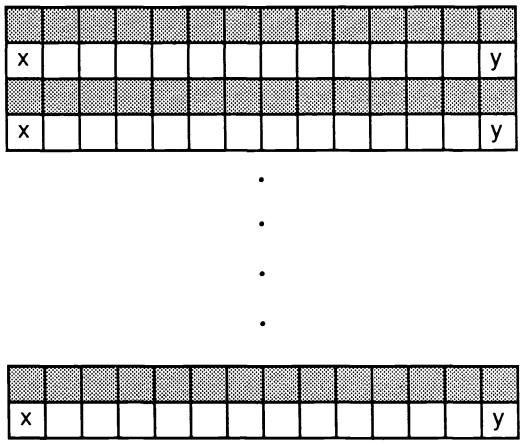


FIG. 7. *An image requiring extensive data movement.*

is always white. Notice that if the two black PEs neighboring a PE marked Y end up with the same label, then the PE marked X which is in Y's row must be black. Since a Y can determine if its black neighbors have the same label in $\theta(1)$ time after the component labeling algorithm is finished, the component labeling algorithm requires at least as much time as it takes to transmit $\theta(n^{1/2})$ bits from one edge to the opposite edge of the pyramid. By Proposition 8 this is $\Omega(n^{1/2}/\log^{1/2}(n))$. This lower bound is a factor of $\log^{1/2}(n)$ smaller than the time achieved in Theorem 5, showing that Theorem 5 is at most $\log^{1/2}(n)$ times optimal. We believe that Theorem 5, as well as Theorems 3, 4 and 6, and Corollary 2, are all optimal, and leave the proof of this as an open problem.

7. Extensions to other dimensions. Many of the data movement techniques that have been presented can be extended to pyramid computers of dimensions other than 2. Define a j-MCC *of size n* to be an SIMD machine in which n PEs are arranged in a j-dimensional cubic lattice. PE$(i(1), \cdots, i(j))$ is connected to PE$(k(1), \cdots, k(j))$, provided that $\sum_{l=1}^{j} |i(l) - k(l)| = 1$.

A j-PC *of size n* is an SIMD machine viewed as a full 2^j-ary tree with additional horizontal links. The base of the j-PC of size n is a j-MCC of size n. Each level of the pyramid is a j-MCC with $1/2^j$ as many PEs as the previous level. A PE at level i is connected to its neighbors (assuming they exist): $2 * j$ adjacent PEs at level i, 2^j children at level $i-1$, and a parent at level $i+1$.

In our analyses of algorithms for j-MCCs and j-PCs, our times consider j fixed and n, m, i and S as the parameters. We do not determine the full dependence on j, e.g., whether each step really needs 2^j comparisons instead of a fixed number, since a PE in a $j-$MCC is fundamentally different from one in a k-MCC for $j \neq k$. In this convention, sorting can be performed on a j-MCC of size n in $\theta(n^{1/j})$ time [33]. Since the MCC RAR and the MCC RAW are implemented via sorting, we can

(1) Perform an RAR in $\theta(n^{1/j})$ time on a j-MCC of size n,

(2) Perform an RAW in $\theta(n^{1/j})$ time on a j-MCC of size n.

We can also extend some of the PC data movement techniques to the j-PC. We first consider the case $j > 1$, and discuss the case $j = 1$ separately at the end.

(1) j-PC *write*: Given that the destination level is a j-MCC of size k and the source PEs are $i-1$ levels below, a j-PC write $(j > 1)$ can be performed in $\theta(\log(n) + [k * i^{j-1}]^{1/j})$ time.

(2) j-PC *read*: Given that the source level is a j-MCC of size k and the destination PEs are $i-1$ levels below, a j-PC read $(j > 1)$ can be performed in $\theta(\log(n) + [k * i^{j-1}]^{1/j})$ time.

(3) *Count keys*: In a j-PC of size n $(j > 1)$, if there are t different keys in the base then count_keys will finish in $\theta(\log(n) + [t + t * \log(n/t)^{j-1}]^{1/j})$ time.

(4) *Funnel read*: Assume that the last stage (j-dimensional) data cubes are of size S. (See the comments preceding Theorem 9 below for a discussion of the sizes of data cubes in a j-PC.) Then a j-PC funnel read $(j > 1)$ can be performed in $\theta(S^{1/j})$ time.

(5) *Reducing a function*: Given sets Q and R, stored 1 per PE in a j-MCC of size m at level i, and given that g and $*$ can be computed in $\theta(1)$ time, in $\theta(m^{1/j} + m/2^{ij})$ time a j-PC $(j > 1)$ can compute the reduction of g.

We have omitted a discussion of the pyramid matrix read and write since the mapping of a matrix into a j-MCC is not natural. Given the data movement techniques that do extend, we are able to adapt several of our algorithms to the j-PC:

THEOREM 7. *Given a j-dimensional pyramid computer of size n $(j > 1)$, if the base contains the unsorted edges of an undirected graph with v vertices, then the extension of*

the component labeling algorithm in § 3.2 *labels the components in* $\theta(\log(n)+v^{1/j}[1+\log(n/v)^{j-1}]^{1/j})$ *time.*

THEOREM 8. *Given a j-dimensional pyramid computer of size n* $(j>1)$, *if the base contains the unsorted weighted edges of an undirected graph with v vertices, then the extension of the algorithm in* § 3.3 *finds a minimal spanning forest in* $\theta(\log(n)+v^{1/j}[1+\log(n/v)^{j-1}]^{1/j})$ *time.*

COROLLARY 4. *Given a j-dimensional pyramid computer of size n* $(j>1)$, *if the base contains the unsorted edges of an undirected graph G with v vertices, then in* $\theta(\log(n)+v^{1/j}[1+\log(n/v)^{j-1}]^{1/j})$ *time one can*

(a) *decide if G is bipartite,*

(b) *determine the cyclic index of G,*

(c) *find all bridge edges of G,*

(d) *find all articulation points of G,*

(e) *decide if G is biconnected.*

In the component label problem with digitized j-dimensional picture input, we again use stages of (j-dimensional) picture cubes and their associated data cubes, where the picture cubes increase in size by a factor of 2^j at each stage. We reduce a picture cube to an amount of data proportional to the cube's surface area, so the data squares at the final stage have $\theta(n^{(j-1)/j})$ PEs, and are on level $\theta((j-1)*\log_2(n))$. The extension of component labeling for such data is now straightforward.

THEOREM 9. *Given a digitized j-dimensional picture stored one pixel per* PE *at the base of a j-*PC *of size n* $(j>1)$, *the components can be labeled in* $\theta(n^{(j-1)/j^2})$ *time.*

Despite our success in extending component labeling to the j-PC, we cannot do as well for the nearest neighbor problem. The difficulty is that, since the final data cubes have size $\theta(n^{(j-1)/j})$, when we try reducing a function it takes $\theta(n^{(j-2)/j})$ time for $j>2$, which is no better than the edgelength of the base $(n^{1/j})$. Therefore our extension to a j-PC does not seem to be able to do any better than a j-MCC algorithm for the nearest neighbor problem.

Finally, a 1-PC behaves differently than a j-PC for $j>1$. One important difference is that a 1-PC of size n can sort in $\theta(n/\log(n))$ time, versus $\theta(n)$ time for a 1-MCC of size n, while for $j>1$ a j-PC and a j-MCC sort in the same time [25]. This sorting difference is most apparent when considering problems such as component labeling, finding a minimal spanning forest, deciding if a graph is bipartite, etc., for a graph with $\theta(n)$ vertices, given as unsorted edge input. For a j-PC, $j>1$, these problems take $\theta(n^{1/j})$ time, but on a 1-PC they can be solved in $\theta(n/\log(n))$ time.

Another important difference is that one-dimensional digitized pictures are simplistic. Using the special properties of such input, there are easy 1-PC algorithms to do component labeling and finding nearest neighbors in $\theta(\log(n))$ time.

8. Conclusion. Because of its similarity to some animal optic systems, its similarity to the (region) quadtree structure, and its natural use in multiresolution image processing, the pyramid computer has long been suggested for low-level image processing [5], [6], [25], [29], [30], [32], [34], [35]. Due to advances in technology, some pyramid computers are currently under construction [8], [12], [23], [30], leading us to believe that they deserve further algorithm study. This paper begins that study by showing that the pyramid computer can be used for more complex tasks than originally considered. For instance, we have shown that the pyramid computer can be used to efficiently solve problems in higher-level image processing, graph theory, and digital geometry.

A major contribution of this paper is the introduction of fundamental data movement operations for the pyramid computer to be used with a variety of standard input formats. These data movement operations are quite different from those used by earlier authors, in that most previous pyramid computer algorithms either concentrated soley on the child-parent links (adapting quadtree algorithms to the pyramid), or solely on the mesh-connected links of the base. In contrast, the data movement operations that we have presented intermingle the use of both types of links. They also make extensive use of intermediate levels of the pyramid to do calculations, store results and communicate data. Furthermore, we have shown how to use the base of the pyramid to aid in the computation of functions being performed at higher levels.

We have used our data movement operations to efficiently solve several geometric and graph-theoretic problems. Since there are numerous other problems in these and related fields which have divide-and-conquer solutions, the problem-solving techniques and data movement operations presented here should have a wide range of applications. For example, [15], [17] contain algorithms which use the pyramid read, pyramid write, and reducing a function operations to compute geometric properties such as convexity, diameter and smallest enclosing box.

It is interesting to compare the pyramid computer with other parallel architectures. Using the standard VLSI model in which processing elements are separated by at least unit distance and a wire has unit width, [5] has shown that a pyramid computer with a base of n processing elements can be laid out in $\theta(n)$ area by a simple modification of the standard "*H tree*" layout scheme. The space of a layout for an interconnection scheme is one measure of its cost, as is the regularity of the layout. A mesh-connected computer of n processing elements also requires only $\theta(n)$ with an extremely regular layout, but because it has a communication diameter of $\theta(n^{1/2})$ it requires $\Omega(n^{1/2})$ time to solve all of the problems considered here, compared with, say, the $\theta(n^{1/4})$ time needed by the pyramid computer to label the connected components of an image. (Mesh-connected computer algorithms taking $\theta(n^{1/2})$ time to solve problems presented in this paper appear in [19], [16].)

Another simple model that can be easily laid out in $\theta(n)$ area is the quadtree machine, which is simply a pyramid computer without the nearest neighbor links. Like the pyramid, the quadtree has a logarithmic communication diameter, but unlike the pyramid, the apex often acts as a bottleneck. For example, it is easy to show that the quadtree needs $\Omega(n^{1/2})$ time to label components or find nearest neighbors of an image, even if higher PEs have additional memory (as suggested in [1]). On the pyramid, we have used nearest neighbor connections at the intermediate levels to circumvent this bottleneck.

General-purpose interconnection schemes such as the shuffle-exchange, butterfly and cube-connected cycles can be used to provide poly-log time solutions to all the problems considered herein. (An algorithm is poly-log if it finishes in $P(\log^k(n))$ time for some constant k.) Unfortunately, these interconnection schemes require area that is nearly proportional to the square of that required to lay out the pyramid computer [36]. Although this extra area and complexity provide the capability of poly-log sorting, it is more than is needed for the problems considered here.

A more interesting model is the orthogonal trees or mesh of trees [36]. This model has a mesh-connected base of size n augmented so that each row and column of the base mesh has a binary tree over it, with these trees being disjoint except at their leaves. In this model $\theta(n^{1/2}\log^2(n))$ bits can be moved from the leftmost $\log(n)$ columns to the rightmost $\log(n)$ columns in $\theta(\log(n))$ time. This is a significant improvement over the pyramid computer bound in Proposition 8, though not enough

to provide poly-log sorting. This machine model has not received much consideration as an image processing machine, but for all of the problems considered herein involving images or adjacency matrices, orthogonal trees can solve them in poly-log time.

Orthogonal trees do have some drawbacks, however. While the pyramid computer can be laid out in linear area, orthogonal trees need a factor of $\log^2(n)$ more area [36]. Further, orthogonal trees seem to have few ties to other objects of interest for researchers in image processing, as opposed to the neural, data structure, and multi-resolution ties mentioned above for the pyramid computer. It is because of these ties that the image processing community is building pyramids and not orthogonal trees. Additional models which are closer to the pyramids, and which solve all of the image processing problems considered herein in poly-log time, have recently been suggested [28]. These models were designed by starting with the pyramid computer and modifying it to be much faster on the algorithms presented here.

Acknowledgment. The authors would like to express their appreciation to the referees for their constructive comments.

REFERENCES

[1] N. AHUJA AND S. SWAMY, *Multiprocessor pyramid architecture for bottom-up image analysis*, IEEE Trans. PAMI, PAMI-6 (1984), pp. 463-474.

[2] M. J. ATALLAH AND S. R. KOSARAJU, *Graph problems on a mesh-connected processor array*, J. Assoc. Comput. Mach. 3 (1984), pp. 649-667.

[3] F. Y. CHIN, J. LAM AND I.-N. CHEN, *Efficient parallel algorithms for some graph problems*, Comm. Assoc. Comput. Mach., 25 (1982), pp. 659-665.

[4] C. R. DYER, *A quadtree machine for parallel image processing*, Tech. Report KSL 51, U. of Illinois at Chicago Circle, Chicago, IL, 1981.

[5] ———, *A* VLSI *pyramid machine for hierarchical parallel image processing*, Proc. PRIP, 1981, pp. 381-386.

[6] ———, *Pyramid algorithms and machines*, in Multicomputers and Image Processing Algorithms and Programs, K. Preston and L. Uhr, eds., Academic Press, New York, 1982, pp. 409-420.

[7] C. R. DYER AND A. ROSENFELD, *Parallel image processing by memory augmented cellular automata*, IEEE Trans. PAMI, PAMI-3 (1981), pp. 29-41.

[8] G. FRITSCH, W. KLEINOEDER, C. U. LINSTER AND J. VOLKERT, EMSY85—*The Erlanger multiprocessor system for a broad spectrum of applications*, Proc. 1983 Internat. Conf. on Parallel Processing, 1983, pp. 325-330.

[9] S. E. HAMBRUSCH, VLSI *algorithms for the connected component problem*, this Journal, 12 (1983), pp. 354-365.

[10] S. E. HAMBRUSCH AND J. SIMON, *Solving undirected graph problems on* VLSI, Tech. Rep. CS-81-23, Computer Science, Penn. State Univ., State College, PA, 1981.

[11] D. S. HIRSCHBERG, A. K. CHANDRA AND D. V. SARWATE, *Computing connected components on parallel computers*, Comm. Assoc. Comput. Mach., 22 (1979), pp. 461-464.

[12] S. LEVIALDI, *A pyramid project using integrated technology*, in Integrated Technology for Parallel Image Processing, S. Levialdi, ed., Academic Press, New York, 1985, to appear.

[13] R. MILLER, *Writing* SIMD *algorithms*, Proc. 1985 Internat. Conference on Computer Design: VLSI in Computers, 1985, pp. 122-125.

[14] R. MILLER AND Q. F. STOUT, *Computational geometry on a mesh-connected computer*, Proc. 1984 Internat. Conf. on Parallel Processing, 1984, pp. 66-74.

[15] ———, *Convexity algorithms for pyramid computers*, Proc. 1984 Internat. Conf. on Parallel Processing, 1984, pp. 177-184.

[16] ———, *Geometric algorithms for digitized pictures on a mesh-connected computer*, IEEE Trans. PAMI, PAMI-7 (1985), pp. 216-228.

[17] ———, *Pyramid computer algorithms for determining geometric properties of images*, Proc. 1985 ACM Symposium on Computational Geometry, 1985, pp. 263-277.

[18] D. NASSIMI AND S. SAHNI, *Finding connected components and connected ones on a mesh-connected parallel computer*, this Journal, 9 (1980), pp. 744-757.

[19] ———, *Data broadcasting in* SIMD *computers*, IEEE Trans. Comput. C-30 (1981), pp. 101-107.

[20] A. P. REEVES, *On efficient global feature extraction methods for parallel processing*, Comput. Graphics Image Processing, 14 (1980), pp. 159-169.
[21] B. SAKODA, *Parallel construction of polygonal boundaries from given vertices on a raster*, Tech. Rep. CS81 1-21, Computer Science, Penn. State Univ., State College, PA, 1981.
[22] C. SAVAGE AND J. JA'JA', *Fast, efficient parallel algorithms for some graph problems*, this Journal, 10 (1981), pp. 682-691.
[23] D. H. SCHAEFER et al., *The* PMMP—*a pyramid of* MPP *processing elements*, Proc. of the 18th Annual Hawaiian Internat. Conf. on Systems Science, 1, 1985, pp. 178-184.
[24] Q. F. STOUT, *Drawing straight lines with a pyramid cellular automaton*, Inform. Proc. Lett., 15 (1982), pp. 233-237.
[25] ———, *Sorting, merging, selecting, and filtering on tree and pyramid machines*, Proc. 1983 Internat. Conf. on Parallel Processing, 1983, pp. 214-221.
[26] ———, *Pyramid computer solutions of the closest pair problem*, J. Algorithms, 6 (1985), pp. 200-212.
[27] ———, *Tree-based graph algorithms for some parallel computers*, Proc. 1985 Internat. Conf. on Parallel Processing, 1985, pp. 727-730.
[28] ———, *Mesh and pyramid computers inspired by geometric algorithms*, Proc. Workshop on Algorithm-Guided Parallel Architectures for Automatic Target Recognition, 1985, pp. 293-315.
[29] S. L. TANIMOTO, *Programming techniques for hierarchical parallel image processors*, in Multicomputers and Image Processing Algorithms and Programs, K. Preston and L. Uhr, eds., Academic Press, New York, 1982, pp. 421-429.
[30] ———, *Towards hierarchical cellular logic: Design considerations for pyramid machines*, Tech. Rep. 81-02-01, Computer Science, Univ. Washington, Seattle, WA, 1981.
[31] ———, *Sorting, histogramming, and other statistical operations on a pyramid machine*, Tech. Rep. 82-08-02, Computer Science, Univ. Washington, Seattle, WA, 1982.
[32] S. L. TANIMOTO AND A. KLINGER, *Structured Computer Vision: Machine Perception through Hierarchical Computation Structures*, Academic Press, New York, 1980.
[33] C. D. THOMPSON AND H. T. KUNG, *Sorting on a mesh-connected parallel computer*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 263-271.
[34] L. UHR, *Layered 'Recognition Cone' networks that preprocess, classify, and describe*, IEEE Trans. Comput., C-21 (1972), pp. 758-768.
[35] ———, *Algorithm-Structured Computer Arrays and Networks*, Academic Press, New York, 1984.
[36] J. D. ULLMAN, *Computational Aspects of* VLSI, Computer Science Press, Baltimore, MD, 1984.
[37] F. L. VAN SOY, *The parallel recognition of classes of graphs*, IEEE Trans. Comput., C-29 (1980), pp. 563-570.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
005

ON k-HULLS AND RELATED PROBLEMS*

RICHARD COLE†, MICHA SHARIR‡ AND CHEE K. YAP§

Abstract. For any set X of points (in any dimension) and any $k = 1, 2, \cdots$, we introduce the concept of the k-hull of X. The k-hull is the set of points p such that for any hyperplane containing p there are at least k points of X in each closed half-space determined by the hyperplane.

Several computational problems related to k-hulls are studied here, including computing the k-hull and finding a point in the k-hull. Some of our algorithms are of interest in themselves because of the techniques employed; in particular, a "parametric" searching technique is used in a nontrivial way.

Key words. convex hull, computational geometry, ham sandwich, k-set, parametric search

1. Introduction. In this paper we study the computational properties of three related concepts: k-hulls, α-partitioners and the "Ham Sandwich cut." The naturalness of these notions is reinforced by the fact that they are intimately related to other concepts studied elsewhere. Furthermore, efficient algorithms related to these objects have applications to problems arising in different ways. In this introduction, we define these concepts.

Let X be a set of n planar points. A point p inside the convex hull of X can be characterized by the property that any line through p has at least a point of X in each of the closed half-planes determined by the line. Generalizing this, we define the k-hull of X, for positive k, as the set of points p such that for any line through p there are at least k points of X in each closed half-plane. Clearly the k-hull contains the $k+1$-hull, and if k is greater than $\lceil n/2 \rceil$ then the k-hull is empty. An easy consequence of Helly's theorem (any m bounded convex bodies in d-dimensional space possess a common point if every $d+1$ of them have a common point) [YB] shows that the k-hull is always nonempty for k no more than $\lceil n/3 \rceil$; the k-hull will be known as the *center* of X in this (extreme) case. Henceforth we assume $1 \leqq k \leqq \lceil n/3 \rceil$ (unless otherwise noted). It is clear that these definitions may be modified for dimensions above two.

The concept of a k-hull is apparently new but it is intimately related to other concepts which have been studied. It turns out to be (essentially) the dual of the notion of "k-belts" recently introduced by Edelsbrunner and Welzl [EW2]. The notion of "center" was first introduced by Yao [Y]. The concept of k-sets, previously studied by Lovász and others [L], [ELSS], [EW1] is also (clearly) connected to k-hulls: a k-set of X is defined to be a subset of size k of the form $X \cap H$ for some half-space

* Received by the editors September 26, 1984, and in final revised form December 9, 1985. This work was done under the auspices of the NYU/CIMS Laboratory for Robotics and Experimental Computer Science, which is supported in part by grants from Digital Equipment Corporation, IBM, The Sloan Foundation, the System Development Foundation, and the Office of Naval Research under grant N00014-82-K-0381. A preliminary version of this paper appeared in the Proceedings of the Sixteenth Annual ACM Symposium on the Theory of Computing, Washington, DC, April 30–May 2, 1984. © 1984, Association for Computing Machinery, Inc.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The work of this author was partially supported by the National Science Foundation under grant DCR-84-01633, and by an IBM Faculty Development Award.

‡ School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel. The work of this author was partially supported by a grant from the U.S.-Israeli Binational Science Foundation.

§ Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The work of this author was partially supported by the National Science Foundation under grant DCR-84-01633.

H. The maximum number of k-sets for a set of n points in the plane will be denoted by $N_k(n)$. These numbers are important in bounding the running time of our algorithms. The best known bounds are $N_k(n) = O(nk^{1/2})$ and $=\Omega(n \log k)$ [ELSS], [EW1]. A possible application of k-hulls is as a new tool in statistics: they give an alternative measure of the "interiorness" of an arbitrary point with respect to a fixed set X. Previously, the "kth iterated convex hull" (see [OVL]) was used for this purpose; a possible disadvantage of the iterated hulls (in contrast to k-hulls) is that there may be no kth iterated hull for k larger than 1.

To motivate the next concept, we recall there are "divide-and-conquer" algorithms on a set X of points in which the "divide" step calls for partitioning X into four (roughly) equal parts (called *quadrants*) using a a pair of lines. More generally, let $0 < \alpha \leqq \frac{1}{4}$. A pair of lines L_i $(i = 1, 2)$ is called an *α-partitioner* of X if each of the following subsets of X has size at most $(1-\alpha)n$: those subsets comprising the points in exactly three of the four quadrants (points on L_i are not counted in any of the subsets). Willard [W] shows that $\frac{1}{4}$-partitioners always exist. Willard gave an $O(n^2)$ algorithm for finding $\frac{1}{4}$ partitioners. This was improved to $O(N_{n/2}(n) \log^2 n) = O(n^{3/2} \log^2 n)$ by [EW3]. This was further improved by Megiddo who gave a linear time algorithm. Yao [Y] first showed that one can find α-partitioners from centers (she actually did this in 3 dimensions). Willard and Yao needed partitioners in constructing the data-structure for their half-space retrieval algorithms (in 2 and 3 dimensions, respectively). (Our initial work was directed at this problem; our results were superceded by the parallel work of Megiddo.)

The connection between k-hulls and α-partitioners can be seen in the following two facts: clearly the intersection point of a pair of lines which form an α-partitioner is in the αn-hull. Conversely, it is not hard to show that a point in the αn-hull is the intersection of some pair of lines which form an $\alpha/2$-partitioner.

The third and final concept studied here comes from the following classical result. The "Ham Sandwich Theorem" (in a form relevant to us) asserts that given d disjoint sets of points in d-dimensional space, there is a hyperplane (the "ham sandwich cut") which simultaneously divides each point set evenly. It is intuitively clear that this cut is related to partitioners. For related literature on this classical result, see [Ag], [DS], [H].

There are two techniques underlying all our algorithms. The first is a type of "parametric" searching which converts a suitable parallel searching algorithm into an efficient sequential algorithm. This is due to Megiddo [M1] although it will be seen that our use of the technique is nonobvious. The other technique[1] has its roots in a paper by Lovász [L]. We will show a partial generalization of Lovász's algorithm in 3 dimensions. In the remainder of this paper, we give algorithms for computing the k-hull in two and three dimensions, for finding the "ham sandwich cut," and for finding a point in the center. These algorithms will each illustrate one of the techniques. We conclude the paper by showing an application of these results.

We end this introduction by summarizing our results. The bounds given in the Table 1 are worst case running times.

2. Computing the k-hull. Our algorithm for computing the k-hull X, of a planar set of points, has an extremely simple and appealing structure, described thus. Define a (directed) line L to be a *k-divider* if L has at most $k-1$ points of X strictly to its right and at most $n-k$ points of X strictly to its left (so L must contain some point

[1] As Professor R. Pollack pointed out to us.

TABLE 1

Problem	Previous	New Result	Remark
Computing the k-hull in the plane	$O(N_k(n)\log^2 n)$ [EW2]	$O(N_k(n)\log^2 k + n\log n)$	[EW2] solves the dual problem
Finding a point in the center	$O(N_k(n)\log^2 n)$ [EW2]	$O(n\log^5 n)$	Recursive application of Megiddo's technique
Computing the k-hull in 3 dimensions		$O(n^3\log n)$	Partial generalization Lovász's algorithm
Ham sandwich cuts (2-dim)	$O(n^{3/2}\log^2 n)$ [EW3] $O(n)$ [M2]	$O(n\log n(\log\log n)^2)$	No restriction assuming separability
Detecting intersection of n lines with n points	$O(n^{1.5}\log n)$	$O(n^{1.41})$	Posed by J. Hopcroft

of X). Note that for any orientation $0 \leqq \theta < 2\pi$, there is a unique k-divider which we denote by L_θ. Assume (for simplicity) that no three points of X are colinear.

ROTATING LINE ALGORITHM (Lovász). Assume that the horizontal k-divider L_0 passes through a unique point p_0 of X, where L_0 is oriented towards $+\infty$ with respect to a horizontal axis. Let L be a "rotating line" and p its "pivot." Initially, let $L := L_0$ and $p := p_0$. Rotate the line L anticlockwise about p until it meets a new point q. (Note: q may be "forward" or "backward" with respect to p on L.) Set $p := q$ and repeat the rotation until L returns to its starting orientation.

Note that this "algorithm" leaves out details of implementing the rotation. But first let us show how it computes the k-hull. At this point a definition is helpful: a *special k-divider* is a k-divider that contains at least two points. We call the half-space to the left of a k-divider a *special* half-space.

THEOREM 1. *The k-hull is the intersection of the special half-spaces.*

Proof. Let H_k be the intersection of the special half-spaces. We show that H_k is the k-hull. Clearly the k-hull is included in H_k: for if p is in the k-hull and L' is a special k-divider then p must lie either on or to the left of L'. To see the reverse inclusion, if p is in H_k then any line L' through p has at least k points in its right closed half-plane since the k-divider with the orientation of L' is either tangent to, or outside of, H_k, and hence must lie to the right of L' or coincide with L'. □

The obvious implementation of the rotations in the above algorithm calls for a sorting of the rotational order of X about each point p in X, costing $O(n^2\log n)$. This preprocessing is the most expensive part of the algorithm. Finding the k-hull takes an additional $O(N_k(n)\log n)$ time. This actually proves a stronger result as follows.

THEOREM 2. *All k-hulls of X can be (simultaneously) computed in $O(n^2\log n)$ time. (In fact, as shown in the remark below, $O(n^2)$ time can be achieved.)*

If only one k-hull is asked for, we can do better as follows.

THEOREM 3. *A single k-hull can be computed in time $O(N_k(n)\log^2 k + n\log n)$ time.*

Proof. Consider the rotating line L as in our original algorithm. At each instant except when the pivot changes L is pivoting about a point p and partitions $X - \{p\}$ into two sets, X_0 and X_1. Note that the next pivot point q is on the convex hull of either X_0 or X_1. Assuming that each X_i is stored in a dynamic convex hull structure of [OVL], we can easily determine q in $O(\log n)$ time. Then we update p to q, and if q is in X_i ($i = 0$ or 1) we insert p and delete q from X_i, in $O(\log^2 n)$ time. Thus we

take $O(N_k(n)\log^2 n)$ time to do all the rotation. We compute the intersection of the special half-spaces by maintaining the intersection of the special half-spaces found so far. Forming the intersection of the at most $N_k(n)$ special half-planes takes time $O(N_k(n))$, as we discuss below. This gives a running time of $O(N_k(n)\log^2 n)$ which is not quite Theorem 3.

To improve this asymptotic bound, we first need a definition of the *k-iterated convex hull* (kth hull, for short). If k is one, this is just the usual convex hull. For k larger than one, this is the convex hull of the point set after excluding those points that are vertices of the jth hull, for j smaller than k. In [OVL] it is shown that all iterated convex hulls can be computed in time $O(n\log^2 n)$. Recently Chazelle has improved this to $O(n\log n)$ [C].

We now have two observations. The first is that the k-hull contains the kth hull. In particular, for each k-divider there is some r, no larger than k, such that the divider intersects the jth hull for $j=1, 2, \cdots, r$ but no other ith hulls. Using this fact, it is not hard to replace the sets X_i $(i=0, 1)$ used in the above algorithm by subsets Y_i of X_i which have size at most r. (The sets Y_0, Y_1 contain, for $1 \leqq l \leqq r$, the point that is the vertex of the lth hull to the right, respectively left, of L.) Thus the time to do a single rotation is $O(\log^2 k)$. We also have to keep track of where the current divider may cut the $r+1$st-hull. This is just a matter of determining when the divider becomes a tangent to this hull. To do this we traverse the $r+1$st-hull, keeping track of the vertex which has a tangent parallel to the divider. Since the divider turns through an angle of 2π we go round each ith-hull at most once, so the traversal time is at most $O(n+N_k(n))$. (We only advance the vertex on the $r+1$st hull—if the value of r changes we then "catch up" on the new $r+1$st-hull.)

The second observation is that we can form the intersection of the special half-planes in amortized time $O(1)$ per plane. We represent the intersection of all the special half-planes found so far in the algorithm by the sequence (in the order of their discovery)

$$l_1, l_2, \cdots, l_m \qquad (m \geqq 0) \tag{1}$$

where the l_i's are the *nonoccluded* special k-dividers: we say a special k-divider l is *occluded* if the special half-plane which it determines contains the intersection of all the other special half-planes found so far. Now suppose l is a newly found special k-divider. It is not hard to see that it will occlude some suffix of the sequence (1), say $l_h, l_{h+1}, \cdots, l_m$ for some $1 < h \leqq m+1$ ($h = m+1$ means nothing is occluded). We remove each occluded special k-divider, taking time $O(1)$ per removal. Each special k-divider is removed at most once, so forming the intersection of the $N_k(n)$ special half-spaces takes time $O(N_k(n))$. □

This improves slightly the result of [EW2]. The crucial technique here (as well as in [EW2]) is the algorithm, due to van Leeuwen and Overmars [OVL], for the dynamic maintenance of the convex hull of a point set.

Remark. (Due to a referee.) All k-hulls can be computed in $O(n^2)$ time, as follows.

(i) Construct the dual arrangement of the point set in $O(n^2)$ time [EOS]. (The dual is defined by mapping the point (a, b) to the line $y = ax + b$.)

(ii) An edge of the arrangement is defined to belong to the k-belt if it is above k edges, including itself. Identify the edges belonging to the k-belt, for each k, by a traversal of the arrangement from left to right. Consider a vertex that is the right endpoint of two edges, one in the k-belt, the other in the $(k+1)$-belt. This vertex is the dual of a line; each of the edges is the dual of a point. The line passes through the two points. Consider the closed right half-space defined by the line; this half-space contains at least $k+1$ points, while the open right half-space defined by the line

contains at most $k-1$ points. Thus this line is a special k-divider. In fact, each special divider is the dual of a vertex in the arrangement. Next, we show that, for each k, the order in which the special k-dividers are encountered in the traversal is opposite to the order obtained in the proof of Theorem 3; thus we will have obtained the special k-dividers in clockwise order. This follows because the sequence of special k-dividers comprises the duals of a sequence of points ordered by increasing x-coordinate; hence the lines are ordered by decreasing slope. (Strictly speaking, we obtain two sequences: one of special k-dividers, and one of special $(n-k+1)$-dividers; when we concatenate the first sequence with the second sequence we have the complete sequence of special k-dividers.)

(iii) For each k, we intersect the special half-spaces, as in the proof of Theorem 3.

The generalization of Lovász's algorithm to higher dimensions has been mentioned as an open problem in [ELSS]. We shall give a partial generalization (in "2.5-dimension"). This can be vividly described in the following way.

A rotating strip algorithm for a bicolored set of points. Let X be a set of n planar points colored either red or black. For any (oriented) line L and point p in X, we say p is "good" with respect to L if p is red and strictly on the right side of L or if p is black and strictly to the left. "Bad" points are those not on L which are not "good" (so points on L are neither good nor bad). Call L a *k-divider* of X if and only if it has at most $k-1$ good points and at most $n-k+1$ bad ones. (In contrast to the original dividers for a monochromatic set, L need not pass through a point of X and is not unique for a given orientation.) If necessary, to distinguish this new type of divider, we call it a *dichromatic* divider, in contrast to the original *monochromatic* divider. Let r, b be nonnegative integers with $k-1=r+b$. For a given orientation there is at most one (perhaps no) strip of dichromatic k-dividers which have r good red points. We can generalize the rotating line algorithm to a "rotating strip" ("strip" because the potential dichromatic k-dividers lie within a strip) algorithm to find all the dichromatic k-dividers which have r good red points, and which pass through two points of X. We say a dichromatic k-divider passing through two points of X is a *special* k-divider. Our algorithm finds all special k-dividers (we need these to find the k-hull in 3 dimensions, as explained below).

Let $L_R(\theta, i)$ (resp. $L_B(\theta, i)$) be the monochromatic i-divider of the red (resp. black) points with orientation θ. For $r=0, \cdots, k-1$, $LL(\theta, r)$ denotes the intersection of the two strips bounded by $L_R(\theta, r+1)$ and $L_R(\theta, r)$, and $L_B(\theta+\pi, k-r)$ and $L_B(\theta+\pi, k-r-1)$, respectively, where for each strip we include the former, but not the latter, bounding line. See Fig. 1 for an illustration. (There are two special cases. First, if the two bounding lines of a strip are the same line then the strip is defined to be empty. Second, we note $L_R(\theta, 0)$, and $L_B(\theta+\pi, 0)$ are not defined. So, for $r=0$, instead of using the (undefined) strip bounded by $L_R(\theta, 1)$ and $L_R(\theta, 0)$, we use the closed halfplane bounded by $L_R(\theta, 1)$, where the only red points in the halfplane are on the line $L_R(\theta, 1)$. The case $r=k-1$ is handled similarly.) We observe that $LL(\theta, r)$ contains exactly those dichromatic k-dividers with orientation θ having r good red points. Also, apart from points on its boundary, $LL(\theta, r)$ contains no points.

It follows from this observation, that for each θ and r, there are at most two special k-dividers of X with orientation θ and with r good red points. We can now give a simple algorithm to find all special k-dividers with r good red points: Begin by finding $L_R(0, r+1)$, $L_R(0, r)$, $L_B(\pi, k-r)$, and $L_B(\pi, k-r-1)$. We keep track of the relative ordering of these lines together with the five (possibly empty) subsets of X that lie strictly between two adjacent lines. These five sets are maintained in the dynamic

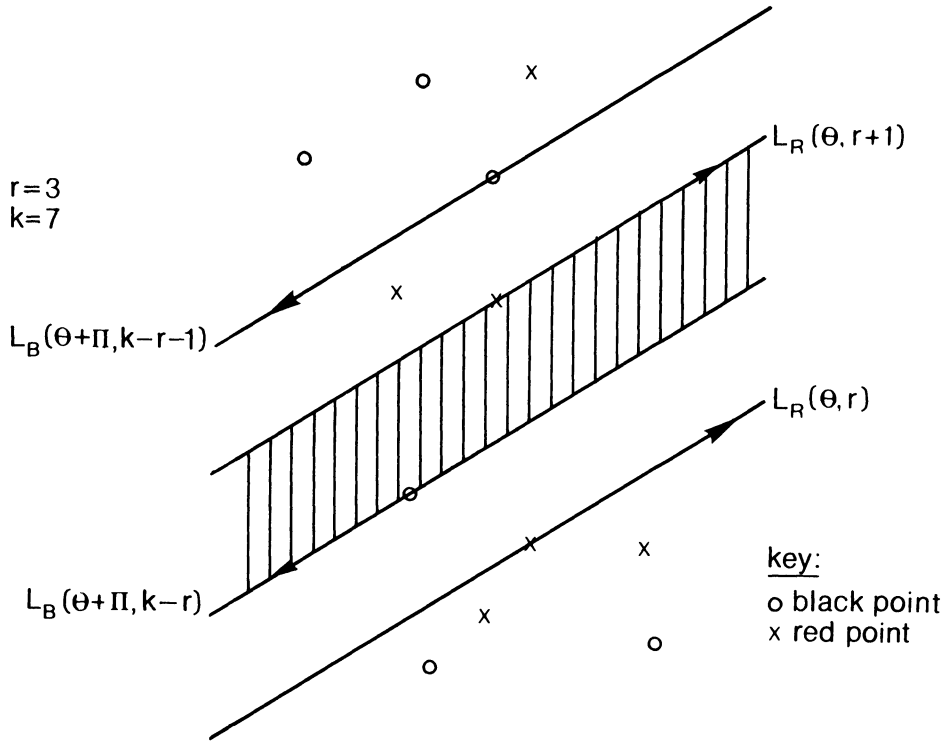


FIG. 1

convex hull structures of [OVL]. Now we can rotate the lines L_R and L_B (say, anticlockwise) simultaneously, updating each of the sets in the obvious manner. Each time one of these lines passes through a special k-divider of X, record its position. Each step of the rotation can be done in time $O(\log^2 n)$.

It is not hard to extend the just-described "rotating strip" algorithm to allow the simultaneous rotation of the strips for $r = 0, \cdots, k-1$. Instead of rotating four parallel lines, we now rotate $2k$ parallel lines. This determines $2k+1$ subsets of the points which we again maintain in dynamic convex hull structures. By suitable bookkeeping of the evolving information, we can detect all special k-dividers of X as they are passed through. Each step of the rotation takes $O(\log^2 n + \log k) = O(\log^2 n)$ time: the $\log k$ term comes from having to take the minimum of $2k$ possible increments of the current orientation.

Let us assess the complexity of this algorithm. We count the number of "events" where an event occurs when one of the $2k$ dividers passes through two points of X. To count the number of "dichromatic" events we first show the following.

LEMMA 1. *Let p be an arbitrary point, which does not coincide with a red point and is not colinear with two red points. The number of times, m, during the algorithm when p crosses any of the k rotating dividers of the red points is at most $2k$.*

Proof. Consider L_r, the $r+1$-divider (for some r) of the red points. Let q_i $(i = 1, 2)$ be the pivots of L_r during any two crossings of p over L_r. If the two crossings are consecutive in time (i.e. there are no other crossings during the interval between these two crossings) then we claim that L_r is oriented from q_1 to p (during the first crossing) if and only if L_r is oriented from p to q_2 (during the second crossing) (see [ELSS] for a similar result). (If several red points are collinear with p (on line K, say), we need

to define how many crossings of p occur when, and if, $L_r = K$. This is quite simple: one crossing occurs if the pivots before and after $L_r = K$ are on the same side of p along K, while two crossings occur otherwise.) Let Q_r be the set of points q such that q is the pivot of the rotating $r+1$-divider of the red points during some instance when p crosses L_r. Let s be the least r for which Q_r is nonempty. Let $Q = \bigcup_{r=s}^{k-1} Q_r$. Then $|Q| \leqq m$, and as each $q \in Q$ is a pivot for just one crossing, $|Q| = m$. For any line L through p the above claim shows that at least $\frac{1}{2}\max\{0, |Q_r| - 2\}$ points of Q_r lie on each side of L. Thus at least $\max\{0, (m-2k+2s)/2\}$ points of Q lie on each side of L. Choosing L to be L_s (which can be done as Q_s is not empty), we see that there are at most s points strictly to the right of L. Thus $s \geqq (m-2k+2s)/2$, or $m \leqq 2k$. □

Each dichromatic event corresponding to a red divider crossing a black point can be charged to the black point; conversely when a black divider crosses a red point, the red point is charged (actually, for a pair of points, one red, one black, the two events are simultaneous). The number of charges to each point is $O(k)$ by the lemma. Thus there are $O(nk)$ dichromatic events. A simple modification of the above lemma to allow p to be one of the red points leads to the following.

COROLLARY. *There are at most nk monochromatic events.*

Proof. We show that there at most $2k$ monochromatic events per point. As each event involves two points we deduce there are at most nk monochromatic events. A monochromatic event occurs each time we find a special r-divider of a monochromatic set, for $r \leqq k$. So we count the number of special r-dividers of the red set passing through some red point p, for $r \leqq k$. Consider S_p, the set of red points with p removed. A line through p is an r-divider of S_p if and only if it is an r-divider of $S_p \cup \{p\}$. So it suffices to count the number of r-dividers, $r \leqq k$, of S_p that pass through p. But by Lemma 1 there are at most $2k$ of these. □

Remark. This is an improvement of our original result (by a multiplicative constant). We obtained it after having learned that Alon had proved that n points in 2 dimensions realize at most nk r-sets for $r \leqq k$, a result that is implied by the corollary.

Thus the total number of events (monochromatic and dichromatic) is $O(nk)$. Since each event incurs $O(\log^2 n)$ work, we conclude that finding all special k-dividers of X takes $O(nk \log^2 n)$ time.

If k is $\omega(n/\log n)$, it is slightly more efficient to first preprocess the rotational order of X about each point p in X, using time $O(n^2 \log n)$. With this preprocessing, each event can be processed in $O(\log n)$ time and so the rotation takes a total of $O(nk \log n)$ time. We have now shown:

THEOREM 4. *The special k-dividers of a bicolored set of n points in the plane can be computed in time* $O(n \log n \cdot \min\{n, k \log n\})$.

In 3 dimensions, the k-hull of a set X of points is defined to be the set of points p such that any plane through p has at least k points in each closed half-space determined by the plane. Again by Helly's theorem [YB], we know that the k-hull is nonempty if k is no more than $\lceil n/4 \rceil$. Again we call the k-hull in this (extremal) case the *center* of the set of points.

Planes are assumed to be oriented and the points of X are *nondegenerate* in the sense that no four points are co-planar. We define a *k-dividing plane* (or k-divider) to be a plane with at most $k-1$ points of X in the (open) positive side of the plane and at most $n-k$ points of X in the (open) negative side. Note that each orientation in space determines a unique k-divider. The k-divider is *special* if it contains 3 points of X. The closed half-space on the negative side of a special k-divider is called a *special k-half-space.* By a proof similar to that of Theorem 1, we obtain the following.

THEOREM 5. *The k-hull is the intersection* H_k *of all the special k-half-spaces.*

In 3 dimensions, there is no obvious "rotating plane" algorithm generalizing the algorithm in 2 dimensions. To simplify the problem, we fix an arbitrary point p_0 in X and consider rotating a plane P such that it continuously contains p_0. Without loss of generality, assume p_0 is the only point of X which lies on the $z=1$ plane. We now reduce to the "2.5 dimensional" algorithm.

Using p_0 as the center of projection, the rest of the points are projected onto the $z=0$ plane. The projected points on the $z=0$ plane are colored "black" (resp. "red") if the original point lies above (resp. below) the $z=1$ plane. Let $\hat{X}$ denote this set of (bicolored) projected points. The projection through p_0, onto the $z=0$ plane, of plane P, containing p_0, is the intersection of P with the plane $z=0$. Thus a plane P is a special k-divider if and only if $L(P)$ is a special k-divider of $\hat{X}$.

To find the special k-half-spaces of a set of n points in 3 dimensions, we find the special k-dividers for the projected problem using p as the center of projection, for each p in X. From these we obtain the corresponding special k-dividers in 3 dimensions. Since there are $O(n^2k)$ special k-half-spaces, their intersection can be found by a divide-and-conquer method in time $O(n^2k \log n)$; here we rely on an algorithm of Preparata and Muller [MP] for intersecting two polyhedra in time $O(m \log m)$ where m is the total number of faces. This proves the following.

THEOREM 6. *The k-hull of n points in 3 dimensions can be computed in time $O(n^2 \log n \cdot \min\{n, k \log n\})$. In particular, the center (i.e. the $\lceil n/4 \rceil$-hull) can be found in $O(n^3 \log n)$ time.*

Remark. An interesting consequence of the corollary to Lemma 1 is that n points in 3 dimensions realize at most kn^2 k-sets. Chazelle and Preparata [CP] show that the number of k-sets is also bounded by $O(nk^5)$.

3. On Megiddo's technique: General remarks. Megiddo's technique [M1] has two parts—one is a notion of "parametric search" and the other is a method for exploiting parallel algorithms to construct sequential algorithms. In the next two sections we illustrate these techniques. Our use of his technique is quite striking because first, it is not apparent that our problems are parametrized and second, in some of our algorithms, we have to apply his technique recursively in a way that is quite nontrivial. In order to be able to use his technique we shall need efficient parallel sorting algorithms, in particular, either the n processor, $(\log n)$ depth network of [AKS], or the $O(n \log n)$ processor, $O(\log n)$ parallel time algorithm of Preparata [P].

In general terms, his technique provides a way to search a partially ordered space, of polynomial (or superpolynomial) size, without actually constructing the space (the polynomial may be large). Instead, an *implicit* binary search of the space is made. We use a sorting algorithm to guide this search (Megiddo's technique is not restricted to sorting algorithms). Typically, a sorted order will correspond to a region of the space being searched. So resolving a comparison in the sort corresponds to reducing the size of the space in which a solution is known to lie. As might be expected, a single comparison is expensive; the surprising observation is that several comparisons can be batched relatively cheaply. This leads us to use a parallel sorting algorithm, for we (in our serial algorithm) can batch the comparisons that are carried out simultaneously by the parallel algorithm. A fascinating feature of this implicit search is that it can be applied recursively, each level of the recursion adding some polylogarithmic factor to the complexity. These remarks are most easily understood by considering some examples, such as the problems we solve below.

3.1. Finding a ham sandwich cut. Instead of computing the entire k-hull, some applications require just a single point in it. For k bounded by $n/4$, we can reduce

this simpler problem to the following: given X, a set with n red points and n black points in the plane, find a line (called a "fair cut") which simultaneously separates each of the monochromatic subsets evenly. The existence of a suitable line is guaranteed by the Ham Sandwich theorem. It is easy to see that each fair cut can be associated with some line through two points of X. By checking each pair of points in turn, the desired cut can be found in $O(n^3)$ time. We shall drastically reduce this bound.

To simplify the discussion, we may assume that n is odd and no three points are collinear. A *halver* of a set Y of n points (recall n is odd) is an (oriented) line with at most $(n-1)/2$ points of Y in each of the open half-planes it determines. (Thus a halver is just a $(n+1)/2$-divider as defined in § 2.) For each orientation θ, let L_θ^R be the halver of the red subset of X with orientation θ. Similarly L_θ^B is the halver of the black subset. Let $r(\theta)$ denote the distance that L_θ^B lies to the right of L_θ^R. This distance is negative (resp. zero) if L_θ^B actually lies to the left of (resp. coincides with) L_θ^R. Thus, our goal is to find some (any) orientation θ^* such that $r(\theta^*)$ is zero. Note that a fair cut is easily derived from θ^* in linear time. We note $r(\theta)$ varies continuously with θ.

For any point p and orientation θ in $(0, \pi)$, let $p(\theta)$ denote the projection of p along the direction θ onto the x-axis. For the orientation θ^*, let p^* be used instead of $p(\theta^*)$. Although we do not know the value of θ^*, we shall show how we can sort the points in $X^* = \{p^* : p \in X\}$ (which will allow us to deduce the value of θ^*). Let us inductively suppose that an angular interval Φ contained in $(0, \pi)$ is known such that either Φ is a single angle θ with $r(\theta)$ zero (in which case we are done), or Φ is an open interval (θ_1, θ_2) such that (without loss of generality) $r(\theta_1)$ is positive and $r(\theta_2)$ is negative. (And thus there is a value θ^* in Φ, with $r(\theta^*)$ zero.) Initially, we may take Φ to be $(0, \pi)$ unless $r(0)$ is zero, in which case we are done. Clearly Φ contains some θ with $r(\theta)$ zero. Suppose that in sorting X^*, we compare p_i^* to p_j^* (denoted $p_i : p_j$), where p_i, p_j are points in X. Let $\theta_{i,j}$ denote the angle of the line from p_i to p_j. Assume $0 \leqq \theta_{i,j} < \pi$ (otherwise, take $\theta_{j,i}$ instead). If $\theta_{i,j}$ is outside the interval (θ_1, θ_2) then the relative ordering of p_i^* and p_j^* can be deduced at once. If $r(\theta_{i,j})$ is zero then θ^* is determined. Otherwise let $r(\theta_{i,j})$ be positive (the case $r(\theta_{i,j})$ negative is similar). We refine Φ into $\Phi' = (\theta_{i,j}, \theta_2)$; since the relative ordering of $p_i(\theta)$ and $p_j(\theta)$ is fixed for θ larger than θ_{ij}, and hence for θ in Φ', and as there is a value of θ^* in Φ', we can deduce the relative ordering of p_i^* and p_j^*. Clearly, for any θ, $r(\theta)$ (and hence the comparison $p_i^* : p_j^*$) can be computed in linear time (using a linear time median algorithm [AHU]). It is easy to see that this gives an $O(n^2 \log n)$ time algorithm for computing θ^*. To obtain a faster algorithm, we apply Megiddo's technique.

We use a p processor, h time parallel sorting algorithm to sort X^*. Assume inductively that an angular interval Φ_k ($k = 0, 1, \cdots, h$) containing (some) θ^* is known before the start of the $(k+1)$st step, where Φ_k is either an open interval or a single orientation. (If Φ_k is a single orientation then this orientation is θ^*. So in this case we can halt the execution of the parallel algorithm.) In the $(k+1)$st parallel step of the sorting algorithm, p comparisons of the form $p_i^* : p_j^*$ are made. The p comparisons determine p orientations $\theta_{i,j}$. As above, we may assume that $\theta_{i,j}$ is in the range $(0, \pi)$. For each $\theta_{i,j}$ lying outside the range Φ_k, the outcome of the corresponding comparison is known, as shown in the previous paragraph. For those orientations in Φ_k we first sort them (using any straightforward method) and then do a "binary search" of the sorted orientations. (Actually, we do not have to sort the orientations in Φ_k; it suffices to find the orientations probed by the binary search, for which we use a fast median algorithm [AHU]. Each step of the binary search reduces by half the size of the set to be searched, so we only need $O(p)$ serial steps to find the orientations to be probed.) Each probe of the binary search computes $r(\theta_{i,j})$ for some $\theta_{i,j}$ in Φ_k. Unless $r(\theta_{i,j})$ is

zero (in which case we are done), we refine the interval Φ_k as described in the last paragraph. Each probe takes $O(n)$ time, and $O(\log p)$ probes suffice to determine a subinterval Φ_{k+1} of Φ_k containing θ^* with the property that each of the p comparisons in the parallel step is decided. Therefore each parallel step is accomplished in $O(n \log p + p)$ serial steps. Since there are h parallel steps, the total time is $O((n \log p + p)h)$. If we use the sorting algorithm of Preparata with $O(n \log n)$ processors, and running in $O(\log n)$ parallel time, or the asymptotically more efficient network of [AKS], the total time complexity becomes $O(n \log^2 n)$. For reference, call the serial algorithm just described the *derived* algorithm (with respect to any parallel sorting algorithm).

We now prove the correctness of the derived algorithm. For any orientation θ in $(0, \pi)$, let the *θ-ordering* of X refer to the total order on X induced by the ordering of the projection of X onto the x-axis in the direction θ. (This ordering is well defined because the orientation of the x-axis is 0.) Consider any comparison $p_i^* : p_j^*$ made in the kth parallel step of the parallel sorting algorithm. The outcome of this comparison depends on the relative order of p_i and p_j in the θ^*-ordering. The relative order of p_i and p_j in the θ-ordering is invariant as θ varies in Φ_{k+1}. And this is the outcome we have assigned to the comparison $p_i^* : p_j^*$. Next observe that

$$(*) \qquad (0, \pi) = \Phi_0 \supseteq \Phi_1 \supseteq \cdots \supseteq \Phi_h \qquad (h \leqq O(\log n)).$$

Let T be the θ-ordering on X for some θ in Φ_h. We conclude from $(*)$ that the outcomes of all the comparisons obtained by the (parallel or derived) algorithm are consistent with T; the correctness of the parallel algorithm then implies that the outcomes determine a unique total ordering T' of X; in fact T' equals T. We can say more as follows.

CORRECTNESS LEMMA. *The interval Φ_h consists of a single orientation θ^* with $r(\theta^*) = 0$.*

Proof. Suppose $\Phi_h = (\theta_1, \theta_2)$, $\theta_1 < \theta_2$. Then $r(\theta_1) > 0$, $r(\theta_2) < 0$. Notice that $r(\theta)$ varies continuously with θ. Thus there is some value θ^* such that $r(\theta^*)$ is zero; then for $\theta = \theta^*$ the dividers of the two sets are coincident and thus pass through a red point p and a black point q. So the relative order of $p(\theta)$ and $q(\theta)$ depends on whether θ is larger or smaller than θ^* and thus is not constant over Φ_h. Contradiction. Thus Φ_h is a single orientation. □

Megiddo [M2] showed that under the assumption that the two sets were separated by a straight line then a fair cut could be found in $O(n)$ time.

Remark. Instead of using a sorting algorithm to guide the search, for this problem we can use a parallel median algorithm. Cole and Yap [CY] describe an $M(n) = O((\log \log n)^2)$ time algorithm for finding the median with n processors in the "adaptive" model of Valiant [V]. The method for deriving an algorithm (for finding a fair cut) from parallel median algorithms is exactly the same as above. Again, one parallel step is accomplished in $O(n \log n)$ serial steps. So we deduce that a fair cut for the ham sandwich problem can be found in time $O(n \log n M(n)) = O(n \log n (\log \log n)^2)$ time where $M(n)$ is the parallel time to find the median of n elements using n processors in the Valiant model.

3.2. Finding a point in the center. The algorithm in the last section easily finds a point in the $n/4$-hull (since it can be used to find a $\frac{1}{4}$-partitioner). Unfortunately (and somewhat surprisingly) it does not appear to extend directly to finding a point in the center (i.e. the $\lceil n/3 \rceil$-hull). We now develop such an algorithm. Megiddo's technique will again be exploited (to the hilt).

First we note that it is easy to determine if a given point p is in the center: sort the points of X in angular order about p and then rotate a line about p through a full circle of angles, keeping count of the number of points on the strictly left side of the rotating line. The point p is in the center if the number of points on the left is always at least $\lceil n/3 \rceil$. The time complexity of this procedure is $O(n \log n)$. There are two noteworthy points. First, to define a sorted order we have to start it somewhere—say at the angle 0. Second, we note that if one point lies at angle $\pi + \alpha$ and a second one at angle β, where $0 < \alpha$, $\beta < \pi$, we are really interested in the relative size of α and β when performing the rotation. That is, we want to know the rotational order of the points above the horizontal line through p where the points below this line are represented by their reflection through p; henceforth we shall use the term rotational order for this order.

If we know a point p^* in the center, then the rotational order T^* of the points in X about p^* is easily determined. We show how to compute (some) T^* without knowing (any such) p^*, using an m processor parallel sorting algorithm. What does "comparing" a pair of points p_i and p_j in X mean, relative to the ordering T^*? The relative rotational order of p_i and p_j about a point q depends on which of six regions q lies in (see Fig. 2). The regions are defined by the line L_{ij} through p_i and p_j, and by the horizontal lines through p_i and p_j. Note that the need for the two horizontal lines through p_i, p_j arise because the relative order of p_i and p_j in the rotational order is partly a function of which side of these lines p^* lies. So to resolve the comparison we determine if the center intersects any of these lines and if not in which region the center lies. To do this we use an algorithm which given a line L determines if the center intersects L, and if so gives a point on L in the center, and if not determines on which side of L the center lies. This algorithm runs in time $O(n \log^3 n)$. The idea is similar to the ham sandwich cut algorithm; we describe it following the main algorithm.

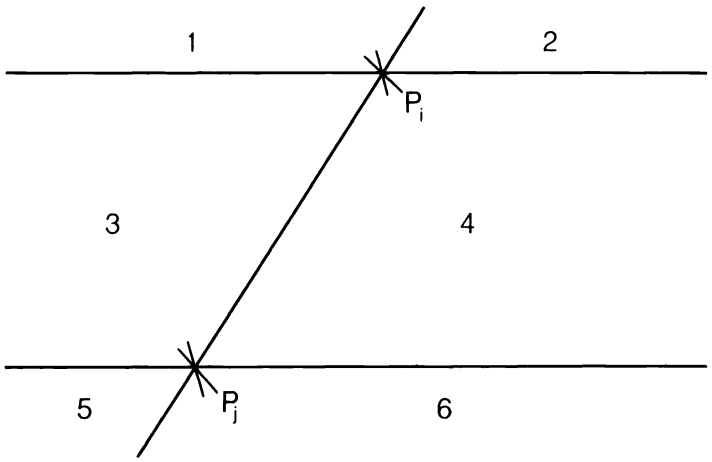


FIG. 2

To eliminate the consideration of the horizontal lines when performing comparisons between pairs of points we first either determine between which pair of horizontal lines the center lies, or we find a point in the center on one of these lines. We do this by performing a binary search of the n horizontal lines, using the algorithm mentioned in the last paragraph; we use a total of $O(n \log^4 n)$ time.

The crucial idea is how to "batch process" m of the comparisons between pairs p_i and p_j, performed during one parallel step of the sorting. The m comparisons give rise to m lines of the form $L_{i,j}$. For each of these lines we want to determine on which side p^* lies. This can be solved by using another technique of Megiddo's: the technique for linear programming in fixed dimension [M3].

We show that in $O(n \log^3 n)$ time either we determine for at least one eighth of the lines on which side p^* lies, or we find p^*; also, in the former case, we will have found two lines bounding the region in which p^* lies. By iterating $O(\log m)$ times, either we determine for every line on which side p^* lies, or we find p^*; also, in the former case, we will have determined $O(\log m)$ lines bounding the region in which p^* lies. To eliminate one eighth of the lines we proceed as follows. We find the line of median slope (among the m lines $L_{i,j}$). Suppose it has slope s. We then pair the lines, so that each pair contains a line of slope less than s and a line of slope greater than s. For each pair, we compute the intersection point of the two lines in the pair. We find x_{med}, the median of the x-coordinates of the intersection points, and determine whether p^* lies on, to the left of, or to the right of the vertical line L through x_{med}. If p^* lies on L we are done, so assume without loss of generality that p^* lies to the left of L. We then find the line K of slope s which evenly divides the collection of lines of slope s consisting of those lines of slope s in the original set of lines plus the lines of slope s through the intersection points to the right of L. We determine whether p^* is on, to the left of, or to the right of K. If p^* is on K we are done. Otherwise we have determined that p^* lies in a quadrant through which at least one eighth of the original set of lines do not pass (since for every intersection point in the opposite quadrant, one of the two lines through the intersection point cannot pass through the quandrant in which p^* lies). This method for eliminating lines is the same as one given in [M3]. It is illustrated in Fig. 3.

Hence to find the rotational order of the points about p^* (using Preparata's algorithm or the [AKS] network) takes time $O(n \log^5 n)$. At the end of this process we have either found a point p^*, or a bounded region R, with respect to which the rotational order of the points in X is fixed. The region R is the intersection of $O(\log n)$ regions, one for each parallel step of the point sorting algorithm. Each such region is bounded by $O(\log m) = O(\log n)$ lines.

Remark. This algorithm is an interesting variant of Megiddo's technique: to bound the number of iterations, a proportional number of lines is eliminated at each iteration; however, the time needed for one iteration does not decrease geometrically.

CORRECTNESS LEMMA. *The center contains R.*

Proof. From the existence of the center and the form of the algorithm it is clear that R contains a point in the center. Because the rotational order of the points is the same about any point in R, either every point in R is in the center or none of R is in the center, and plainly only the first case is possible. □

It remains to give an algorithm to determine whether the center intersects a given line L, or on which side of L it lies, where for convenience L is imagined to be the x-axis. For each orientation θ, $-\pi < \theta \leqq \pi$, and point x, let $l_\theta(x)$ denote the directed line through x and orientation θ. Define $f_\theta(x)$ to be the fraction of the points of X which lie on the closed half-plane to the right of $l_\theta(x)$ and also define $g(x) = \min_\theta f_\theta(x)$, $g_1(x) = \min_{0<\theta\leqq\pi} f_\theta(x)$, and $g_2(x) = \min_{-\pi<\theta\leqq 0} f_\theta(x)$. We state some simple properties of f, g, g_1, and g_2.

LEMMA 2. (i) *Let L be any line (imagined as the x-axis). Then for each θ, the function $f_\theta(x)$ is monotone as x varies along L. It is constant for $\theta = 0$ or π, decreasing*

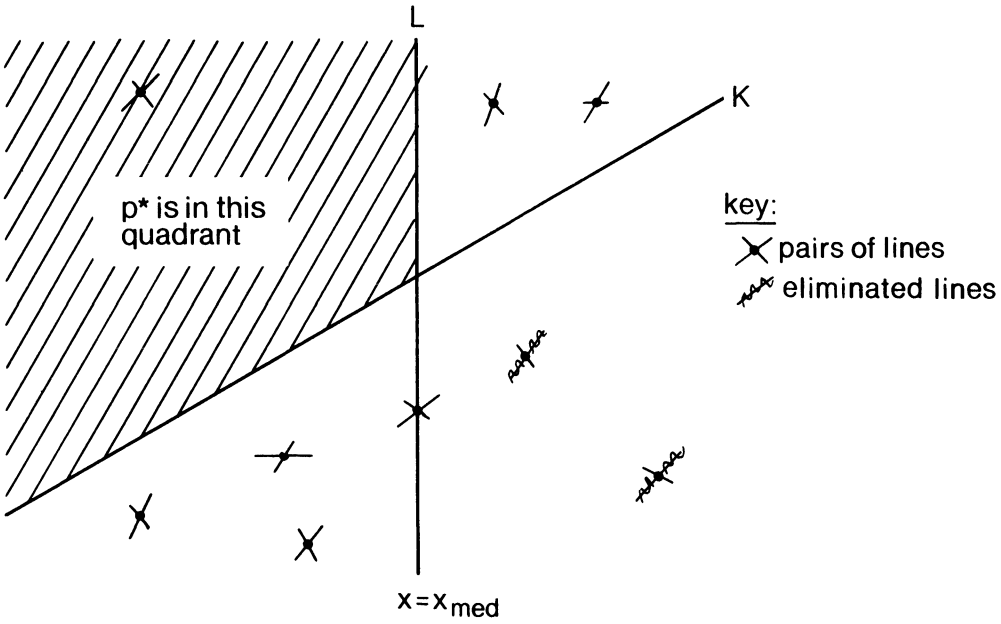


FIG. 3

from 1 *to* 0 *for* $\theta \in (0, \pi)$ (*the "upward" orientations*), *and increasing from* 0 *to* 1 *for* $\theta \in (-\pi, 0)$ (*the "downward" orientations*).

(ii) $g_1(x)$ *is a decreasing function on* L *and* $g_2(x)$ *is an increasing function on* L. *Let* I *be the interval of* L *over which* $g_1(x) = g_2(x)$. *Then* $g(x)$ *is a unimodal function attaining its maximum at* $x \in I$.

(iii) *For any* x, $g(x)$ *attains its minimum at some set* $\Theta(x)$ *of orientations. Suppose* x *is not in the center. If* $\Theta(x)$ *consists only of upward* (*resp. downward*) *orientations then* x *lies to the right* (*resp. left*) *of* I, *and conversely. Otherwise* x *lies in* I *and* $\Theta(x)$ *has both types of orientations.*

Proof. The proof of (i) is immediate and (ii) follows since the minimum of a set of monotone increasing functions is also monotone increasing. Finally, (iii) holds because if x is to the right (resp. left) of I then $g(x) = g_2(x)$ (resp. $g(x) = g_1(x)$) and if $x \in I$ then $g(x) = g_1(x) = g_2(x)$. □

We seek a point p^* in I. To guide the search for p^* we find the rotational order of the points in X about p^*, using an m processor algorithm, running in parallel time h. A comparison of two points p_i and p_j, as described above, determines a line L_{ij}; it intersects L in a point, x say. The result of the comparison depends solely on which side of x the point p^* lies. We determine whether I includes x (in which case we are done), or if I is to the left or right of x, by rotating a line about x and computing $f_\theta(x)$ as θ varies; using (iii) above we deduce where x lies. This takes time $O(n \log n)$. To batch m of these comparisons, we sort the resulting m points on L and perform a binary search among them (actually, find medians etc.). This yields an interval J (containing p^*) over which the comparisons performed so far have a fixed result. This takes time $O(n \log n \log m + m)$. Hence to find the rotational order of the points about p^* takes time $O((n \log n \log m + m)h)$. Using either Preparata's algorithm ($m = n \log n$, $h = \log n$) or the [AKS] network ($m = n$, $h = \log n$) this becomes a time of $O(n \log^3 n)$.

The algorithm finds either p^*, or an open subinterval J of I with respect to which the rotational order of the points in X is fixed. But any point in J will serve as p^* so

in all cases this algorithm finds p^*. If p^* is not in the center, we show how to use this algorithm to decide on which side of L the center lies. If p^* is not in the center, $g(p^*)$ is less than one third. Observe that this implies that there exist two orientations, an upward θ_1 and a downward θ_2, such that $f_{\theta_1}(p^*) = f_{\theta_2}(p^*) < \frac{1}{3}$. See Fig. 4.

It follows that the center of S must lie in the shaded area of the Fig. 4 (the side of L that this shaded area lies on depends on θ_i), because for any point not in this region either the line with orientation θ_1, or the one with orientation θ_2 is guaranteed to have fewer than $n/3$ points to its right. So if L does not pass through the center of X, the procedure for finding p^* also tells us on which side of L the actual center of X lies.

We have shown

THEOREM 7. *A point in the center of n points can be found in time* $O(n \log^5 n)$.

Remark. This algorithm can be used to find a point in the maximal center (a nontrivial k-hull such that k is maximum). We just have to make minor changes to the algorithm for deciding on which side of a line L the (maximal) center lies. To be precise, suppose we have found p^* such that $g(p^*) = g_1(p^*) = g_2(p^*) = h$. This implies there exist two orientations, an upward θ_1 and a downward θ_2, such that $f_{\theta_1}(p^*) = f_{\theta_2}(p^*) = h$. We deduce the maximal center (a k-hull with $k \geqq h$) can only be to one side of L (see Fig. 4); if there are many such orientations we may deduce that the maximal center cannot be to either side of L, in which case p^* is the maximal center.

A much more complex version of these ideas gives us an algorithm for determining a point in the center in 3 dimensions in time $O(n^2 \log^7 n)$. The method probably can be extended to any dimension.

4. An application: Intersecting lines and points. Hopcroft posed the following problem which arose in the context of collision detection in robotics: given a set Y of n lines and a set X of n points, determine if any point lies in any line. Hopcroft and Seidel gave a solution running in time $O(n^{1.5} \log n)$ [Ho]. We first sketch a solution which takes time $O(n^2)$. Based on this solution, we shall obtain further improvements

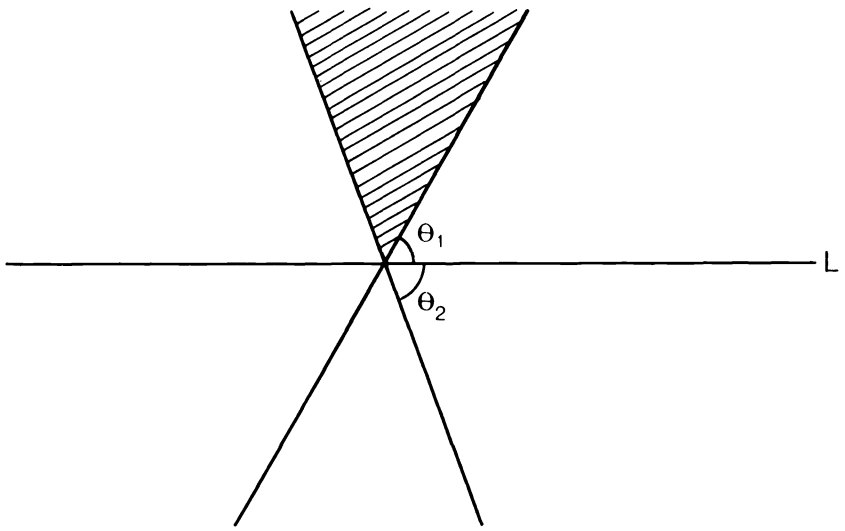


FIG. 4

by using $\frac{1}{4}$-partitioners. This result is not intended to be practical, but does serve to illustrate the use of partitioners in divide and conquer algorithms.

For the first solution, we dualize the problem, so each point is replaced by a line, and each line by a point. We construct the arrangement defined by the n points (which are now lines) in time $O(n^2)$ [EOS]. In a further $O(n^2)$ time we construct a data structure for supporting point location queries on this arrangement [EGS]. For each line (now a point) we perform a point location (in $O(\log n)$ time). A point (now a line) and a line (now a point) intersect if and only if the dual objects intersect. Putting everything together, we see we use $O(n^2+n\log n)$ time. More generally, if we have m lines this solution uses $O(n^2+m\log n)$ time.

We now improve the $O(n^2)$ bound. The basic idea is illustrated as follows: suppose we have to check the intersection of a set Y of lines with a set X of points. Let X_i $(i=1,\cdots,4)$ be subsets of X determined by a $\frac{1}{4}$-partition. Note that any line intersects at most 3 of the quadrants of the $\frac{1}{4}$-partition. The original problem is thus reduced to four subproblems of checking intersections of subsets Y_i of Y with X_i $(i=1,\cdots,4)$, where Y_i consists of the lines which intersect the quadrant containing X_i. Furthermore, $\sum_{i=1}^{4}|Y_i|\leqq 3|Y|$. The recursive application of this idea will yield a time bound of $O(n^c\log n)$ for some $c<1.5$. To get the best constant for c, we build a search tree based on the Conjugation Tree of Edelsbrunner and Welzl [EW3].

Each internal node in the search tree has two children. Each node u is associated with a convex polygonal region R_u of the plane, and at u we store a line L_u which bisects the set $X_u=X\cap R_u$. The two children of internal node u are associated with the subregions of R_u defined by L_u. It is required that the lines stored at the children of u be the same line (K_u, say), for each internal node u. That is, L_u and K_u form a $\frac{1}{4}$-partitioner of the set X_u. We call the four subregions of R_u defined by L_u and K_u the *quadrants* of u. The root is associated with the whole plane. The height of the search tree is chosen to be $\log n-h$ (h is specified below) and thus each leaf is associated with a subset of X of size $O(2^h)$. Note that each line intersects $O((n/2^h)^\alpha)$ regions associated with leaves, where $\alpha=\log((1+\sqrt{5})/2)(O(n^\alpha)$ is the solution to the recurrence equation $C(n)=C(n/2)+C(n/4)$, for $n>1$, and $C(1)=1$ [EW3]). The cost of finding the lines K_u at all the nodes u of the search tree is seen to be $O(n\log n)$ using Megiddo's linear time algorithm for partitioning [M2].

The "search" for intersections of lines in Y with points in X proceeds as follows: initially, we are at the root and want to check if any line in Y contains any point in X. In general, suppose we are some node u and want to check if any line in some subset Y_u of Y contains any point in the subset X_u of X associated with u. If u is a leaf then $|X_u|=O(2^h)$ and we solve this problem directly by the method described above in time $O(2^{2h}+|Y_u|\log n)$. If the children, v and w, of u are leaves then we add Y_u to each of Y_v and Y_w; we then recursively solve the problem at nodes v and w. Otherwise, for each line L in Y_u we proceed as follows. We determine which quadrant of u (defined by L_u and K_u) is not intersected by L; without loss of generality suppose this quadrant is a child of the left child of u. Let v and w be the left and right children of u, and let x and z be the children of v, x associated with the quadrant not intersected by L. Then we add L to each of Y_w and Y_z. After distributing all the lines in Y_u, we recursively solve the problem at nodes v and w.

We now assess the complexity of this procedure. Let U be the set of leaves in the search tree which were visited. For u in U, let n_u denote the size of the set Y_u of lines to be checked at u. We observe $\sum_{u\in U}n_u=O(n\cdot(n/2^h)^\alpha)$, and $|U|=O(n/2^h)$. The work done at the internal nodes is $O(\sum_{u\in U}n_u)$. The work done at the leaves is $O(\sum_{u\in U}(2^{2h}+n_u\log n))$. So the total work done is $O(n2^h+n(n/2^h)^\alpha\log n)$, which is

minimized by setting $(2^h)^{1+\alpha} = n^\alpha$. We find the total work done is $O(n^{1+\alpha/(1+\alpha)} \log n) = O(n^{1.41})$.

This solution was independently discovered by Chazelle.

5. Conclusions. The general contribution of this paper is to introduce a natural generalization of the concepts of convex hulls and centers, and to investigate a number of computational problems surrounding it. The rich connection between k-hulls and the previously studied concepts of k-belts, k-sets, α-partitioners, and the classical Ham Sandwich theorem is particularly noteworthy, and no doubt sheds some insights into these concepts.

The algorithms presented here develop, or at least make more accessible, some powerful algorithmic techniques which are quite new in computational geometry. Specifically, we have extended the domain of applications for Megiddo's technique. Recently, an improvement of Megiddo's technique has been discovered [Co], which in many instances yields faster algorithms than the original technique. In particular, it provides faster algorithms for finding a ham sandwich cut in the plane, and for finding a point in the k-hull. We expect to find further new applications for these techniques.

Finally, our "parametric searching" algorithms are significantly faster than previous solutions. This in turn translates into efficient algorithms for a number of applications. Again, we expect other applications to be found.

Acknowledgments. We are indebted to R. Pollack for references [L], [ELSS] and for pointing out that our algorithm in § 2 had previously been discovered by Lovász. We are also grateful to N. Megiddo for suggesting that an earlier version of our algorithm for finding a point in the k-hull could be improved. We would also like to thank one of the referees for the many detailed comments which markedly improved the paper.

REFERENCES

[Ag] M. K. AGOSTON, *Algebraic Topology*, Marcel Dekker, New York, 1976.

[AHU] A. V. AHO, J. HOPCROFT AND J. D. ULLMAN. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

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

[C] B. CHAZELLE, *Optimal algorithms for computing depths and layers*, Brown Univ. Tech. Report CS-83-13, March 1983.

[CP] B. CHAZELLE AND F. PREPARATA, *Halfspace range search: an algorithmic application of k-sets*, Proc. of the ACM Symp. on Comp. Geom., 1985, pp. 107-115.

[Co] R. COLE, *Slowing down sorting networks to obtain faster sorting algorithms*, J. Assoc. Comput. Mach., to appear.

[CY] R. COLE AND C. K. YAP, *A parallel median algorithm*, Inform. Process. Lett., 20 (1983), pp. 137-139.

[DS] L. DUBINS AND E. SPANIER, *How to cut a cake fairly*, Amer. Math. Monthly, 68 (1961), pp. 1-17.

[EGS] H. EDELSBRUNNER, L. GUIBAS AND J. STOLFI, *Optimal point location in a monotone subdivision*, this Journal, 15 (1986), pp. 317-340.

[EOS] H. EDELSBRUNNER, J. O'ROURKE AND R. SEIDEL, *Constructing arrangements of lines and hyperplanes with applications*, Proceedings, 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 83-91.

[EW1] H. EDELSBRUNNER AND E. WELZL, *On the number of line separations of a finite set in the plane*, J. Comb. Theory, A, 38, 1 (1985), pp. 15-29.

[EW2] ———, *Constructing belts in two-dimensional arrangements with applications*, this Journal, 15 (1986), pp. 271-284.

[EW3] ———, *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Report F111, Inst. Inf. Proc., Tech. Univ. Graz, Austria (1983).

[ELSS] P. ERDÖS, L. LOVÁSZ, A. SIMMONS AND E. G. STRAUS, *Dissection graphs of planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava et al., eds., North-Holland, Amsterdam, 1973, pp. 139-149.

[H] T. P. HILL, *Determining a fair border*, Amer. Math. Monthly, 90:7 (1983), pp. 438-442.

[Ho] J. HOPCROFT, private communication.

[L] L. LOVÁSZ, *On the number of halving lines*, Ann. Univ. Sci. Budapest, Eötvos Sect. Mat., 14 (1971), pp. 107-108.

[M1] N. MEGIDDO, *Applying parallel computation algorithms in the design of serial algorithms*, J. Assoc. Comput. Mach., 30 (1983), pp. 852-865.

[M2] ———, *Partitioning with two lines in the plane*, J. Algorithms, to appear.

[M3] ———, *Linear time algorithms for linear programming in R^3 and related problems*, this Journal, 12 (1983), pp. 759-776.

[MP] D. MULLER AND F. PREPARATA, *Finding the intersection of two convex polyhedra*, Theoret. Comput. Sci., 7 (1979), pp. 217-236.

[OVL] M. H. OVERMARS AND J. VAN LEEUWEN, *Maintenance of configurations in the plane*, J. Comput. System Sci., 23 (1981), pp. 166-204.

[P] F. PREPARATA, *New parallel sorting schemes*, IEEE Trans. Comput., C-27 (1978), pp. 669-673.

[V] L. VALIANT, *Parallelism in comparison problems*, this Journal, 4 (1975), pp. 348-355.

[W] D. WILLARD, *Polygon retrieval*, this Journal, 11 (1982), pp. 149-165.

[YB] I. M. YAGLOM AND V. G. BOLTYANSKII, *Convex Figures*, (trans.) Holt, Rinehart and Winston, New York, 1961.

[Y] F. YAO, *A 3-Space partition and its application*, Proceedings, 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 258-263.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
006

POWER DIAGRAMS: PROPERTIES, ALGORITHMS AND APPLICATIONS*

F. AURENHAMMER†

Abstract. The power $\text{pow}(x, s)$ of a point x with respect to a sphere s in Euclidean d-space E^d is given by $d^2(x, z) - r^2$, where d denotes the Euclidean distance function, and z and r are the center and the radius of s. The power diagram of a finite set S of spheres in E^d is a cell complex that associates each $s \in S$ with the convex domain $\{x \in E^d \mid \text{pow}(x, s) < \text{pow}(x, t), \text{ for all } t \in S - \{s\}\}$.

The close relationship to convex hulls and arrangements of hyperplanes is investigated and exploited. Efficient algorithms that compute the power diagram and its order-k modifications are obtained. Among the applications of these results are algorithms for detecting k-sets, for union and intersection problems for cones and paraboloids, and for constructing weighted Voronoi diagrams and Voronoi diagrams for spheres. Upper space bounds for these geometric problems are derived.

Key words. Voronoi diagrams, Laguerre metric, cell complexes, convex hulls, hyperplane arrangements, duality, efficient algorithms, concrete complexity

AMS(MOS) subject classifications. F.2.3, E.2.2

1. Introduction.

Voronoi diagrams for point-sets in d-dimensional Euclidean space E^d have been studied by a number of people in their original as well as in generalized settings. For a finite set $M \subsetneq E^d$, the (*closest-point*) *Voronoi diagram* of M associates each $p \in M$ with the convex region $R(p)$ of all points closer to p than to any other point in M. More formally, $R(p) = \{x \in E^d \mid d(x, p) < d(x, q), \forall q \in M - \{p\}\}$, where d denotes the Euclidean distance function. Voronoi diagrams are of importance in a variety of areas other than computer science whose enumeration exceeds the scope of this paper. Shamos and Hoey [35] were the first to introduce the planar diagram to computational geometry and also demonstrated how to construct it efficiently. Using a dual correspondence to convex hulls discovered by Brown [7], its higher-dimensional analogues can be obtained using methods in Seidel [32].

As the variety of applications of the Voronoi diagram was recognized, people soon became aware of the fact that many practical situations are better described by some modification than by the original diagram. Following these requirements, diagrams under more general metrics [21], [23], for more general objects than points [19], [9], and of higher order [35], [22], [10] have been investigated. A different generalization stems from the concept of weighting the given points. Each $p \in M$ has assigned an individual real number $w(p)$, the *weight* of p, and the distance of a point $x \in E^d$ is measured as a (suitable) function of $d(x, p)$ and $w(p)$. The diagrams induced by the distance function $d(x, p)/w(p)$ are studied in [3] and [2] for $M \subsetneq E^2$ and $M \subsetneq E^1$. The formula $d(x, p) - w(p)$ leads to diagrams whose planar instances can be interpreted as Voronoi diagrams for circles with centers p and radii $w(p)$, see e.g. [9]. The practical relevance of these weighting schemes is reported in [5] and [17] which describe applications in geography and economics.

This paper is concerned with the distance function $d(x, p)^2 - w(p)$ and the resulting diagrams in E^d. Again, this particular weighting allows a nice geometric interpretation: Viewing a point p with weight $w(p) \geqq 0$ as a sphere $s = \{x \in E^d \mid d(x, p) = \sqrt{w(p)}\}$, the above distance function is known as the *power* of a point $x \in E^d$ with respect to s, abbreviated $\text{pow}(x, s)$. Let S denote a finite set of spheres in E^d. For $s \in S$, we call

* Received by the editors April 19, 1984, and in final revised form December 13, 1985.

† Institutes for Information Processing, Technical University of Graz and Austrian Computer Society, Schiesstattgasse 4a, A-8010 Graz, Austria.

the set cell $(s) = \{x \in E^d \mid \text{pow}(x, s) < \text{pow}(x, t), \forall t \in S - \{s\}\}$ the *power cell* of s and the collection of all cell (s), for $s \in S$, the *power diagram* of S, or PD (S) for short. (Also, *Laguerre diagram* or *Dirichlet cell complex* are used synonymously for PD (S).)

Power diagrams have been studied for a long time in mathematics and other areas of science. The power function was mentioned as a generalized distance function by Laguerre (see e.g. [4]) and by Voronoi [36]. PDs play an important role in packing and covering spheres [31], [14], and in illuminating balls [15], [25], and are applicable to particular problems in number theory [24]. Independently of the author, the equivalence of PDs and the projections of the boundaries of convex polyhedra is shown in [28]. This equivalence implies a dual correspondence between PDs and polytopes. As a matter of fact, the PD for congruent spheres is the Voronoi diagram of their centers. References [26] and [33] prove that there exist Voronoi diagrams dual to polytopes which realize a maximal number of facets. Practical applications of PDs can be found in [20] (crystallography), [34] (metallurgy), and [17] (economics).

Surprisingly, power diagrams (in their general form) have not yet received attention in computational geometry.[1] We shall show that PDs in E^d, for $d \geqq 1$, can be constructed using known algorithms, and are applicable to seemingly unrelated problems. In fact, the contributions of this paper are of algorithmic as well as of mathematical nature. Properties of PDs and their generalizations to higher order are described in §§ 2 and 3. In § 4, the relationship between arbitrary PDs in E^d and convex hulls and arrangements of hyperplanes in E^{d+1} is investigated by exploiting a known geometric transform. Efficient algorithms that compute PDs, PDs of higher order, and particular power cells are outlined in § 5. Based on these results, computational solutions of several geometric problems are presented and their space complexity is analyzed. Finally, conclusions and open problems are given in § 7.

2. Analyzing power diagrams. In this section we discuss several properties of power diagrams in E^2 and E^d, for $d \geqq 3$. While § 2.1 introduces some notations, § 2.2 is mainly a review of known geometrical facts that provide the basis for our investigations in §§ 3 to 6.

2.1. Geometrical notations. We first recall the generalizations of lines, circles, and polygons in higher dimensions.

Let $p \in E^d$ be different from the origin and have positive dth coordinate, and let c be real. Then $h = \{x \in E^d \mid xp = c\}$ is a *hyperplane* and $h^+ = \{x \in E^d \mid xp > c\}$ and $h^- = \{x \in E^d \mid xp < c\}$ is an *upper* and a *lower* (open) *halfspace* in E^d, respectively (xp denotes the inner product of x and p). $q \in E^d$ lies *above* (*in*, *below*) h if $q \in h^+$ ($q \in h$, $q \in h^-$). A *j-flat* is a set expressible as the intersection of $d-j$, but no fewer hyperplanes. For example, lines (planes) are hyperplanes in E^2 (E^3). In E^3, there exist 0-flats (points), 1-flats (lines), 2-flats (planes), and a 3-flat (E^3 itself).

For $p \in E^d$ and $c \geqq 0$, $\{x \in E^d \mid d(x, p) = c\}$ is a *sphere* and $\{x \in E^d \mid d(x, p) < c\}$ is an (open) *ball* in E^d with *center* p and *radius* c. Two spheres are *co-centric* (*congruent*) if they have the same center (radius).

A *polyhedron* f in E^d is the nonempty intersection of a finite number of halfspaces. We call f a *j-polyhedron* if there is a j-flat, but no $(j-1)$-flat that contains f. Examples for 0-, 1-, 2- and 3-polyhedra are points, line segments, polygons and polyhedra in E^3.

A j-polyhedron f is a (relatively) open convex set and its boundary consists of a finite number of i-polyhedra ($0 \leqq i < j \leqq d$), also called *i-faces* of f. We use the terms

[1] The author recently learned that Imai, Iri and Murota have outlined a divide-and-conquer algorithm for planar PDs.

vertex, *edge* and *facet* for 0-, 1- and $(j-1)$-face, respectively. Two polyhedra f and g are *incident* if f is a facet of g, and *adjacent* if they are incident to the same facet. f is *bounded* (f is a *polytope*) if there is some ball that contains f. The *convex hull* of a finite point-set M in E^d is defined as the intersection of all halfspaces containing M and thus is a polytope. Two d-polyhedra P and Q are *dual* if there is a one-to-one correspondence ψ between the j-faces of P and the $(d-j-1)$-faces of $Q (0 \leqq j \leqq d-1)$ such that $f \subseteq g$, for any two faces f and g of P, exactly if $\psi(f) \supseteq \psi(g)$. For an extensive treatment of polyhedra, the reader is referred to [16] or to [6].

A *cell complex* C in E^d is a partition of E^d into finitely many j-polyhedra, the *j-faces* of $C (0 \leqq j \leqq d)$. d-faces of C will be also called *cells*. Let C be contained in some hyperplane of E^{d+1}. C and a $(d+1)$-polyhedron P are said to be *affinely equivalent* if there exists a central or parallel projection φ such that, for each face f of C, $f = \varphi(g)$ holds for some face g of P. Cell complexes which are affinely equivalent to some polyhedron will be called *polytopal*. Clearly, affine equivalence between C and P implies *combinatorial equivalence*, i.e. the graphs induced by the edges and vertices of C and P are isomorphic, but the reverse, in general, is not true.

2.2. Basic observations on power diagrams. We continue with a description of basic properties of power diagrams in E^d.

Recall that the power of a point x with respect to a sphere $s \subsetneq E^d$ with center z and radius r is defined as $\text{pow}(x, s) = d^2(x, z) - r^2$. Thus $\text{pow}(x, s) < 0$ if x lies in the ball bounded by s, $\text{pow}(x, s) = 0$ if $x \in s$, and $\text{pow}(x, s) > 0$ otherwise. In the last case, the Pythagorean Theorem implies that $\text{pow}(x, s)$ equals the square of the distance of x to the touching point of a line tangent to s and through x. A fundamental property is expressed in the following observation.

Observation 1. Let s and t be spheres in E^d with centers $z_s \neq z_t$ and radii r_s and r_t. The points x satisfying $\text{pow}(x, s) = \text{pow}(x, t)$ describe the hyperplane

$$h: \quad 2x(z_s - z_t) = r_t^2 - r_s^2 - z_t^2 + z_s^2$$

which is perpendicular to the line joining z_s and z_t.

For circles s and t, h is known as the *power line*, *radical axis*, or *chordale*. Accordingly, we will term h the *chordale of s and t*, or $\text{chor}(s, t)$ for short. Note that $\text{chor}(s, t)$ is not defined for $z_s = z_t$. It seems reasonable to assume the chordale of two co-centric spheres to be at infinity. For $z_s \neq z_t$, we have the following nice properties: If $s \cap t \neq \emptyset$ then $s \cap t \subsetneq \text{chor}(s, t)$. Otherwise, s and t are contained in the same open halfspace bounded by $\text{chor}(s, t)$ if and only if s encloses (or is enclosed in) t. If $r_s = r_t$ then $\text{chor}(s, t)$ is the perpendicular bisector of the line joining z_s and z_t.

It is well known that the chordales defined by three circles whose centers are not collinear intersect in a common point. This fact generalizes nicely to higher dimensions.

Observation 2. Let s, t and u be spheres in E^d. If z_s, z_t and z_u are not collinear then $f = \text{chor}(s, t) \cap \text{chor}(s, u) \cap \text{chor}(t, u)$ is a $(d-2)$-flat. Otherwise, the three chordales are parallel.

f is termed the *radical center* of s, t and u. In contrast to three chordales, three hyperplanes in E^d intersect, in general, in a common $(d-3)$-flat.

Now let S be a finite set of spheres in E^d. By definition the power cell, $\text{cell}(s)$ of $s \in S$, is the intersection of $n-1$ halfspaces bounded by chordales (Observation 1) and hence is a d-polyhedron with at most $n-1$ facets. Obviously, $\text{cell}(s) \cap \text{cell}(t) = \emptyset$ for $s, t \in S$, $s \neq t$, and there is some $s \in S$ which minimizes $\text{pow}(x, s)$ for each $x \in E^d$. This implies that $\text{PD}(S)$ is a partition of E^d into polyhedra (namely the power cells and the lower-dimensional faces bounding them), that is, $\text{PD}(S)$ is a cell complex in E^d. Figure 1 illustrates these observations by depicting the PD of 6 circles in E^2. The

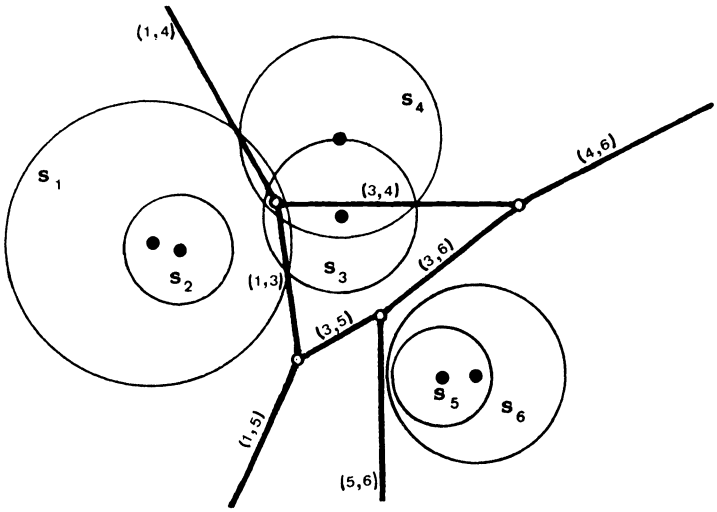


FIG. 1. *Power diagram for 6 circles.*

plane is partitioned into 5 cells, 8 edges and 4 vertices. Each edge e is labelled by (i,j) if e separates cell (s_i) and cell (s_j). Note the possible occurrence of empty power cells if a sphere is contained in the union of the balls bounded by the remaining spheres (for example, see circle s_2). However, circle s_5 shows that this condition does not suffice. It should be mentioned here that PDs are the generalized Voronoi diagrams that have the strongest similarities to the original diagrams.

Next we concentrate on the *size* of a PD for $n \geqq 3$ spheres in E^d, i.e., the number of its j-faces, for $0 \leqq j \leqq d$.

For $S \subsetneq E^2$, PD(S) can be interpreted as a *planar graph* consisting of *regions, edges,* and *vertices.* Each vertex is in the closure of at least three edges (Observation 2) and the number of regions does not exceed n. It is easy to verify that, in a PD(S) of maximal size, there are n regions and exactly three edges emanating from each vertex. Thus a dual graph of PD(S) is a triangulation on n vertices. (A *dual graph* $D(G)$ of a graph G contains exactly one vertex for each region of G. Two vertices of $D(G)$ are connected by an edge if and only if the boundaries of the corresponding regions of G have an edge in common. If S degenerates to a set of n points and $D(\text{PD}(S))$ has straight edges then $D(\text{PD}(S))$ is known as the *Delaunay triangulation* of S.) But a triangulation on n vertices cannot have more than $2n-4$ regions (including the only not simply-connected region which does not give rise to a vertex of PD(S) here) and $3n-6$ edges; consult e.g. [18]. We conclude as follows.

LEMMA 1. *Let S be a set of $n \geqq 3$ circles in E^2. Then* PD(S) *consists of at most n cells, $3n-6$ edges, and $2n-5$ vertices.*

Let us return to the d-dimensional case. The following assertion results from the affine equivalence of PDs in E^d and $(d+1)$-polyhedra established in Theorem 4. The formula on the size of a polyhedron is taken from [6]. McMullen [27] succeeded in proving its maximality (upper bound theorem).

THEOREM 1. *Let S be a set of $n > d > 0$ spheres in E^d, and let f_j denote the maximal number of j-faces of a $(d+1)$-polyhedron with n facets. Then*

$$f_j = \sum_{i=0}^{a} \binom{i}{j}\binom{n-d+i-2}{i} + \sum_{i=0}^{b} \binom{d-i+1}{j}\binom{n-d+i-2}{i}$$

for $a = \lceil d/2 \rceil$ and $b = \lfloor d/2 \rfloor$

(f_j *is in* $O(n^{\lceil d/2 \rceil})$), *and* PD(S) *contains at most n cells,* f_j *j-faces, for* $1 \leqq j \leqq d-1$, *and* $f_0 - 1$ *vertices. These bounds are tight.*

The diagram induced by 7 spheres in E^3 is shown in Fig. 2. For the sake of clearness, only the (parts of) faces above the plane that intersects the spheres in the dashed circles are drawn.

3. Power diagrams of higher order. Power diagrams are generalized in this section to higher order. Let S be a set of n spheres in E^d and let $T \subsetneq S$. The *power cell* of T, abbreviated cell (T), is defined as $\{x \in E^d \mid \text{pow}(x, t) < \text{pow}(x, s), \forall t \in T, \forall s \in S - T\}$. Clearly, an arbitrary T may give rise to an empty cell (T). The set of nonempty cells, defined by subsets of S with cardinality k, is termed the *order-k power diagram* of S, or k-PD(S) for short. This extension to order k is analogous to the concept of order-k Voronoi diagrams first considered in [35] which represent the special case of k-PDs for congruent spheres.

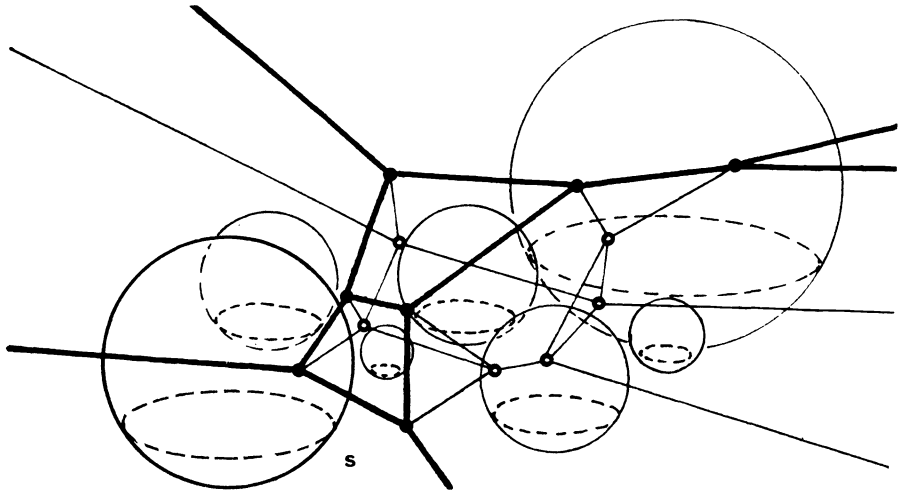


FIG. 2. *View of* PD (S) *from* cell (s).

In the following, several properties of k-PDs are analyzed. Our main interest is to determine the size of these diagrams. As easily seen, the cells of k-PD(S) are d-polyhedra which (together with their bounding faces) define a cell complex in E^d. Trivially, 1-PD(S) = PD(S) holds. Concerning $(n-1)$-PD(S), each cell Z belongs to $n-1$ spheres, and the power of the points in Z w.r.t. the nth sphere s is maximal. Z is named the *maximal power cell* of s, for short $\text{cell}_m(s)$, and $(n-1)$-PD(S) is the *maximal power diagram* of S, for short $\text{PD}_m(S)$. If we generalize $\text{PD}_m(S)$ to order k then obviously k-$\text{PD}_m(S) = (n-k)$-PD(S) holds for $k = 1, \cdots, n-1$. PD_m for congruent spheres is the *farthest-point Voronoi diagram* [35] for their centers.

THEOREM 2. *Let S be a set of n spheres in* E^d. *The maximal number of j-faces of a* 1-PD *with n cells is a tight upper bound for the number of j-faces of* $\text{PD}_m(S)$ $(0 \leqq j \leqq d)$.

This assertion is a consequence of the identity of k-PDs and $(n-k)$-PDs (see Lemma 4). We next establish bounds on the number of faces attained by all power diagrams of higher order of a finite set of spheres. The result relies on a correspondence to arrangements of hyperplanes and is proved in § 4.3.

THEOREM 3. *For a set S of n spheres in* E^d, *the total number of j-faces* $(0 \leqq j \leqq d)$ *of* 1-PD(S), $\cdots$, $(n-1)$-PD(S) *is in* $\Theta(n^{d+1})$.

In spite of this general result, the author did not succeed in deriving improved bounds for individual k-PD(S)s, $2 \leqq k \leqq n-2$. Let us briefly consider some special cases. For $d = 1$, the number of edges and vertices of k-PD(S) is, respectively, $\Omega(n \log k)$ and $O(n\sqrt{k})$. This follows from results in [13] or [11], and a dual correspondence between configurations of points and arrangements of lines [12]. As shown in [22], the planar order-k Voronoi diagram contains $O(k(n-k))$ cells, edges and vertices. Reference [10] mentions that the bound $\Omega(n^2 \log n)$ is achieved by $(n/2)$-PD(S) in E^2.[2] For $d \geqq 3$, no nontrivial bounds are currently known.

Let $T \subsetneq S$ with $|T| = k \geqq 1$. The remainder of this section gives some results about cell (T).

LEMMA 2. *Let $t \in T$ and $s \in S - T$. Further, let $Z_t = \text{cell}_m(t)$ in $\text{PD}_m(T)$ and $Z_s = \text{cell}(s)$ in $\text{PD}(S-T)$. If $\text{chor}(t, s)$ contains a facet f of $\text{cell}(T)$ then $f = \text{chor}(t, s) \cap Z_t \cap Z_s$.*

Proof. For any $x \in f$, $\text{pow}(x, t) = \text{pow}(x, s)$ and $\text{pow}(x, t') < \text{pow}(x, s')$ holds for all $t' \in T$ and $s' \in S - T$. Thus $\text{pow}(x, t)$ is maximal for all spheres in T and minimal for all spheres in $S - T$. This implies $x \in \text{chor}(t, s) \cap Z_t \cap Z_s$. But each x in this intersection clearly is in cell (T) which completes the argument. □

cell (T) is the intersection of $k(n-k)$ halfspaces and hence a polyhedron with at most $k(n-k)$ facets. It can be shown by example that this bound is tight for $d \geqq 3$. However, we are able to improve it for $d = 2$.

LEMMA 3. *Let S be a set of n circles in E^2 and let $T \subsetneq S$ with $|T| = k \geqq 1$. Then* cell (T) *in* k-PD(S) *is a polygon with at most n edges and vertices.*

Proof. We define an *inner segment* as the union of all edges of cell (T) contained in the same cell of $\text{PD}_m(T)$, and an *outer segment* as the union of all edges of cell (T) contained in the same cell of $\text{PD}(S-T)$. Thus we have at most k inner and $n-k$ outer segments since these diagrams do not have more than k and $n-k$ cells. Due to Lemma 2, each edge of cell (T) is the intersection of an inner and an outer segment. As the union of the inner segments (as well as of the outer segments) is the union of the edges of cell (T), at most $k+n-k$ such intersections are possible. See Fig. 3 for a cell of higher order with n edges. Since cell (T) is a polygon, there are at most n vertices, also. □

Figure 3 illustrates a cell of 3-PD (S) for a set S of 8 congruent circles. Since their radii have no influence, only their centers are marked with ▲ (●) for circles in T ($S-T$). $\text{PD}_m(T)$ is drawn with dashed lines and $\text{PD}(S-T)$ with full lines. The thick lines indicate the boundary of cell (T).

Note that Lemma 3 is also valid for planar order-k Voronoi diagrams. As far as the author knows, no such result has been given even for these diagrams.

As previously observed, not every $T \subsetneq S$ need have a cell (T) $\neq \varnothing$. It is of interest to find sufficient conditions to guarantee the existence of cell (T). To this end, we give the criterion that follows immediately from Lemma 2.

Observation 3. cell (T) $\neq \varnothing$ if and only if there exists a point x with $x \in \text{cell}_m(t)$ in $\text{PD}_m(T)$, $x \in \text{cell}(s)$ in $\text{PD}(S-T)$, and $\text{pow}(x, t) < \text{pow}(x, s)$.

4. Embedding in $d+1$ dimensions. This section investigates the close relationship between power diagrams in E^d and objects in E^{d+1}. Hereby, we make use of a known geometric transform which relates spheres in E^d to hyperplanes and points in E^{d+1}

[2] A referee has kindly informed the author that the overall number of faces of the diagrams 1-PD (S), $\cdots$, k-PD (S) in E^2 is bounded by $O(nk^5)$, based on a result in B. Chazelle and F. P. Preparata, *Halfspace range search: an algorithmic application of k-sets*, Proceedings of the Symposium on Computational Geometry, 1985, pp. 107-115.

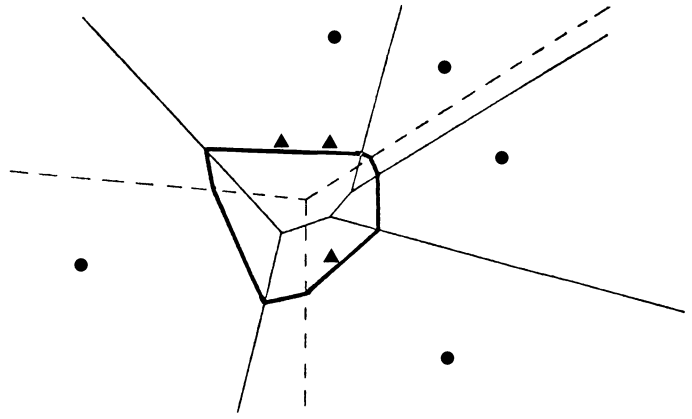


FIG. 3. cell (T) *for* $|T|=3$.

in an elementary and intuitive way. The concept of PDs turns out to be very general and allows a connection between various well-known geometric structures.

Subsection 4.1 formulates the basic geometric ideas and discusses some elementary consequences. In § 4.2, the equivalence to polyhedra and the dual correspondence to convex hulls is exhibited. These results can be viewed as a generalization of the results in [7] and [33] concerning Voronoi diagrams. In fact, we show that power diagrams are exactly those generalized Voronoi diagrams that correspond to polyhedra and thus, in the dual environment, to convex hulls. The equivalence to polyhedra has been independently perceived in [28]. Section 4.3 describes the relationship of higher-order power diagrams to arrangements of hyperplanes. For the particular instance of Voronoi diagrams, this relationship was first exploited in [10].

4.1. The transform. That PDs can be embedded in one dimension higher relies on the fact that the power function of a sphere in E^d can be described by a hyperplane in E^{d+1}.

Let E^{d+1} be spanned by the coordinate axes $x_1, \cdots, x_{d+1}$ and let h_0 denote the hyperplane $x_{d+1}=0$. Furthermore, let U denote the paraboloid $x_{d+1}=x^2$, for $x=(x_1, \cdots, x_d)$. We define the transform Π that maps a sphere $s \subsetneq h_0$ with center z and radius r into the hyperplane

$$\Pi(s): \quad x_{d+1}=2xz-z^2+r^2$$

and vice versa.

It is an easy analytic exercise to prove that Π is a bijective mapping between arbitrary spheres in h_0 and nonvertical (i.e. not parallel to the x_{d+1}-axis) hyperplanes in E^{d+1} that intersect U. In fact, $\Pi(s) \cap U$ is the *vertical projection* of s onto U. The next observation shows that $\Pi(s)$ expresses the power function of s in an elegant way.

Observation 4. Let s and x be a sphere and a point in h_0, and let x' (x'') denote the vertical projection of x onto U $(\Pi(s))$. Then $\text{pow}(x, s)=d(x, x')-d(x, x'')$ holds.

Proof. Let z and r denote the center and the radius of s, respectively. From the equations describing U and $\Pi(s)$ we obtain $d(x, x')=x^2$ and $d(x, x'')=2xz-z^2+r^2$, and thus the difference $(x-z)^2-r^2=\text{pow}(x, s)$. □

Observation 4 implies the following main assertion.

LEMMA 4. *Let s and t be nonco-centric spheres in h_0. Then* $\text{chor}(s, t)$ *is the vertical projection of* $\Pi(s) \cap \Pi(t)$ *onto* h_0.

Figure 4 displays the correspondence between the radical axis of two circles in the x_1x_2-plane and the intersection line of their associated planes in E^3. Observe that the plane of a circle can never be parallel to the x_3-axis.

Our next aim is to derive a dual correspondence between hyperplanes and points in E^{d+1}. To this end, the *polarity* Δ w.r.t. U is defined to map a hyperplane h: $x_{d+1} = a_1x_1 + \cdots + a_dx_d + a_{d+1}$ into the point

$$\Delta(h) = (a_1/2, \cdots, a_d/2, -a_{d+1})$$

and a flat f in E^{d+1} into $\Delta(f) = \cup_{h \supseteq f} \Delta(h)$. Simple analytic calculations imply that Δ is a bijection between j-flats and $(d-j)$-flats $(0 \leqq j \leqq d)$. Moreover, Δ is *involutory*, i.e., its own inverse. In particular, $\Delta(\Delta(p)) = p$ for $p \in E^{d+1}$, where p is called the *pole* of $\Delta(p)$ w.r.t. U, and $\Delta(p)$ is called the *polar hyperplane* of p w.r.t. U. An additional characteristic property of Δ is that it preserves the relative position of points and hyperplanes.

Observation 5. Let p and h be a point and a nonvertical hyperplane in E^{d+1}. Then p is above (in, below) h exactly if $\Delta(h)$ is above (in, below) $\Delta(p)$.

It is worthwhile to mention that spheres, hyperplanes and points can be also related to each other without using the paraboloid U. For example, U may be replaced by any other paraboloid in E^{d+1} with vertical rotation axis, and *stereographic projection* (*inversion*, see [8]) may be used instead of vertical projection to map spheres into hyperplanes. Further, $T(h) = p$ for h: $x_{d+1} = p_1x_1 + \cdots + p_dx_d + p_{d+1}$ and $p = (p_1, \cdots, p_{d+1})$, and $T(p) = h'$, for $h' = -p_1x_1 - \cdots - p_dx_d + p_{d+1}$, is also a dual correspondence [8]. For the above mappings, Lemma 4 and Observation 5 remain valid.

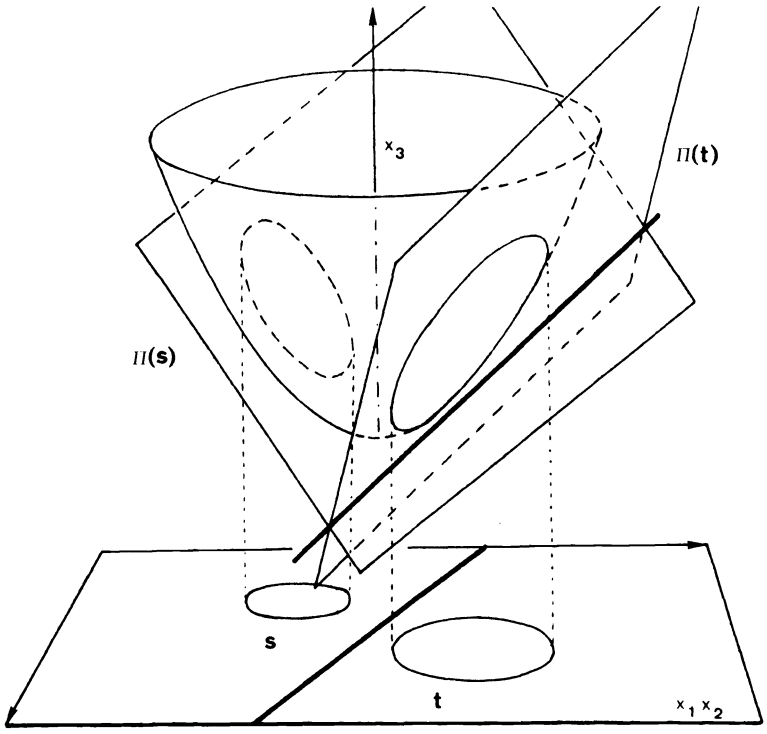


FIG. 4. *Spatial interpretation of* chor (s, t).

The reasons for using Π and Δ are their simple geometric interpretation and that they result in a simple analysis.

This subsection closes with a necessary and sufficient condition for a set of hyperplanes to be the chordales defined by n spheres.

LEMMA 5. *Let H be a set of hyperplanes in E^d. Then there exists a set S of n spheres in E^d such that H is the set of chordales for S if and only if there is a mapping $\gamma: \{1, \cdots, n\}^2 \to H \cup \emptyset$ with the following properties:*

(1) $\gamma(i,j) = \gamma(j,i) \neq \emptyset$ *for* $1 \leqq i < j \leqq n$,

(2) $\gamma(i,i) = \emptyset$ *for* $1 \leqq i \leqq n$,

(3) $\gamma(i,j) \cap \gamma(i,k) = \gamma(i,j) \cap \gamma(j,k) = \gamma(i,k) \cap \gamma(j,k)$ *for* $1 \leqq i < j < k \leqq n$.

Proof. Without loss of generality, we identify E^d with the hyperplane h_0 of E^{d+1}. Then γ completely characterizes a set H of $(d-1)$-flats in h_0 which are the vertical projections of the pairwise intersections of n nonvertical and nonparallel hyperplanes $g_1, \cdots, g_n$ in E^{d+1}: If H represents such a projection then γ obviously exists. If γ exists, then $g_1, \cdots, g_n$ can be constructed as follows. Let $f_{i,j}$ denote the vertical projection of $\gamma(i,j)$ onto g_i, for $1 \leqq i < j \leqq n$. g_1 is chosen to be arbitrary but nonvertical, g_2 is chosen to be nonvertical, to pass through $f_{1,2}$, and to be distinct from g_1, and g_k, for $k = 3, \cdots n$, through $f_{1,k}$ and $f_{2,k}$. According to (3), g_k exists and $g_k \cap g_i = f_{i,k} (2 < i < k)$, since $g_1 \cap g_i \cap g_k \subsetneq f_{i,k}$ and $g_2 \cap g_i \cap g_k \subsetneq f_{i,k}$. The reader may convince himself that g_k can be chosen such that $g_k \cap U \neq \emptyset$, for $1 \leqq k \leqq n$. Then each g_k corresponds to a unique sphere $\Pi(g_k)$, and by Lemma 4, we deduce that H is the set of chordales for $S = \{\pi(g_k) \mid k = 1, \cdots, n\}$. □

4.2. Power diagrams and polyhedra. The geometrical results in § 4.1 enable us to prove a main theorem of this paper. It asserts that power diagrams in E^d are, essentially, the same as the boundaries of $(d+1)$-polyhedra.

THEOREM 4. *For any $(d+1)$-polyhedron P, which is expressible as the intersection of upper halfspaces, there exists an affinely equivalent power diagram in h_0, and vice versa.*

Proof. Let $P = \bigcap_{i=1}^{n} h_i$, such that the upper halfspace h_i determines a facet f_i of P and, w.l.o.g., assume $h_i \cup U \neq h_i$, for $i = 1, \cdots, n$. There is a set $S = \{s_1, \cdots, s_n\}$ of spheres in h_0 such that $\Pi(s_i)$ bounds h_i below. Observation 4 now implies that the vertical projection of $x \in h_0$ onto P is in f_i exactly if $x \in$ cell (s_i). In other words, cell (s_i) is the vertical projection of f_i onto h_0, for $i = 1, \cdots, n$. This shows the existence of an affinely equivalent power diagram PD (S) for arbitrary P. Conversely, the existence of P for arbitrary PD (S) in h_0 results from the existence of $\Pi(s_i)$ for any $s_i \subsetneq h_0$. □

Observe that the reflection P' of P through h_0 also is affinely equivalent to PD (S). Observation 4 now implies that PD (S) is the maximal power diagram of some set of spheres. We thus have the following.

COROLLARY 1. *For any P in Theorem 4, there is an affinely equivalent maximal power diagram in h_0, and vice versa.*

Another straightforward consequence of Theorem 4 is given.

COROLLARY 2. *A cell complex C in E^d is polytopal if and only if C is a power diagram.*

We proceed to demonstrate the dual correspondence between power diagrams and convex hulls. Let M denote a finite subset of E^{d+1}. In a natural way, the boundary of the convex hull CH (M) of M splits into two parts: The *top part* $\mathrm{CH}_t(M)$ (the *bottom part* $\mathrm{CH}_b(M)$) of M consists of all facets f of CH (M) (and their bounding faces), such that CH (M) lies in the lower (upper) halfspace bounded by the hyperplane of E^{d+1} that contains f. Vertical facets, if some exist, are assigned neither to $\mathrm{CH}_t(M)$ nor to $\mathrm{CH}_b(M)$. We shall prove the following.

THEOREM 5. *For any finite set $M \subsetneq E^{d+1}$, there exists a set S of spheres in h_0 such that* PD (S) *is dual to* $\mathrm{CH}_b(M)$ *and* $\mathrm{PD}_m(S)$ *is dual to* $\mathrm{CH}_t(M)$. *The reverse of the assertion also holds.*

Proof. By Theorem 4, there is an affinely equivalent polyhedron P for each PD (S) in h_0 and vice versa. P is the intersection of the halfspaces above the hyperplanes in $H = \{\Pi(s) \mid s \in S\}$. We let $M = \{\Delta(h) \mid h \in H\}$ and show the duality of P and $\mathrm{CH}_b(M)$. Clearly, there exists exactly one $p_i \in M$ for each facet f_i of P. Let h_i be the hyperplane containing f_i, and let f_i and f_j be adjacent in a $(d-1)$-face g $(1 \leqq i < j \leqq |H|)$. Then each $x \in g$ is in $h_i \cap h_j$ and above each $h \in H - \{h_i, h_j\}$. Observation 5 now implies that exactly then $p_i, p_j \in \Delta(x)$, and each $p \in M - \{p_i, p_j\}$ is above $\Delta(x)$, that is, p_i and p_j define an edge of $\mathrm{CH}_b(M)$. The duality of P, and thus of PD (S), and $\mathrm{CH}_b(M)$ immediately follows. By Corollary 1, the proof of the duality of $\mathrm{PD}_m(S)$ and $\mathrm{CH}_t(M)$ is completely analogous. □

For $d+1=3$, $\mathrm{CH}_b(M)$ has the following interpretation. The vertical projection D of $\mathrm{CH}_b(M)$ onto the x_1x_2-plane is a planar graph which is dual to PD (S). The vertices of D are the centers of the circles in S, and dual edges are perpendicular. (The interested reader may verify this fact.) If S is a finite set of points then D is the Delaunay triangulation of S.

4.3. Power diagrams and arrangements. A finite set H of hyperplanes in E^{d+1} dissects E^{d+1} into polyhedra which define a cell complex, often called the *arrangement* $A(H)$ of H. In this subsection, the correspondence of higher-order power diagrams to arrangements is described.

Let $S = \{s_1, \cdots, s_n\}$ be a set of spheres in h_0, and let $H = \{\Pi(s_1), \cdots, \Pi(s_n)\}$. We start by showing that each cell of k-PD (S) is the projection of a cell of $A(H)(1 \leqq k \leqq n-1)$. For $x \in h_0$, let r_x denote the vertical ray in E^{d+1} that contains x and is directed to $-\infty$. Since the hyperplanes in H are nonvertical, they can be assigned to x in the same order as they are intersected by r_x. According to Observation 4, r_x intersects $\Pi(s_1), \cdots, \Pi(s_k)$ first and then $\Pi(s_{k+1}), \cdots, \Pi(s_n)$ exactly if pow $(x, s_i) <$ pow (x, s_j), for $i = 1, \cdots, k$ and $j = k+1, \cdots n$. By definition, $x \in$ cell $(\{s_1, \cdots, s_k\})$ then. This yields the following.

LEMMA 6. *Let $Z = Z^- \cap Z^+$, where Z^- is the intersection of the halfspaces below $\Pi(s_1), \cdots, \Pi(s_k)$ and Z^+ is the intersection of the halfspaces above $\Pi(s_{k+1}), \cdots, \Pi(s_n)$.* cell (T), *for $T = \{s_1, \cdots, s_k\}$, is the image of Z under vertical projection onto h_0.*

The existence of cell (T) thus depends on the existence of Z. The relationship to Lemma 2 is worth mentioning. As proved in § 4.2, the faces of the unbounded polyhedra Z^- and Z^+ project to the faces of $\mathrm{PD}_m(T)$ and PD $(S-T)$. The intersection of a facet f_i of Z^- in $\Pi(s_i)$ with a facet f_j of Z^+ in $\Pi(s_j)$ has dimension $d-1$ if and only if chor (s_i, s_j) defines a facet of cell (T). Figure 5 illustrates the two 3-polyhedra corresponding to the power diagram and the maximal power diagram in Fig. 3, as well as their intersection Z. The thick-lined edges of Z project to the edges of cell (T) onto the x_1x_2-plane.

Lemma 6 makes it clear that k-PD (S) is the projection of certain faces of $A(H)$. To characterize this relationship more precisely, we introduce particular subsets of $A(H)$. The *k-level* L_k of $A(H)$ consists of all d-faces f of $A(H)$ (and the faces bounding f) such that f is below exactly k hyperplanes in H. It is not difficult to see that L_k is a connected subset of $A(H)$, and that $L_k \cap L_j$ consists of faces of dimension less than d $(1 \leqq k < j \leqq n)$. Furthermore, L_1 (L_n) bounds the intersection of the upper (lower) halfspaces defined by H. As a consequence, L_1 projects to PD (S) and L_n projects to $\mathrm{PD}_m(S)$. Moreover, we have the following.

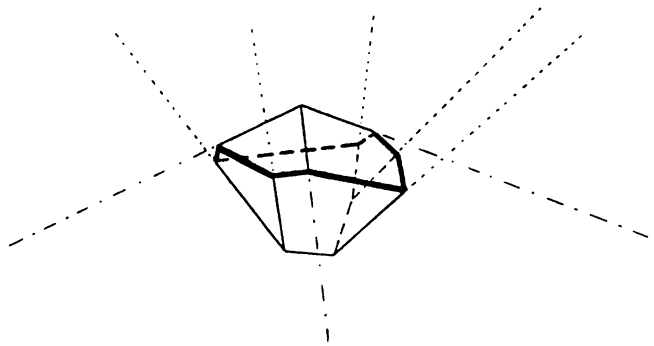


FIG. 5. *The polyhedron Z exists.*

THEOREM 6. *Let h_0, S, and L_k be defined as above. The vertical projection of $L_k \cap L_{k+1}$ onto h_0 represents the collection of j-faces of k-PD (S), for $j = 0, \cdots, d-1$ and $1 \leqq k \leqq n-1$.*

Proof. By Lemma 6, each cell Z of $A(H)$ (except the two polyhedra with boundary L_1 and L_n) projects to a cell Z' of k-PD (S), for some k. Z is the intersection of k lower and $n-k$ upper halfspaces and thus is enclosed between L_k and L_{k+1}. Let Z^- (Z^+) denote the intersection of these lower (upper) halfspaces. Then the intersection of the boundaries of Z^- and Z^+ is a subset of $L_k \cap L_{k+1}$ and, on the other hand, it projects to the boundary of Z' in h_0. □

As a consequence, the information inherent in all k-PD (S)s, for $k = 1, \cdots, n-1$, is inherent in $A(H)$. Note also that every arrangement in E^{d+1} that does not contain a vertical hyperplane (and whose hyperplanes intersect the paraboloid U, which can always be achieved by translating the arrangement vertically) corresponds to the k-PDs of some set of spheres in h_0. Furthermore, the following interesting property holds.

LEMMA 7. *Let S be a set of n spheres in h_0. There is a set S^* of spheres in h_0 such that k-PD $(S) = (n-k)$-PD (S^*), for $k = 1, \cdots, n-1$.*

Proof. Let r be a fixed vertical vector such that $h_s + r \cap U \neq \emptyset$, for each $s \in S$ and let h_s be the mirror image of $\Pi(s)$ through h_0. Then $S^* = \{s^* | \Pi(s^*) = h_s, s \in S\}$ has the desired properties by Theorem 6 and the definition of a k-level. □

It should be noted that Lemma 7, in general, is not true for Voronoi diagrams. More precisely, we cannot always find a point-set S^* for any set S of n points in E^d such that the order-k Voronoi diagram of S coincides with the order-$(n-k)$ Voronoi diagram of S^*.

The maximal size of an arrangement (see [37] or [1] for a deduction of the following formula) implies asymptotically tight bounds on the number of faces of all k-PDs for a fixed set of spheres.

LEMMA 8. *Let H be a set of n hyperplanes in E^d. Then the number of j-faces of $A(H)$ is less than or equal to*

$$\sum_{i=d-j}^{d} \binom{i}{d-j}\binom{n}{i}$$

which is in $\Theta(n^d)$, for $0 \leqq j \leqq d$.

Proof of Theorem 3. It follows from Lemma 8 in conjunction with Theorem 6 that 1-PD $(S), \cdots, (n-1)$-PD (S) together have $O(n^{d+1})$ j-faces, for $0 \leqq j \leqq d$. The bound

is asymptotically tight since each j-face of $A(H)$, for $j<d$, projects to some j-face, and each $(d+1)$-face of $A(H)$ projects to some d-face, of a particular k-PD (S). □

5. How to construct power diagrams. We turn our attention from geometry to the design of algorithms for constructing power diagrams and order-k power diagrams. By using the correspondences proven in § 4, we obtain straightforward methods. We only give a rough idea of how to proceed since the expensive parts of the algorithms are known geometric tasks. Throughout this section, let S denote a set of n spheres in E^d, and let E^d be identified with h_0: $x_{d+1}=0$ in E^{d+1}.

5.1. Constructing PD (S) and PD$_m$ (S). First, we concentrate on the construction of PD (S) and PD$_m$ (S). By construction we mean the computation of a data structure that represents the induced cell complex C in an appropriate way. Clearly, the data structure depends on the kind of inherent information required. A data structure adaptable to most requirements is the so-called *incidence lattice* $I(C)$ of C (see e.g. [3] or [10]): Each j-face of C is represented by a node $(0\leqq j\leqq d)$, and nodes of incident faces are associated via pointers. Beside the combinatorial structure of C, we have to fix the position of C in E^d. It suffices to store the coordinates of the vertices or the positions of the hyperplanes through the facets of C.

According to Theorem 5, $I(\text{PD}(S))$ and $I(\text{PD}_m(S))$ can be obtained from the bottom and the top part of a certain convex hull in E^{d+1}. Thus the main part of the algorithm is the determination of a convex hull. We refer to [32] for a detailed description of a convex hull algorithm for finite point-sets in arbitrary finite dimensions. Since the method in [32] is optimal only in even dimensions, the algorithm in [29] is preferable for point-sets in E^3. We state their results explicitly as follows.

PROPOSITION 1. *Let M be a set of n points in E^d. The convex hull of M can be determined in $O(n\log n)$ time for $d=3$ and in $O(n\log n+n^{\lceil d/2\rceil})$ time for $d\neq 3$ and requires $O(n+n^{\lfloor d/2\rfloor})$ space.*

In what now follows, a more formal and detailed description of the algorithm to construct the (maximal) power diagram is given.

ALGORITHM (MAXIMAL) POWER DIAGRAM.

Let $S=\{s_1,\cdots,s_n\}$. The computation of PD (S) and PD$_m$ (S) requires three steps.

Step 1. For $i=1,\cdots,n$, compute the hyperplanes $\Pi(s_i)$ and their poles $p_i=\Delta(\Pi(s_i))$, using the formulae in § 4.1.

Step 2. Determine the intersection Z^+ of the halfspaces above $\Pi(s_1),\cdots,\Pi(s_n)$ and the intersection Z^- of their complements.

Step 2.1. Construct the convex hull CH of $\{p_1,\cdots,p_n\}$ (by means of the algorithm in [29] for $d+1=3$ and the algorithm in [32], otherwise) such that CH is represented by $I(\text{CH})$.

Step 2.2. Split $I(\text{CH})$ into I_t for the top part and I_b for the bottom part of CH. The facets f of CH (and the faces in the boundary of f) are assigned to I_t or I_b according to the slope of the hyperplane containing f.

Step 2.3. Replace each j-face in I_t and I_b by a $(d-j)$-face, for $j=0,\cdots,d$. For each vertex that arises from a facet f, we store the coordinates of the pole $\Delta(h)$ of the hyperplane h through f. The coordinates of $p_1,\cdots,p_n$ are deleted. We now have $I_t=I(Z^-)$ and $I_b=I(Z^+)$.

Step 3. Project Z^+ and Z^- vertically onto h_0 (that is, ignore the $(d+1)$st coordinates of their vertices). This finally yields PD (S) and PD_m (S) in h_0.

Let us analyze the time used by this algorithm. Obviously, Step 1 can be performed in $O(n)$ time. As a consequence of the properties of an incidence lattice, Steps 2.2, 2.3 and 3 require time proportional to the size of PD (S) and PD_m (S). Since the size of these diagrams implies a trivial lower time bound for their construction, the time complexity essentially depends on Step 2.1. This yields the following.

THEOREM 7. *Let S be a set of n spheres in E^d, and let $T_d(n)$ $(S_d(n))$ denote the amount of time (space) necessary to compute the convex hull of n point in E^d. Algorithm* (MAXIMAL) POWER DIAGRAM *constructs* PD (S) *and* PD_m (S) *in $O(T_{d+1}(n))$ time and $O(S_{d+1}(n))$ space.*

The time optimality of the algorithm thus depends on the optimality of convex hull algorithms. According to Proposition 1 and Theorem 1, the method is asymptotically time-optimal for $d=2$ and every odd d, and space-optimal for arbitrary d (in the worst case).

5.2. Building up all higher-order PDs. This subsection deals with the construction of all higher-order power diagrams of S. Theorem 6 provides the strategy: Construct the arrangement $A(H)$ induced by $H=\{\Pi(s)|s\in S\}$, determine the faces contained in $L_k\cap L_{k+1}$, and project them vertically onto h_0. This yields the faces bounding the cells of k-PD (S), for $k=1,\cdots,n-1$. Evidently, the major part of the algorithm is building $A(H)$. An asymptotically time- and space-optimal method for constructing arbitrary arrangements in E^d by incrementally inserting the hyperplanes is presented in [10]. Their main result reads as follows.

PROPOSITION 2. *Let H be a set of n hyperplanes in E^d and let $h\in H$. Then $A(H)$ can be obtained from $A(H-\{h\})$ in $O(n^{d-1})$ time.*

We are left with the problem of projecting $L_k\cap L_{k+1}$ onto h_0, for $k=1,\cdots,n-1$. The representation of $A(H)$ by its incidence lattice $I(A(H))$ allows us to identify the j-faces $(0\leqq j\leqq d-1)$ in $L_k\cap L_{k+1}$ in time proportional to the size of L_k without asymptotically increasing the space requirement. See [10] for the augmentation of the nodes of $I(A(H))$ necessary to carry out this task. We conclude as follows.

THEOREM 8. *Let S be a set of n spheres in E^d. Then* 1-PD $(S),\cdots,(n-1)$-PD (S) *can be computed in $O(n^{d+1})$ time and space.*

By Theorem 3, the result is asymptotically optimal for any d. Note that the strategy can be used for on-line constructing of higher-order PDs by inserting the given spheres.

5.3. Computing particular power cells. In some situations it might be of interest to determine a particular power cell of k-PD (S) $(1\leqq k\leqq n-1)$ or to detect its existence. Let $T\subsetneq S$ and $|T|=k$. Trivially, cell (T) can be computed by intersecting $k(n-k)$ halfspaces in E^d which are bounded by chordales. While this method seems to be optimal e.g. in E^3 (Lemma 2 suggests that cell (T) may contain $k(n-k)$ facets), the number of bounding faces of cell (T) in E^2 is limited by n (compare Lemma 3). Thus this problem begs for a more efficient solution.

Indeed, Lemma 6 offers an alternative construction method for $S\subsetneq E^2$. The boundary of cell (T) is the vertical projection of the intersection of the boundaries of two 3-polyhedra Z^- and Z^+ which correspond to the intersection of k lower and $n-k$ upper halfspaces in E^3. References [30] and [33] give algorithms that compute the intersection of n arbitrary halfspaces in E^3 in $O(n\log n)$ time and $O(n)$ space. (It is worth mentioning that, similarly to Algorithm (MAXIMAL) POWER DIAGRAM,

these algorithms apply convex hull algorithms to point-sets obtained from the set of half-spaces via dual transforms.)

ALGORITHM POWER CELL.
Let $S \subsetneq h_0$, $T \subsetneq S$, and $T \neq \emptyset$. cell (T) is constructed as follows:
Step 1. Construct the 3-polyhedron $Z = Z^- \cap Z^+$ using the algorithm in [30] or [33].
Step 2. If $Z \neq \emptyset$ then compute the intersection of the boundaries of Z^- and Z^+. This yields a cyclic sequence Q of adjacent edges.
Step 3. Project Q vertically onto h_0. This yields the edges of cell (T).

From the preceding discussion, we obtain the following.

THEOREM 9. *For a set S of n circles in E^2, Algorithm* POWER CELL *computes* cell (T) *for arbitrary fixed $T \subsetneq S$ in $O(n \log n)$ time and $O(n)$ space.*

Clearly, the space requirement is optimal to within a constant factor since cell (T) can have up to n edges. We shall prove the time bound to be optimal, too. Let M and M' be the set of centers of the circles in T and $S - T$, respectively. Lemma 2 makes it clear that M and M' can be obtained, sorted by polar angles w.r.t. a point p interior to the convex hull of M', in $O(n)$ time from the cyclic order of the edges of cell (T). (A suitable point p can be found in $O(1)$ time by determining the mass center of three points in M'.) Since $\Omega(k \log k) + \Omega((n-k) \log (n-k)) = \Omega(n \log n)$, for $k = |T|$, is a lower time bound for this sorting problem, the same bound holds for the construction of cell (T), too.

Note that Algorithm POWER CELL can be used to detect the existence of cell (T). Compare this with Observation 3 that suggests a different approach to the detection problem.

6. Applications. Power diagrams are applicable to several problems involving spheres and polyhedra in E^d. In this section we give a brief description of four situations where PDs lead to an efficient computational solution, and serve to analyze the space complexity. We mention only marginally that PDs apply naturally to computing the union or intersection of balls in E^d, since these tasks have been solved in [8] with the same time and space complexity that would result from using PDs. The efficiency of many of the algorithms given below depends only on convex hull algorithms. So throughout this section, let $T_d(n)$ ($S_d(n)$) denote the time (space) required to compute the convex hull of n points in E^d (compare Proposition 1).

6.1. Illumination of balls. Let V and B be subsets of E^d. We say that *V illuminates B* if, for each point x on the boundary of B, there is a point y in V such that the relatively open line segment joining x and y does not intersect B. As reported in [25], the union B of n balls in E^d can be illuminated by the vertices of the power diagram for the corresponding spheres. Thus finding illumination points for B is reduced to computing a PD. From Theorem 7, we conclude as follows.

THEOREM 10. *Let B be the union of n balls in E^d. Illumination points for B can be found in $O(T_{d+1}(n))$ time and $O(S_{d+1}(n))$ space.*

Since the minimal number of points necessary to illuminate B is $\Theta(S_{d+1}(n))$ in the worst-case [25], the space bound is optimal for any d, and the time bound is

optimal for odd $d \geqq 3$. By reducing the illumination problem to sorting $O(n)$ reals, we deduce that the time bound is optimal for $d = 1, 2$, also.

6.2. Detecting k-sets. Let M denote a set of n points in E^d, and let $M' \subsetneq M$ with $|M'| = k (1 \leqq k \leqq n-1)$. M' is called a *k-set* of M if there exists a halfspace h such that $h \cap M = M'$.

The k-sets of M can be characterized in terms of power diagrams by using the geometrical transforms Π and Δ introduced in § 4.1. (Without loss of generality, assume that the sphere $\Pi(h)$ exists for each halfspace h below.)

LEMMA 9. *M' is a k-set of M if and only if* cell (T) *in* k-PD (S) *is nonempty, for* $T = \{\Pi(\Delta(q)) | q \in M'\}$ *and* $S = \{\Pi(\Delta(p)) | p \in M\}$.

Proof. Assume that M' is a k-set of M, i.e., there exists a hyperplane h such that, w.l.o.g., each $q \in M'$ is below h, and each $p \in M - M'$ is above h. By Observation 5, this is equivalent to the fact that point $\Delta(h)$ is below all hyperplanes $\Delta(q)$, and above all hyperplanes $\Delta(p)$. Thus $\Delta(h)$ is contained in the intersection Z of the halfspaces below all $\Delta(q)$ and above all $\Delta(p)$, that is, $Z \neq \emptyset$. Since cell (T) in k-PD (S) is the vertical projection of Z (see Lemma 6), the assertion follows directly. □

The above discussion implies that M' can be recognized to be a k-set of M by intersecting either n halfpaces in E^d or $k(n-k)$ halfpaces in E^{d-1}. Since intersecting halfspaces is essentially equivalent to determining a convex hull [8], [32] we obtain the following.

THEOREM 11. *Let M' be a subset of cardinality k of a set M of n points in E^d. Then* $O(\min\{T_d(n), T_{d-1}(k(n-k))\})$ *time suffices to detect whether or not M' is a k-set of M.*

Note that, for each $p \in Z$, M' is separated from $M - M'$ by $\Delta(p)$. A point p interior to Z can be computed in $O(1)$ time given Z (choose p as the mass center of $d+1$ boundary points of Z which do not belong to the same facet), and in $O(n)$ time given cell (T) (choose a point x interior to cell (T), intersect the parts of the vertical ray through x which are contained in the halfspaces determining Z, and choose p in this intersection). Thus a separating hyperplane for M' can be computed within the time bounds in Theorem 11.

6.3. Multiplicative weighted Voronoi diagrams. Let $p \in E^d$ have an associated *weight* $w(p) > 0$. The *multiplicative weighted distance* from $x \in E^d$ to p is defined as $d_m(x, p) = d(x, p)/w(p)$. Among a set M of n weighted points, the *region* (of dominating influence) of p is determined by $\{x \in E^d | d_m(x, p) < d_m(x, q), \forall q \in M - \{p\}\}$. The subdivision of E^d induced by the regions of the points in M is called the *multiplicative weighted Voronoi diagram* of M.

By easy analytic arguments, the regions of any distinct $p, q \in M$ are separated by a sphere sph (p, q). Moreover, sph (p, q), sph (p, r), and sph (q, r), for distinct $p, q, r \in M$, define a common chordale. Thus $\Pi(\text{sph}(p, q)) \cap \Pi(\text{sph}(p, r)) \cap \Pi(\text{sph}(q, r))$ is a $(d-1)$-flat (compare Lemma 4), such that $\{\Pi(\text{sph}(p, q)) | p, q \in M, p \neq q\}$ is the set of chordales of some set S of spheres in E^{d+1} (Lemma 5). It is not difficult to observe that there is a one-to-one correspondence σ between M and S such that the boundary of the region of $p \in M$ is the vertical projection of the intersection of cell $(\sigma(p))$ in PD (S) with the projection paraboloid U. S can be constructed iteratively from M in $O(n)$ time in a straightforward manner. Since projecting does not affect the asymptotic runtime, Theorem 7 implies the following.

THEOREM 12. *The multiplicative weighted Voronoi diagram for n points in E^d can be constructed in $O(T_{d+2}(n))$ time and $O(S_{d+2}(n))$ space.*

See [2] and [3] for algorithms yielding the same (optimal) bounds for $d = 1, 2$, respectively. For $d \geqq 3$, the result appears to be new. It tells us that the space complexity

of the diagram is asymptotically the same as of a power diagram in E^{d+1}, and that the method is time-optimal for d even, and space-optimal for any d.

As a matter of fact, the weight of $p \in M$ can be expressed by a cone [2] or a paraboloid $v(p)$ with vertical rotation axis, such that the components of the boundary of the union of the point-sets below all $v(p)$ project vertically to the components of the weighted diagram of M. This yields the following.

COROLLARY 3. *The union (intersection) of n axis-parallel cones in E^d with apexes in a common hyperplane, or of n axis-parallel paraboloids in E^d, can be determined in $O(T_{d+1}(n))$ time and $O(S_{d+1}(n))$ space. This is time-optimal for $d=2$ and odd d, and space-optimal for any $d \geqq 2$.*

Similarly to PDs, the weighted diagram of M can be generalized to order k ($1 \leqq k \leqq n-1$). (The intersection of the above objects then corresponds to the order-$(n-1)$ diagram.) Its regions then can be constructed by intersecting the cells of k-PD(S), for $S=\{\sigma(p)|p \in M\}$, with the paraboloid U. Hence all higher-order diagrams of M can be built up in $O(n^{d+2})$ time and space by Theorem 8. However, this strategy is not optimal since k-PD(S) contains much more information than is inherent in the weighted order-k diagram for $2 \leqq k \leqq n-2$.

6.4. Voronoi diagrams for spheres. Let T denote a set of n spheres in E^d. The *(minimum) distance* of $x \in E^d$ to $s \in T$ is defined as $d(x, s)=\min\{d(x, y)|y \in s\}$. If x is not contained in the ball bounded by s then $d(x, s)=d(x, z_s)-r_s$, where z_s and r_s denote the center and the radius of s. (We therefore could alternatively define $d(x, s)$ as the *additive weighted distance* of x from $p=z_s$ with $w(p)=r_s$.) The *Voronoi diagram of T* associates each $s \in T$ with its *region* $\{x \in E^d|d(x, s)<d(x, t), \forall t \in T-\{s\}\}$. This diagram (which is clearly different from PD(T)) can be related to a power diagram in E^{d+1} in the following way.

Let T be contained in h_0: $x_{d+1}=0$ in E^{d+1}. Further, let $\kappa(s)$ denote the unique cone in E^{d+1} with vertical axis, apex below h_0, and $\kappa(s) \cap h_0=s$, and let $\sigma(s)$ denote the unique sphere inscribed in $\kappa(s)$ with $\sigma(s) \cap \kappa(s)=s$. We have the following properties.

LEMMA 10. *Let $s, t \in T$ such that $r_s<r_t$.*

(1) *$\mathrm{hyp}(s, t)=\{x \in h_0|d(x, t)=d(x, s)\}$ is a hyperboloid with focus z_s and rotation axis through z_s and z_t.*

(2) *$\mathrm{hyp}(s, t)$ is the vertical projection onto h_0 of $\kappa(s) \cap \kappa(t)=\kappa(s) \cap \mathrm{chor}(\sigma(s), \sigma(t))$.*

We omit the proof since it follows from simple geometric calculations. From the above relationship it is obvious that the boundary of the region of s is the vertical projection of $\kappa(s) \cap \mathrm{cell}(\sigma(s))$ in PD(S), for $S=\{\sigma(s)|s \in T\}$. Hence Theorem 7 implies the following computational complexity.

THEOREM 13. *There exists an algorithm that constructs the Voronoi diagram of a set of n spheres in E^d in $O(T_{d+2}(n))$ time and requires $O(S_{d+2}(n))$ space.*

This is optimal only for $d=1$ where the problem degenerates to a sorting problem. In E^2, the diagram realizes (in contrast to the multiplicative weighting scheme) only $O(n)$ components; improvements (e.g., to $O(n \log^2 n)$ time [9]) exist. However, no methods had been known for $d \geqq 3$. Furthermore, our strategy leads to a worst-case optimal construction of the individual region R of a circle $s \subsetneq T \subsetneq E^2$. R may be bounded by $O(n)$ hyperbolic arcs, from which a certain ordering of T (compare Theorem 9) can be obtained.

THEOREM 14. *Let T be a set of n circles in E^2 and let $s \in T$. The region R of s in the Voronoi diagram of T can be computed in $O(n \log n)$ time and $O(n)$ space. This is worst-case optimal.*

If the radius of s is minimal among the circles in T, then R is the intersection of $n-1$ halfplanes bounded by hyperbolas with one common focus. This yields the following.

COROLLARY 4. *The common intersection of n hyperbolically bounded halfplanes with common foci can be determined in $O(n \log n)$ time and $O(n)$ space.*

7. Conclusion. In this paper, the model of power diagrams is introduced. The contributions fall into two parts: The analysis of power diagrams and their construction and application, and the exposition of the central role of power diagrams in geometric environments. As far as is known to the author, no methods for computing power cell complexes in E^d, for $d \geqq 3$ (in their general form), have been outlined before. The main results of this work rely on a fundamental property of the chordale of two spheres (Lemma 5). Similar correspondences between spheres, hyperplanes and points, but mainly expressed in other terms, have been observed, e.g., in [8], [28] and [33]. The various applications of PDs mentioned suggest that PDs serve as a tool for drawing connections between diverse areas in computational geometry.

The equivalence of PDs and convex polyhedra sheds new light into the interesting connection between Voronoi diagrams and particular types of polyhedra first perceived in [7], thus showing that power diagrams are, in some sense, the most natural generalizations of Voronoi diagrams. Consequently, PDs provide a new way of characterizing cell complexes, namely those that can be obtained by projecting the faces bounding a convex polyhedron. This is independently observed in [28], but is presented here in a more natural and simple way. In fact, many naturally arising cell complexes are PDs (see e.g. §§ 6.3 and 6.4), and hence are constructable via our algorithms. We have also shown that PDs can be related to convex hulls and arrangements of hyperplanes. It seems surprising that all these elementary structures are related by duality and affine equivalence. As a consequence, the computation of PDs and their order-k modifications can be efficiently carried out using known geometric algorithms. This work illustrates once more that geometric transforms are the key to new fruitful methods.

Let us summarize the algorithms outlined in this paper.

(1) Algorithm (MAXIMAL) POWER DIAGRAM constructs PD (S) as well as $\mathrm{PD}_m(S)$, for a set S of n spheres in E^d. Its complexity depends on that of convex hull algorithms and hence is optimal for $d=2$ and any odd d.

(2) An algorithm for simultaneously constructing 1-PD $(S), \cdots, (n-1)$-PD (S), for $n=|S|$, is outlined. It is based on the construction of arrangements of hyperplanes and is optimal for any d.

(3) Algorithm POWER CELL computes the power cell of a particular subset T of S in E^2. The method is worst-case optimal and can be used to decide whether cell (T) is empty.

Applications of PDs to the following problems are addressed:

(i) Finding illumination points for a finite set of spheres in E^d.

(ii) Recognizing k-sets of a finite set of points in E^d resp. determining a separating hyperplane.

(iii) Constructing the multiplicative weighted Voronoi diagram of a finite set of points in E^d (computing the union or intersection of certain types of cones or paraboloids in E^{d+1}), and constructing its order-k modifications.

(iv) Constructing the Voronoi diagram of a finite set of spheres in E^d, and determining the region of a particular circle of the given set in E^2 (determining the intersection of hyperbolically bounded halfplanes in E^2 with common foci).

In all four applications, power diagrams lead to an improved or to the first known solution, and serve to analyze the space requirement.

Finally, we state some open problems that arise in connection with PDs. First, fast algorithms for computing the convex hull of a finite set in E^d, for $d \geqq 5$, d odd, would be of importance. This paper shows once more the applicability of convex hull algorithms in higher dimensions to problems in E^2 and E^3. Further, the efficient construction of the individual order-k power diagram is still an open problem. It seems that the correspondence to k-levels (Theorem 6) can be exploited to obtain a satisfactory solution. In addition, improved lower and upper bounds on the number of faces of a k-PD are of interest. Establishing bounds on the size of a k-level would answer this question.

REFERENCES

[1] G. L. ALEXANDERSON AND J. E. WETZEL, *Simple partitions of space*, Math. Mag., 51 (1978), pp. 220–225.

[2] F. AURENHAMMER, *The one-dimensional weighted Voronoi diagram*, in Inf. Proc. Letters, to appear, also available: Rep. F 110, IIG, Tech. Univ. of Graz, Austria, 1983.

[3] F. AURENHAMMER AND H. EDELSBRUNNER, *An optimal algorithm for constructing the weighted Voronoi diagram in the plane*, Pattern Recognition, 17(2) (1984), pp. 251–257.

[4] W. BLASCHKE, *Vorlesungen ueber Differentialgeometric* III, Springer-Verlag, Berlin, 1929.

[5] B. N. BOOTS, *Weighting Thiessen polygons*, Econom. Geog. (1979), pp. 248–259.

[6] A. BRÖNSTED, *An Introduction to Convex Polytopes*, Springer-Verlag, New York, Heidelberg, Berlin, 1982.

[7] K. Q. BROWN, *Voronoi diagrams from convex hulls*, Inform. Process Lett., 9 (1979), pp. 223–228.

[8] ———, *Geometric transforms for fast geometric algorithms*, Ph.D. thesis, Rep. CMU-CS-80-101, Dep. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, 1980.

[9] R. L. DRYSDALE, III AND D. T. LEE, *Generalization of Voronoi diagrams in the plane*, this Journal, 10 (1981), pp. 73–87.

[10] H. EDELSBRUNNER, J. O'ROURKE AND R. SEIDEL, *Constructing arrangements of lines and hyperplanes with applications*, Proc. 24th. Ann. IEEE Symp. FOCS (1983), pp. 83–91.

[11] H. EDELSBRUNNER AND E. WELZL, *On the number of line separations of a finite set in the plane*, J. Combin. Theory Ser. A (1985), pp. 15–29.

[12] ———, *Halfplanar range estimation*, Rep. F 98, IIG, Tech. Univ. of Graz, Austria, 1982.

[13] P. ERDÖS, L. LOVASZ, A. SIMMONS AND E. G. STRAUS, *Dissection graphs for planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava et al., eds., Amsterdam, 1973.

[14] L. FEJES TOTH, *Lagerungen in der Ebene, auf der Kugel und im Raum*, Springer-Verlag, Berlin, Heidelberg, 1972.

[15] ———, *Illumination of convex discs*, Acta Math. Acad. Sci. Hungar., 29, 3-4 (1977), pp. 355–360.

[16] B. GRÜNBAUM, *Convex Polytopes*, Wiley Interscience, London, 1967.

[17] P. HANJOUL, H. BEGUIN AND J. C. THILL, *Shapes of market areas under Euclidean distance*, Rep., Univ. de Louvaine-La-Neuve, Belgium, 1984.

[18] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[19] D. G. KIRKPATRICK, *Efficient computation of continuous skeletons*, Proc. 20th. Ann. IEEE Symp. FOCS, (1979), pp. 18–27.

[20] E. KOCH, *Wirkungsbereichspolyeder und Wirkungsbereichsteilungen zu kubischen Gitterkomplexen mit weniger als drei Freiheitsgraden*, Zeitschr. für Kristallographie, 138 (1973), pp. 196–215.

[21] D. T. LEE, *Two-dimensional Voronoi diagrams in the L_p metric*, J. Assoc. Comput. Mach., 27 (1980), pp. 604–618.

[22] ———, *On k-nearest neighbor Voronoi diagrams in the plane*, IEEE Trans. Comp., C-31, 6 (1982), pp. 478–487.

[23] D. T. LEE AND C. K. WONG, *Voronoi diagrams in L_1 (L_∞) metrics with two dimensional storage applications*, this Journal, 9 (1980), pp. 200–211.

[24] G. C. LEKKERKERKER, *Geometry of Numbers*, Wolters-Noordhoff, Groningen, 1969.

[25] J. LINHART, *Die Beleuchtung von Kugeln*, Geom. Dedicata, 10 (1981), pp. 145–154.

[26] ———, *Dirichletsche Zellenkomplexe mit maximaler Eckenanzahl*, Geom. Dedicata, 11 (1981), pp. 363–367.

[27] P. MCMULLEN, *The maximal number of faces of a convex polytope*, Mathematica, 17 (1970), pp. 179–184.

[28] I. PASCHINGER, *Konvexe Polytope und Dirichletsche Zellenkomplexe*, Dissertation, Inst. of Math., Univ. Salzburg, Austria, 1982.

[29] F. P. PREPARATA AND S. J. HONG, *Convex hulls of finite sets of points in two and three dimensions*, Comm. ACM, 20 (1977), pp. 87-93.

[30] F. P. PREPARATA AND D. E. MULLER, *Finding the intersection of n halfspaces in time $O(n \log n)$*, Theoret. Comput. Sci., 8 (1979), pp. 45-55.

[31] C. A. ROGERS, *Packing and Covering*, Univ. Press, Cambridge, 1964.

[32] R. SEIDEL, *A convex hull algorithm optimal for point sets in even dimensions*, M. S. thesis, Rep. 81-14, Dep. Computer Science, Univ. of British Columbia, 1981.

[33] ———, *On the size of closest point Voronoi diagrams*, Rep. F 94, IIG, Tech. Univ. of Graz, Austria, 1982.

[34] F. W. SMITH, *The structure of aggregates and the molecular viscosity of a Bernal liquid*, Canad. J. Phys., 42 (1964).

[35] M. I. SHAMOS AND D. HOEY, *Closest point problems*, Proc. 16th Ann. IEEE Symp. FOCS, (1975), pp. 151-162.

[36] G. F. VORONOI, *Nouvelles applications des paramètres continues à la théorie des formes quadratiques, Deuxième Mémoire: Recherches sur les parallélloedres primitifs*, J. Reine Angew. Math., 134 (1908), pp. 198-287, 136 (1909), pp. 67-181.

[37] T. ZASLAVSKY, *Facing up to arrangements: Face-count formulas of partitions of space by hyperplanes*, Memoirs Am. Math. Soc., 154, Providence, 1975.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
007

A TIME-SPACE TRADEOFF FOR ELEMENT DISTINCTNESS*

A. BORODIN†[1], F. FICH‡[1], F. MEYER AUF DER HEIDE§[1], E. UPFAL¶ AND A. WIGDERSON*[1]

Abstract. In *A time space tradeoff for sorting on non-oblivious machines*, Borodin et al. [J. Comput. System Sci., 22 (1981), pp. 351-364] proved that to sort n elements requires $TS = \Omega(n^2)$ where T = time and S = space on a comparison based branching program. Although element distinctness and sorting are equivalent problems on a computation tree, the stated tradeoff result does not immediately follow for element distinctness or indeed for any decision problem. In this paper, we are able to show that $TS = \Omega(n^{3/2}\sqrt{\log n})$ for deciding element distinctness (or the sign of a permutation).

Key words. time-space tradeoffs, computational complexity, time lower bounds, space lower bounds

AMS(MOS) subject classification. 68Q25

1. Introduction. Time-space tradeoffs are one of the more classical issues in complexity studies. Cobham's [C-66] seminal paper establishes such tradeoffs for the restricted Boolean model of one-tape Turing machines. A number of time-space tradeoffs were established for both the Boolean and arithmetic circuit models (see Tompa [T-80]). Within these models, merging was essentially as difficult as sorting. For the problem of sorting $\{x_1, x_2, \cdots, x_n\}$ from an arbitrary linearly ordered set, the appropriate model is the comparison based branching program. Such programs are labelled directed acyclic graphs derived from comparison computation trees by identifying common subtrees. Following Cobham, space (or capacity) is measured as the $\log_2$ (# *of nodes*) in the program and time (as for computation trees) is the length of the longest path. Borodin, Fischer, Kirkpatrick, Lynch and Tompa [B-81] established the "near optimal" bound of $TS = \Omega(n^2)$ for sorting but were not able to establish a similar result for any decision problem, conjecturing that such a result should hold for the problem of determining element distinctness (i.e. $f(x_1, x_2, \cdots, x_n) = true$ iff $x_i \neq x_j$ for all $i \neq j$).

We are not able to establish the result as conjectured, but we are able to show $TS = \Omega(n^{3/2})$. Since there is no need to output during the computation, we need to find an appropriate notion of progress and show how this progress is constricted by a bound on the space. It turns out that once the appropriate progress notion is made, the basic outline of proof parallels the development in the sorting result.

2. The model and proof of the main result. Let $\{x_1, x_2, \cdots, x_n\}$ be elements chosen from an arbitrary linear order $[D, \leqq]$. A comparison branching program is a labelled directed acyclic graph. Each nonsink node has outdegree three and is labelled by a comparison $x_i : x_j$, with one branch for each of the three possible outcomes $<$, $=$, $>$. The sinks are labelled accept and reject. An input $\langle x_1, x_2, \cdots, x_n\rangle \in D^n$ follows a path in a program P in the obvious way determined by the comparisons, and we define acceptance of a set $L \subset D^n$ as usual.

We consider the set $L = \{\langle x_1, x_2, \cdots, x_n\rangle \mid x_i \neq x_j\}$. We say that $\langle x_1, x_2, \cdots, x_n\rangle \in L$ is ordered by the permutation π if $x_{\pi(1)} < x_{\pi(2)} < \cdots < x_{\pi(n)}$. It is clear that π determines

* Received by the editors August 29, 1985; accepted for publication (in revised form) March 13, 1986.
† University of Toronto, Toronto, Ontario, Canada M5S1A4.
‡ University of Washington, Seattle, Washington 98195.
§ Johann Wolgang Goethe Universität Frankfurt a. M., Federal Republic of Germany.
¶ IBM Research Laboratory, San Jose, California 95120.
* Mathematical Science Research Institute, Berkeley, California 94720.
[1] This work was done while the authors were visiting the IBM Research Laboratory, San Jose, California 95120.

the comparison path τ that $\langle x_1, x_2, \cdots, x_n\rangle$ follows in any comparison tree or branching program. Clearly, τ must terminate in an accepting node. Moreover, it is easy to see that τ must contain a comparison for every "adjacent" pair $x_{\pi(i)}:x_{\pi(i+1)}$, $1 \leqq i \leqq n-1$. Otherwise, we can set $x_{\pi(i+1)} = x_{\pi(i)}$ and still follow the same path τ (and erroneously accept). The number of adjacent pairs tested on a path will measure the progress for any π. Once we establish the following main lemma, we can follow the same proof structure as in Borodin et al. [B-81].

MAIN LEMMA. *Let T be a comparison computation tree of height $(=$ time$)$ t. If $t \leqq \sqrt{ns/16e}$ then for all S the fraction p of input permutations π for which π follows a path containing more than S comparisons of adjacent pairs is bounded by $p \leqq (\frac{1}{4})^S$.*

Proof of Main Lemma. Let τ be a computation path of length $t \leqq \sqrt{sn/16e}$. There are at most $r \leqq 2t$ elements x_i that are involved in some comparison in τ. τ determines a partial order on r elements. We will compute the fraction of permutations π following τ for which we have made at least S comparisons $x_{\pi(i)}:x_{\pi(i+1)}$. Let σ be any total order of the r elements consistent with τ. We bound the fraction for each σ as follows:

There are at most $\binom{n}{r}$ ways to assign ranks to the accessed elements. We now bound the number of rank assignments under the constraint that at least S adjacent pairs have been tested. There are $\binom{t}{S}$ ways to choose the S pairs and then $\binom{n}{(r-S)}$ ways to assign ranks to the r elements. (If $x_j < x_k$ is one of the S adjacent pairs, fixing the rank of x_j fixes the rank of x_k.)

Hence,

$$p \leqq \frac{\binom{t}{S}\binom{n}{(r-S)}}{\binom{n}{r}} = \binom{t}{S}\frac{r(r-1)\cdots(r-S+1)}{(n-r+S)\cdots(n-r+1)}$$

$$\leqq t^S \frac{r^S}{(n-r)^S} \leqq (4e)^S\left(\frac{t^2}{sn}\right)^S \leqq \left(\frac{1}{4}\right)^S. \qquad \square$$

With the main lemma now established, we proceed exactly as in Borodin et al. [B-81].

THEOREM. *Let P be a time T, space S comparison branching program for deciding element distinctness on n elements. Then $T^2S = \Omega(n^3)$ and $TS = \Omega(n^{3/2}\sqrt{\log n})$.*

Proof. Consider P in stages, where each stage represents $t = \frac{1}{4}\sqrt{n}$ steps. Without loss of generality, $S \geqq \log n$, since each of the possible comparisons appears at least once in the program, which therefore has at least $\binom{n}{2}$ nodes.

Let q_i be the fraction of input permutations (of distinct elements) for which P has compared at least iS adjacent pairs by the end of the ith stage. Using the main lemma, we will show that $q_i \leqq i(\frac{1}{2})^S$. Hence $q_i < 1$ for $i \leqq n/S$. This in turn implies that there must be at least n/S stages so that $T \geqq (n/4S)\sqrt{n}$. Otherwise, there will be a permutation for which some adjacent pair has not been tested, forcing a contradiction as previously explained. Thus $T^2S = \Omega(n^3)$. As $S \geqq \log n$, $TS = \Omega(n^{3/2}\sqrt{\log n})$.

To establish the claimed bound for q_i, we can consider each of the 2^S nodes at the end of stage i to be the root of a *subtree* of height t. That is, expand the branching program for stage $i+1$ into at most 2^S computation trees. For each such tree at most a fraction $(\frac{1}{4})^S$ of all $n!$ permutations can have more than S adjacent pairs tested. (Note that this is independent of the permutations that actually arrived at the root of this tree.) By the main lemma, the fraction of permutation for which P compared at least S adjacent pairs in the jth step is $\geqq (\frac{1}{2})^S$. Hence, the fraction of permutations for which P compared at least iS adjacent pairs in the first i stages is $i(\frac{1}{2})^S$. $\square$

It is easy to see that the same proof establishes the same bound for determining the sign of a permutation. In this case, if not all adjacent pairs have been tested, we can change the ordering of some pair and thereby change the sign.

3. The obvious open questions. Since we do not know how to improve the known upper bound of $TS = O(n^2 \log n)$ for element distinctness (this upper bound holding even for sorting [B-81]), the most obvious open question is whether $TS = \Omega(n^2)$. To see why the existing proof techniques do not extend, we note that we are only exploiting the space bound at the start of each stage. If we allow a stage to be $O(n)$ steps (as in the sorting lower bound), then we can decide distinctness in $O(\log^2 n)$ stages using only constant width at the beginning of each stage. (Simply partition the elements into $O(\log n)$ blocks of size $O(n/\log n)$, and then each stage can test a pair of blocks for distinctness by sorting.) We still conjecture that $TS = \Omega(n^2)$ holds for deciding distinctness but any proof will have to frequently exploit the space bound.

Perhaps the most important extension of our results is to the Boolean branching program model. In Borodin and Cook [BC-82] the sorting lower bound is extended to the setting of sorting n integers in the range $[1, n^2]$, the input represented by a string of $2n \log n$ bits. This lower bound is made possible since we have an explicit notion of progress, namely how many ranks have been established. Our comparison lower bound exploits the fact that we could also find a "monotone" concept of progress, that of "adjacent pairs tested." For the Boolean or "R-way model" (see Borodin and Cook [BC-82]) this concept does not apply. The difficulty of this issue is realized when one notes that in the comparison model, "silent sorting" (i.e. being able to infer the sort from the comparison paths) is equivalent to deciding element distinctness (see Reingold [R-72]) whereas in the Boolean or R-way model, silent sorting is trivial (simply look at each input once). Thus far, we still have not established a nontrivial time-space lower bound for any specific decision problem in the Boolean setting.

Acknowledgments. We wish to thank D. Coppersmith and M. Tompa for comments on an earlier version of this paper including an improvement of $\sqrt{\log n}$ to our previous lower bound.

REFERENCES

[BC-82] A. Borodin and S. Cook, *A time-space tradeoff for sorting on a general sequential model of computation*, this Journal, 11 (1982), pp. 287–297.

[B-81] A. Borodin, M. Fischer, D. Kirkpatrick, N. Lynch and M. Tompa, *A time-space tradeoff for sorting on non-oblivious machines*, J. Comput. System Sci., 22 (1981), pp. 351–364.

[C-66] A. Cobham, *The recognition problem for the set of perfect squares*, Research Paper RC-1704, IBM Watson Research Center, Yorktown Heights, NY, April 1966.

[R-72] E. Reingold, *On the optimality of some set algorithms*, Assoc. Comput. Mach., 19 (1972), pp. 649–659.

[T-80] M. Tompa, *Time-space tradeoffs for computing functions using connectivity properties of their circuits*, J. Comp. System Sci., 20 (1980), pp. 118–132.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
008

THE COMPLEXITY OF PARALLEL SORTING*

FRIEDHELM MEYER AUF DER HEIDE† AND AVI WIGDERSON‡

Abstract. The model we consider is the (concurrent-write, PRIORITY) PRAM. It has n synchronous processors, which communicate via an infinite shared memory. When several processors simultaneously write to the same cell, the one with the largest index succeeds. We allow the processors arbitrary computational power.

Our main result is that sorting n integers requires $\Omega(\sqrt{\log n})$ steps in this strong model. This bound is proved in two stages. First, using a novel Ramsey theoretic argument, we "reduce" sorting on a PRAM to sorting on a parallel merge tree. This tree is a generalization of Valiant's parallel comparison tree from [V] in which at every step n pairs of (previously ordered) sets are merged (rather then n pairs of elements compared). The second stage is proving the lower bound for such trees.

The Ramsey theoretic technique, together with known methods for bounding the "degree" of the computation, can be used to unify and generalize previous lower bounds for PRAM's. For example, we can show that the computation of any symmetric polynomial (e.g. the sum or product) on n integers requires exactly $\log_2 n$ steps.

Key words. parallel computation, PRAM, sorting, lower bounds, Ramsey theory

AMS(MOS) subject classification. 68Q25

1. Introduction.

The interest in parallel sorting is obvious. In the last few years we have witnessed a multitude of upper bounds for parallel sorting on various models. These results culminated in the remarkable $O(n \log n)$ sorting network of Ajtai, Komlos and Szemeredi [AKS], and its beautiful adaptation to bounded degree n-node networks by Leighton [L].

Surprisingly, there are essentially no lower bound results for sorting. The reason seems to be that for many parallel models such bounds follow either from simple considerations or from parallel lower bounds on simpler functions. Here are a few examples.

1. Comparison trees and algebraic decision trees. The $\Omega(n \log n)$ sequential lower bound for sorting (Paul and Simon [PS]) implies an $\Omega(\log n)$ lower bound on the parallel versions of these trees with n processors.
2. Bounded degree n-node networks. An $\Omega(\log n)$ lower bound follows from diameter considerations.
3. Polynomial size, unbounded fan-in circuits. The $\Omega(\sqrt{\log n})$ on depth for computing the parity function, due to Ajtai from [A], yields a similar bound for sorting.
4. Exclusive-write PRAM. The $\Omega(\log n)$ lower bound for computing the Or function, due to Cook, Dwork and Reischuk from [CDR] implies a similar bound for sorting.

In all the parallel models mentioned above, there is a restriction on either the computational power of the individual processors, or on the nature of communication between them. Here we pose no such restrictions; we consider a concurrent-write PRAM, in which n processors of arbitrary computational power communicate via an infinite shared memory. Moreover, we use the strongest convention of resolving write conflicts; the processor with the largest index succeeds.

The only nontrivial lower bound on this model appears in [FMRW]. It is a (tight) $\Omega(\log \log n)$ bound on finding the maximum of n integers. As is almost traditional

* Received by the editors May 8, 1985; accepted for publication (in revised form) February 18, 1986.
† J. W. Goethe Universität, Frankfurt, West Germany.
‡ Mathematical Sciences Research Institute, Berkeley, California 94720.

with lower bounds for models in which the computation and communication behaviour depend on the input in a complicated way, Ramsey theory is used to "clear the smoke" and find some structure. Such an argument showed that, for finding the maximum, a parallel comparison tree is essentially (i.e. for a large input set) as good as a PRAM. Then Valiant's $\Omega(\log \log n)$ lower bound from [V] on such trees is applied.

The problem with the Ramsey theoretic argument in [FMRW] is that input variables are fixed to constants at a doubly exponential rate as the computation proceeds, and hence it can never yield a lower bound bigger than $\Omega(\log \log n)$.

We prove an $\Omega(\sqrt{\log n})$ on sorting n integers. We use a new Ramsey theoretic argument in which variables are not fixed, but instead some information about their ordering is given to the algorithm as the computation proceeds. With this argument we show that essentially all a processor can do in a step is to merge two ordered sets (chains) of variables. It is natural now to define a parallel (m, n)-merge tree, in which at every node n pairs of chains of size m each are merged. (Note that for $m = 1$ this tree is simply Valiant's parallel comparison tree.)

We prove a general lower bound, parametrized by m, for such trees. Then we combine it with the dependency of m on the time of the PRAM computation to obtain our $\Omega(\sqrt{\log n})$ lower bound. We believe that our lower bound on PRAM's can be improved to $\Omega(\log n)$.

As mentioned, our Ramsey theoretic argument gives information about the ordering of input variables to the algorithm. For sorting, we had to take care to give as little information of this type as possible. For symmetric functions, however, applying this argument becomes much simpler, since we can assume that the input variables are sorted to begin with. Using this idea, combined with known techniques to making the "degree" of the computation finite (albeit the infinite address space), we get a simple proof of the following result. The computation of any symmetric polynomial of n integers (e.g. the sum or the product) requires exactly $\log_2 n$ steps.

This result was independently obtained (at least for addition) by Israeli and Moran in [IM]. It unifies and generalizes similar bounds on models that restrict either the computational power of processors (Meyer auf der Heide and Reischuk [MR]) or the interprocessor communication (Parberry [P]).

2. Formal definitions and statements of results. The two important models in this paper are the PRAM and the parallel merge tree. We start with the (standard) definition of the PRAM.

A (PRIORITY) PRAM M consists of n processors (RAM's) $P_1, P_2, \cdots, P_n$, and infinitely many shared memory cells (which we abbreviate "cells"), indexed by the integers N. We say that M computes a function $f: N^n \to N^m$ in T steps, if initially P_i has the value a_i of the ith input variable in its local memory, all cells are initialized to "0", and after T steps the first m cells contain the m values of $f(a_1, a_2, \cdots, a_n)$.

Every step of the computation consists of two phases, synchronously performed by all processors. In the write phase, each processor writes some value into some cell. In the read phase, each processor reads some cells to its local memory. (These addresses and this write value depend on the processor's local memory, but we place no restriction on the complexity of computing them.) If a cell is written into by several processors at one step, its contents will be the value written by the highest index processor among them.

For the definition of the merge tree, we need some notation for partial orders. Let V be a set of elements $\{x_1, x_2, \cdots, x_n\}$. A directed acyclic graph $Q = (V, E)$ defines a partial order $<_Q$ in the natural way. $\bar{Q} = (V, \bar{E})$ is the *transitive closure* of Q. A set

$U \subset V$ is a *chain* (or a *total order*) in Q if the elements of U belong to a directed path in Q. A set U is an *antichain* in Q if U is an independent set in $\bar{Q}$.

Let $U_1, U_2 \subset V$ and $Q_1 = (U_1, E_1)$, $Q_2 = (U_2, E_2)$ be partial orders. Q_1 and Q_2 are *consistent* if there is no pair $a, b \in V$ such that $(a, b) \in \overline{E_1}$ and $(b, a) \in \overline{E_2}$. A family of partial orders on subsets of V, $Q_1, Q_2, \cdots, Q_k$ is *consistent* if every pair is consistent. The *union* of two consistent orders Q_1 and Q_2 is $Q_1 \cup Q_2 = (V, E_1 \cup E_2)$. Similarly we define $\bigcup_{i=1}^{k} Q_i$.

We say that $Q' = (V, E')$ is an *extension* of $Q = (V, E)$ iff $\bar{E} \subset \bar{E}'$. If Q' is a total order, we say that it is a *linear extension* of Q.

Let $C_1, C_2 \subset V$ be two chains in $Q = (V, E)$. A *merge* of C_1 and C_2 in Q is any total order $Q' = (C_1 \cup C_2, E')$, consistent with Q.

Now we are ready to define the model. An (m, n)-*merge tree* (for sorting) is a rooted tree with labels on the nodes and on the edges. The label of an internal node v is a partial order Q on V, together with n pairs of chains in Q, (C_{i1}, C_{i2}), $i = 1, 2, \cdots, n$, satisfying $|C_{ij}| \leqq m$ for all $i = 1, 2, \cdots, n$, $j = 1, 2$. Every branch e out of v is labeled with a different consistent family of merges C_i of C_{i1} and C_{i2}, $i = 1, 2, \cdots, n$. If u is the child of v connected by e, then the partial order in u is $Q \cup \bigcup_{i=1}^{n} C_i$. The root is labeled with the empty partial order on V. The cost of an (m, n)-merge tree H, denoted $c(H)$, is the length of the longest root-leaf path in H. H sorts n numbers, if each leaf is labeled with a total order. The cost of sorting on (m, n)-merge trees is $c(m, n)$, the minimum cost $c(H)$ of an (m, n)-merge tree H which sorts n numbers.

The main result of this paper is the following.

MAIN THEOREM. *Any* PRAM *needs* $\Omega(\sqrt{\log n})$ *steps to sort n integers.*

The main Theorem follows directly from the Theorems 1 and 2 below.

THEOREM 1. *Let* M *be a* PRAM *sorting n integers in T steps. Then there is a* $(2^T, n)$*-merge tree which sorts n numbers in T steps.*

THEOREM 2.

$$c(m, n) = \Omega\left(\frac{\log n}{\log (m \log n)}\right).$$

As a corollary to the proof method of Theorem 1, we get a general tight bound for a family of symmetric functions. For $S \subset N$ let $S_*^n = \{(a_1, \cdots, a_n) \in S^n \mid a_i \neq a_j \text{ for } i \neq j\}$. A function $f: N^n \to N$ is *strongly nonconstant*, if for each infinite $S \subset N$, f restricted to S_*^n is nonconstant.

THEOREM 3. *A* PRAM *needs exactly* $\log_2 n$ *steps to compute a symmetric, strongly nonconstant function.*

Example. Every nonconstant, symmetric polynomial on n variables (e.g. the sum or the product) is a symmetric, strongly nonconstant function.

3. Simulating PRAM's by merge trees.

THEOREM 1. *Let M be a* PRAM *sorting n integers in T steps. Then there is a* $(2^T, n)$*-merge tree which sorts n elements in T steps.*

We simulate M by a computation tree. We want to maintain at each node v of this tree in depth t the following property: Let I be the set of inputs arriving at v. Then, for inputs from I, each processor P_i only knows variables with indices from some set $X_i \subset [n]$ at time t. ($[n] = \{1, 2, \cdots, n\}$.) The exact meaning of this is that for inputs from I, the configuration of P_i after t steps (considered as a function in I) only depends on the variables with indices from X_i. In the sequel we shall refer to them as the variables from X_i for short.

Clearly, because of its arbitrary computational power, P_i can now sort the variables it knows in one step and can proceed dependent on their order. We shall see that this

is essentially the only property of the input which can influence the behaviour of P_i, if we restrict the input set suitably. Namely, if the orders of the sets of variables the processors know are fixed, the next "communication pattern" is fixed, too, in the following sense. For each $i \in [n]$, P_i reads the value that was written by some fixed processor P_{j_i} at some fixed time t_i. Thus it is fixed which new variables P_i gets to know in this step, namely just those which P_{j_i} knew before. As they also were already sorted before, the behaviour of P_i in the next step only depends on the outcome of the merging of the two ordered sets of variables it now knows. Thus M behaves like a merge tree.

In the sequel we only consider inputs which consist of distinct numbers. Recall that for a set $S \subset N$ $S^n_* = \{(a_1, \cdots, a_n) \in S^n \mid a_i \neq a_j \text{ for } i \neq j\}$. For $i \in [n]$ let $X_i \subset [n]$ and π_i be a total order on X_i. Let $X = (X_1, \cdots, X_n)$, $\pi = \bigcup_{i=1}^n \pi_i$. We always assume that the π_i's are consistent. Then, following the above intuition, the set of inputs arriving at a node of the merge tree should be of the form $I(X, \pi, S) = \{(a_1, \cdots, a_n) \in S^n_* \mid a_p < a_q \text{ for all } p, q \text{ such that } p <_\pi q\}$.

The following lemma is the heart of the proof of the theorem.

MAIN LEMMA I. *Suppose that at time t, for inputs from $I = I(X, \pi, S)$, P_i only knows variables from X_i. Then there are $j_1, \cdots, j_n \in [n]$ and an infinite $S' \subset S$, such that after step t, for inputs from $I' = I(X, \pi, S')$, P_i only knows variables from $X_i \cup X_{j_i}$.*

Before we prove this lemma we first conclude Theorem 1 from it. For this purpose we define inductively a $(2^t, n)$-merge tree H of depth t. We shall show that finally, when H has depth T, it sorts n numbers. For the inductive construction to work, we keep extra information at the nodes.

The set of inputs at the root is $N^n_* = I((\{1\}, \cdots \{n\}), (\phi, \cdots, \phi), N)$, where ϕ denotes the trivial order on one element. At this time P_i only knows x_i. So the assumption in the lemma holds for $t = 0$.

Now let $t \in [T]$ and assume that we have constructed a $(2^t, n)$-merge tree of depth t for inputs from S^n_*, such that the set of inputs arriving at a node v in depth t is of the form $I = I(X, \pi, S)$, and for inputs from I, P_i only knows variables from X_i at time t. Furthermore we maintain that although X and π depend on v, S is the same set for all nodes in depth t.

For each node v in depth t successively perform the construction below.

Let $I = I(X, \pi, S)$ be the set of inputs arriving at v, and $S', j_1, \cdots, j_n$ be as in the main Lemma I. Now we define a son v' of v for each consistent tuple of mergings $\pi' = \bigcup_{i=1}^n \pi'_i$ of the sets in $X' = (X'_1, \cdots, X'_n)$ with $X'_i = X_i \cup X_{j_i}$. Then the set of inputs arriving at v' is $I' = (X', \pi', S')$. As $I' \subset I(X, \pi, S')$, we know by main Lemma I that, for inputs from I', P_i only knows variables from X'_i at time $t+1$. Now for the already defined sets of inputs arriving at nodes replace S by S'. Finally, after having performed this construction for all nodes in depth t, we get a $(2^{t+1}, n)$-merge tree of depth $t+1$.

Now suppose that we have constructed H up to depth T. We finally have to verify that H sorts n numbers, i.e. to prove that for each set of inputs $I = I(X, \pi, S)$ arriving at a leaf of H the order on $[n]$ induced by π is total. Suppose it is not. Then, as for inputs from I each P_i only knows variables from X_i at time T, M cannot distinguish between the different possible total orders and would compute the wrong output for some inputs. Thus H is a merge tree as demanded in the theorem.

Before we prove main Lemma I, we introduce some notations and useful Ramsey theoretic results. For an infinite set $S \subset N$ and a total order Q on $[n]$ let $S^n_Q = \{(a_1, \cdots, a_n) \in S^n \mid a_i < a_j \text{ if } i <_Q j\}$. Such a set is called a fixed order type. If Q is the natural order on $[n]$, we write $S^n_<$ for S^n_Q.

We apply the following "canonical" Ramsey theorem due to Erdös and Rado from [ER] (see also [GRS]).

THEOREM [ER] (Erdös-Rado Theorem). *Let $f: S_Q^n \to N$. Then there is $\tilde{S} \subset S$, $\tilde{S}$ infinite, such that $f' = f|_{\tilde{S}_Q^n}$ is 1-1 on the variables it depends on. Precisely, there is a $J \subset [n]$ such that $f'(a_1, \cdots, a_n) \neq f'(b_1, \cdots, b_n)$ if and only if $a_i \neq b_i$ for some $i \in J$. In particular, if f has a finite range, f' is constant.*

LEMMA 1. *Let $f: S_<^n \to N$ and $g: S_<^{n'} \to N$ be 1-1 functions. Then there is $\tilde{S} \subset S$, $\tilde{S}$ infinite, such that, restricted to $\tilde{S}_<^n$, f and g either have disjoint ranges or are identical. Precisely, either $f(\tilde{S}_<^n) \cap g(\tilde{S}_<^{n'}) = \varnothing$ or $n = n'$ and $f|_{\tilde{S}_<^n} \equiv g|_{\tilde{S}_<^n}$.*

Proof. Assume without loss of generality that $n \geqq n'$. Add dummy variables such that also g is defined on $S_<^n$, but only depends on the first n' variables. We first consider the 2-coloring c on $S_<^n$ with $c(\bar{a}) = 1$ if $f(\bar{a}) = g(\bar{a})$, and $c(\bar{a}) = 0$ otherwise. By the Erdös-Rado Theorem there is $\bar{S} \subset S$, $\bar{S}$ infinite, such that c is constant on $\bar{S}_<^n$. If $c \equiv 1$, then $f|_{\bar{S}_<^n} \equiv g|_{\bar{S}_<^n}$ and we are done. If $c \equiv 0$, then let G be the directed graph on $\bar{S}_<^n$ with $(\bar{a}, \bar{b}) \in E(G)$ if $f(\bar{a}) = g(\bar{b})$. G has no self-loops because $c \equiv 0$ on $\bar{S}_<^n$. G has indegree 1, because f is 1-1. Therefore it is easy to see that the underlying undirected graph is 3-colorable. Color it with 3 colors. By the Erdös-Rado Theorem there is an infinite $\tilde{S} \subset \bar{S}$ such that $\tilde{S}_<^n$ is monocromatic. Thus, $f(\tilde{S}_<^n) \cap g(\tilde{S}_<^n) = \varnothing$. Q.E.D.

Proof of Main Lemma I. Let $I = I(X, \pi, S)$ be such that P_i only knows variables from X_i at time t. Consider what P_i has done until time t. At each time $d \in [t]$, it wrote some value v_i^d to some cell w_i^d. Furthermore P_i read cell r_i at time t. These values are functions of the input set N^n. But, as P_i only knows variables from X_i, they only depend on the variables from X_i. Our goal now is to restrict the input set I to a set I' in such a way that, for inputs from I', P_i reads what some P_j wrote at some time d, only if r_i and w_j^d are the "same functions", and are applied to the "same arguments".

We refer to the functions w_i^d and r_i as address functions. The *clean form* of such a function f is derived by fixing all variables f does not depend on to arbitrary constants. Thus a clean form of a function always depends on all its variables. Let f' be the clean form of some address function f. If f was used by P_i (i.e. is r_i or w_i^d for some d), then it only depends on some (not necessary all) variables from X_i. As they are totally ordered according to π_i, f' is defined on a set of fixed order type. Thus we may apply the Erdös-Rado theorem successively to the clean forms of all address functions. The result is an infinite set $\tilde{S} \subset S$ such that, on $\tilde{I} = I \cap \tilde{S}^n$, all address functions are 1-1 on the variables they depend on.

From now on we assume that the domain of all address functions is $\tilde{I}$. Note that the clean form of the address functions can now depend on fewer variables than before we applied the Erdös-Rado theorem.

We know that the clean form of an address function is defined on a set of fixed order type. The function derived from it by reordering these variables such that it is defined on $\tilde{S}_<^n$ is called the *standard form* of f.

As we now know that the standard forms of address functions are 1-1 functions and are defined on $S_<^n$, we may apply Lemma 1 successively to all pairs of them. As a result we get an infinite set $\bar{S} \subset \tilde{S}$ with the following property (*). Assume from now on that the domain of all address functions is $I' = \tilde{I} \cap \bar{S}^n$.

(*) *Two address functions either have disjoint ranges or have the same standard form.*

Now let f and g be two address functions. Recall that they are defined on I'.

LEMMA 2. *f and g are either identical or $f(\bar{a}) \neq g(\bar{a})$ for all $\bar{a} \in I'$.*

Proof. Suppose $f(\bar{a}) = g(\bar{a})$ for some $\bar{a} \in I'$. Then, by (*), they have the same standard form h. Thus they are identical if they depend on the same variables. Assume that f and g depend on different sets of variables. Let $\bar{a}'$ and $\bar{a}''$ be the subvectors of the above $\bar{a}$ of those variables f and g depend on, increasingly ordered. These vectors

are different, because they contain values of different variables, and the a_i's are distinct. On the other hand, we know that $h(\bar{a}') = f(\bar{a})$, $h(\bar{a}'') = g(\bar{a})$. But as h is 1-1, $h(\bar{a}') \neq h(\bar{a}'')$, which contradicts the supposition we started with. Q.E.D.

Now consider what P_i reads in step t. If r_i was never used for writing (i.e. $r_i \neq w_j^d$ for all j and d), then we know by Lemma 2 that P_i reads 0. Otherwise, because of the PRIORITY write conflict resolution and Lemma 2, P_i reads v_j^d, the value P_j wrote at time d, where (d, j) is lexicographically maximal such that $w_j^d \equiv r_i$. Thus, as v_j^d only depends on variables from X_j, after this step P_i only knows variables from $X_i \cup X_j$ for inputs from I', which proves the lemma. Q.E.D.

4. A lower bound for sorting on merge trees. In this section we complete the proof of our main theorem by showing the demanded lower bound on (m, n)-merge trees.

THEOREM 2.

$$c(m, n) = \Omega\left(\frac{\log n}{\log (m \log n)}\right).$$

Proof of Theorem 2. It is sufficient to prove that $c(m, n) = \Omega(\log n / \log m)$ for $m \geqq 9 \log n$. Let H be any (m, n)-merge tree which sorts n numbers. We shall inductively exhibit a long path from the root in H. Intuitively, the argument is as follows. Suppose that we constructed the last node in the path, v, s.t. the partial order in v has many linear extensions. We shall look for a child u of v with the same property. To do that, we must show that there is a way of merging the n pairs of chains given at v, s.t. as few as possible transitive implications are added to the partial order.

To this end, we shall define a "nice" class of partial orders, in which we can get a handle on the quantities mentioned above. We will show that for each node in the path we construct, its partial order has an extension which is a partial order in the nice class.

We now describe the "nice" partial orders. Let $n = kl + a$ for positive integers k, l, a. A (k, l, a)-*graph* is obtained as follows. Take a chain on $k + a$ vertices. Then choose some k of the vertices, and replace each by l copies of itself (each having the same in and out neighbours).

A (k, l, a)-*partial order* is a partial order Q s.t. $\bar{Q}$ is isomorphic to the transitive closure $\bar{G}$ of some (k, l, a) graph G.

Let $Q = (V, E)$ be a (k, l, a)-partial order. It has k antichains, each of size l. Call them $S_1, S_2, \cdots, S_k$. Clearly, to "sort" Q, it is necessary and sufficient to sort all the S_i's. Consider a merge operation on Q. It involves two chains, C_1 and $C_2 (|C_1|, |C_2| \leqq m)$. Clearly, we may assume that $C_1, C_2 \subset \bigcup_{i=1}^{k} S_l$ (as the rank of the elements outside those antichains is known). Moreover, by the structure of Q, $|C_j \cap S_i| \leqq 1$ for all $j = 1, 2$ and $i = 1, 2, \cdots, k$. Hence, merging C_1 and C_2 reduces to performing at most m comparisons, at most one in each S_i.

MAIN LEMMA II. *Let v be a node in H whose partial order Q_v is contained in a (k, l, a)-partial order, with $a \leqq n/2$ (hence $kl \geqq n/2$). Then there is a child u of v whose partial order Q_u is contained in an (k', l', a') partial order, with $k' \leqq km^4$ and $a' \leqq a + (3n/m) + km^4$.*

Before we prove main Lemma II, let us see how it implies Theorem 2. Clearly, the empty partial order at the root of H is a $(1, n, 0)$ partial order. Apply the lemma t times with $t = \log n / 5 \log m$. We reach a node whose partial order is contained in a (k, l, a)-partial order with $k \leqq m^{4t}$ and $a \leqq 3tn/m + m^{4t}$. (Note that since $m \geqq 9 \log n$, and because of the choice of t, the assumption in the lemma holds at every step.) But for this choice of t, $k \leqq n/4$, $a \leqq n/2$, and therefore $l \geqq 2$. So this partial order cannot

be a total order, and this node at depth t cannot be a leaf. Hence, $c(H) = \Omega(\log n/\log m)$.

Before we proceed to prove Lemma 2, we need the following technical lemma.

LEMMA 3. *Let $G=(V, E)$ be an undirected graph, and b a positive number. Then one can remove a set $V' \subset V$ of vertices, with $|V'| \leqq (2|V|/b)+(2|E|b/|V|)$, s.t. the remaining graph on $V-V'$ can be colored with $(2|E|b/|V|)$ colors, with each color class of the same size.*

This lemma is based on the following result, due to Hajnal and Szemeredi from [HS].

THEOREM [HS]. *Any graph $G=(V, E)$ with maximum degree Δ can be colored with $\Delta+1$ colors s.t. each color class has size $(|V|/\Delta+1)$ or $(|V|/\Delta+1)+1$.*

Proof of Lemma 3. Remove the $2|V|/b$ vertices of highest degree from G. Then the maximum degree is at most $(2|E|b/|V|)-1$. Apply the previous theorem, and then remove one vertex from each of the larger color classes (at most $2|E|b/|V|$), to make them all of equal size.

Proof of Main Lemma II. Assume without loss of generality that Q_v is a (k, l, a)-partial order, and let $S_1, S_2, \cdots, S_k$ be its antichains of size l. Consider the n pairs of chains, (C_{i_1}, C_{i_2}), $i=1, 2, \cdots, n$, that are merged at v. Each such pair gives rise to at most m comparisons. So we have a total of nm comparisons, distributed among the S_j's. By averaging, at most k/m of the S_j's have more than nm^2/k comparisons. For each such S_j, arbitrarily choose a total order (hence resolving all comparisons within it). We obtain an extension Q_1 of Q_v which is a (k_1, l, a_1)-partial order, with $k_1 \leqq k$ and $a_1 \leqq a+(k/m)l \leqq a+(n/m)$.

So each of the remaining S_j's has at most nm^2/k comparisons. Think of these comparisons as edges in an undirected graph with vertex set S_j. Now we use Lemma 3 on each of these graphs. From each S_j remove $\leqq 2l/m+(2(nm^2/k)m/l)$ vertices as in the lemma, make them all (say) bigger than the rest of S_j, and arbitrarily totally order them. This resolves all comparisons involving these vertices. The result is an extension Q_2 of Q_1, which is a (k_2, l, a_2) partial order, with $k_2=k_1 \leqq k$ and $a_2 \leqq a_1+k((2l/m)+(2nm^3/kl)) \leqq a_1+2n/m+4km^3 \leqq a+3n/m+km^4$.

Finally, the remaining graph on each S_j can be colored with $\leqq 2nm^3/kl \leqq 4m^3 \leqq m^4$ colors each color class of equal size. Totally order the *color classes* arbitrarily, thus resolving the remaining comparisons in each S_j. The result is a (k', l', a') partial order Q', with $k' \leqq km^4$ and $a' \leqq a+3n/m+km^4$. Since we resolved all comparisons in a consistent way, there must be a child u of v s.t. Q' is an extension of Q_u.

5. A general lower bound for a family of symmetric functions. In this section we prove Theorem 3. Recall that for $S \subset N$ we defined $S^n_* = \{(a_1, \cdots, a_n) \in S^n \mid a_i \neq a_j \text{ for } i \neq j\}$. A function $f: N^n \to N$ is *strongly nonconstant*, if for each infinite $S \subset N$, f restricted to S^n_* is nonconstant.

THEOREM 3. *A* PRAM *needs exactly $\log_2 n$ steps to compute a symmetric, strongly nonconstant function.*

Proof. The upper bound is obvious because of the computational power of the processors: in $\log_2 n$ steps one processor can get to know all variables and then compute the function in one step.

In order to show the lower bound, we shall again apply the Erdös–Rado theorem to find an infinite $S \subset N$ such that M is oblivious to inputs from $S^n_<$.

We first describe what we mean by oblivious. Let $\bar{x} \in N^n$ be an input for M. Then it is well defined which processor writes to or reads from which memory cell at a given time, if M is started with $\bar{x}$. Thus we can define the following *communication pattern*

for $\bar{x}$: For each $t \in [T]$, $i \in [n]$, P_i reads at time t what P_j has written at time d, where $d \in [t]$ and $j \in [n]$ depend only on i and t. M is oblivious to inputs from $I \subset N^n$, if the communication patterns of all inputs from I are the same. The following lemma is often proved in similar forms in literature, e.g. in [MR], [CDR].

LEMMA 4. *Let $I \subset N^n$ such that $f: I \to N$ depends on all its variables. If the* PRAM *M computing f is oblivious to inputs from I, then M needs at least $\log_2 n$ steps to compute f.*

LEMMA 5. *If M computes $f: N^n \to N$ then there is an infinite $S \subset N$, such that M is oblivious to inputs from $S^n_<$.*

The lower bound in Theorem 3 now follows directly from the above lemmas and the definition of strongly nonconstant functions. As we only deal with symmetric functions, if such a function is nonconstant on S^n_*, then it is so even on $S^n_<$.

Proof of Lemma 5. We restrict f to inputs from $N^n_<$. As the number of processors and the number of steps M executes are finite, there are only finitely many different communication patterns. Thus, by the Erdös-Rado theorem, we find an infinite $S \subset N$ such that all inputs from $S^n_<$ have the same communication pattern, i.e. M is oblivious for inputs from $S^n_<$. Q.E.D.

REFERENCES

[A] M. AJTAI, *Σ^1_1-formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1-48.

[AKS] M. AJTAI, J. KOMLOS AND E. SZEMEREDI, *Sorting in c log n parallel steps*, Combinatorica, 3 (1983), pp. 1-19.

[CDR] S. COOK, C. DWORK AND R. REISCHUK, *Upper and lower time bounds for parallel random access machines without simultaneous writes*, preprint, 1983.

[ER] P. ERDÖS AND R. RADO, *A combinatorial theorem*, J. London Math Soc., 25 (1950), pp. 249-255.

[FMRW] F. FICH, F. MEYER AUF DER HEIDE, P. RAGDE AND A. WIGDERSON, *One, two, three $\cdots$ infinity: Lower bounds for parallel computation*, 16th ACM STOC, Providence, 1985, to appear.

[GRS] R. L. GRAHAM, B. L. ROTHSCHILD AND J. H. SPENCER, *Ramsey Theory*, John Wiley, New York, 1980.

[HS] A. HAJNAL AND E. SZEMEREDI, *Proof of a conjecture of P. Erdös*, Combinatorial Theory and its Applications, 2 (1970), pp. 601-623.

[IM] A. ISRAELI AND S. MORAN, private communication.

[L] T. LEIGHTON, *Tight bounds on the complexity of parallel sorting*, 16th ACM STOC, Washington DC, 1984, pp. 71-80.

[MR] F. MEYER AUF DER HEIDE AND R. REISCHUK, *On the limits to speed up parallel machines by large hardware and unbounded communication*, 25th IEEE FOCS, Miami, 1984, pp. 56-64.

[P] I. PARBERRY, *A complexity theory of parallel computation*, Ph.D. dissertation, Warwick, 1984.

[PS] W. J. PAUL AND J. SIMON, *Decision trees and random access machines*, Symposium ueber Logik und Algorithmik, Zürich, 1980, pp. 331-339.

[V] L. VALIANT, *Parallelism in comparison problems*, this Journal, 4 (1975), pp. 348-355.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
009

A NOTE ON BOTTOM-UP SKEW HEAPS*

DOUGLAS W. JONES†

Abstract. In testing Sleator and Tarjan's skew heap implementations of priority queues, a problem with the bottom-up variant was found. The original definition of skew heaps includes the assumption that the keys of items in the heap are disjoint. When this is not true, the *delete min* operation on bottom-up skew heaps will occasionally discard items from the heap. Modified versions of the *delete min* algorithm are presented which do not require this assumption.

Key words. priority queue, heap, skew heap

AMS(MOS) subject classifications. 68P10, 68Q20, 05-04

1. Introduction. While performing the empirical tests reported in [1], a problem was encountered with the bottom-up variant of Sleator and Tarjan's skew heaps [2]. The following problem was observed: some of the items inserted in a bottom-up skew heap were never found by the *delete min* operation; that is, items were lost from the heap. This problem can be traced to the definition of a heap (or priority queue) used in § 2 of [2]: "a set of items selected from a totally ordered universe." This implies that no two items in the heap have the same key! In practical applications of heaps, for example, in finding shortest paths in a graph, or in discrete event simulation, the keys of items are occasionally equal.

The notation used in the original presentations of skew heaps identified each node by its key; in order to allow discussion of nodes with equal keys, it is necessary to distinguish between a node's identity and its key. Thus, in the following discussion, identifiers such as x and y will be used to refer to nodes, while their keys will be referenced as $key(x)$ and $key(y)$. The original code for the operations on skew heaps can be trivially converted to allow equal keys by preserving the old notation except when comparing the relative order of two nodes. Thus, comparisons for equality, such as $x = y$, remain unchanged, while comparisons such as $x > y$ are changed to $key(x) > key(y)$.

On inspection of the *insert* and *delete min* operations given in § 3 of [2], the problem was found to be in the latter operation. The original version of the bottom-up *delete min* operation is shown here with the notational changes discussed above (labels are referenced in the text; typographical errors in the original are corrected on lines 3 and 4):

```
    function delete min(var h);
        var x, y1, y2, h3;
        if h = null → return null fi;
1:      y1, y2 := up(h), down(h);
2:      if key(y1) < key(y2) → y1 ↔ y2 fi;
3:      if y1 = h → h := null; return y1 fi;
        [initialize h3 to hold y1]
        h3 := y1; y1 := up(y1); up(h3) := h3;
        do true →
4:          if key(y1) < key(y2) → y1 ↔ y2 fi;
5:          if y1 = h → h := h3; return y1 fi;
            [remove x = y1 from its path]
```

* Received by the editors March 13, 1986; accepted for publication June 16, 1986.
† Department of Computer Science, University of Iowa, Iowa City, Iowa 52242.

```
        x := y; y1 := up(y1);
        [add x to the top of h3 and swap its children]
        up(x) := down(x); down(x) := up(h3); h3 := up(h3) := x
      od
end delete min;
```

The loss of data when equal keys are allowed happens on lines 3 and 5. In both cases, the variables y_1 and y_2 refer to the current nodes on the major and minor paths (the right branches of the two subtrees of the structure). In each case, the previous line (2 or 4) ensures that, in terms of the ordering of the key values, y_1 is farther from the root than y_2; thus, if y_1 refers to the root, y_2 must also refer to the root so the merge must be finished.

If items in the heap have equal keys, the above correctness argument does not hold. If y_1 is the same as h, $key(y_2)$ will equal $key(h)$, but that does not imply that y_2 refers to the same node as h. It is possible that y_2 refers to a descendant of h, so long as the keys of all nodes between y_2 and h are equal. In this case, y_2 and all nodes between it and h will be lost from the heap!

2. A solution. The simplest solution to this problem is to replace the comparisons on lines 2 and 4 in the original code with more complex comparisons which treat the root as having a lower key than any other item. The following code fragment will do this:

```
2:  4:  if (y1 = h) or (key(y1) < key(y2)) → y1 ↔ y2 fi;
```

Although this solution introduces a relatively small change in the code for *delete min*, it introduces an extra comparison into each iteration. This can be avoided if there is a key value, *min*, which is less than any key which will be encountered in practice. In this case, the key of the minimum item in the heap can be set to this value before starting the *delete min* operation, and reset to its original value when the work is done. This requires three assignments per visit to *delete min*, and it requires an additional local variable k to save minimum key. These are used by the following modifications to lines 1, 3 and 5 in the original code:

```
1:  y1, y2 := up(h), down(h); k := key(h); key(h) := min;
3:  if y1 = h → h := null; key(y1) := k; return y1 fi;
5:  if y1 = y → h := h3; key(y1) := k; return y1 fi;
```

3. A faster solution. Another solution to this problem is to replace lines 3 and 5 in the original code with a search for the top of the path headed by y_2. Thus, instead of adding code to avoid data loss in the first place, code is added to recover the lost data after the body of *delete min* has finished. The following code fragments do this:

```
3:  if y1 = h →
        if y2 = h → h := null; return y1 fi;
        x := y2;
        do up(y2) ≠ h → y2 := up(y2) od;
        up(y2) := x; h := y2; return y1;

    fi;
5:  if y1 = h →
```

```
    if y2 = h → h := h3; return y1 fi;
    x := y2;
    do up(y2) ≠ h → y2 := up(y2) od;
    up(y2) := up(h3); up(h3) := x; h := y2; return y1;
fi;
```

This code simply patches the remainder of the path pointed to by y_2 into the structure, without attention to preserving the amortized bounds. The code used in the empirical measurements reported in [1] was a Pascal transliteration of this. In [1], keys were represented by randomly selected real numbers so equal keys were extremely unlikely. If equal keys are probable, a more complex solution will be required. The fundamental problem with the above code is that it will preserve long right paths in the tree because the search loops do not exchange the children of each node visited. The following code cures this problem:

```
3:  if y1 = h →
      if y2 = h → h := null; return y1 fi;
      [initialize h3 to hold y2]
      h3 := y2; y2 := up(y2); up(h3) := h3;
      do true →
        if y2 = h → h := h3; return y2 fi;
        [remove x = y2 from its path]
        x := y2; y2 := up(y2);
        [add x to the top of h3 and swap its children]
        up(x) := down(x); down(x) := up(h3); h3 := up(h3) := x;
      od;
    fi;

5:  if y1 = h →
      do true →
        if y2 = h → h := h3; return y2 fi;
        [remove x = y2 from its path]
        x := y2; y2 := up(y2);
        [add x to the top of h3 and swap its children]
        up(x) := down(x); down(x) := up(h3); h3 := up(h3) := x;
      od
    fi;
```

These changes result in an annoying lengthening of what was originally a rather elegant algorithm, but they result in a trivial increase in the algorithm's run-time. If keys are disjoint, a single additional comparison is introduced per *delete min* operation. If keys are not disjoint, an additional comparison is added in determining that the new code must be executed, but the traversal of the final path to the root in this code is actually faster than it would have been had the keys been disjoint!

REFERENCES

[1] D. W. JONES, *An empirical comparison of priority-queue and event-set implementations*, Comm. ACM, 29 (1986), pp. 300-311.

[2] D. D. SLEATOR AND R. E. TARJAN, *Self adjusting heaps*, this Journal, 15 (1986), pp. 52-69.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987

© 1987 Society for Industrial and Applied Mathematics
010

THREE FAST ALGORITHMS FOR FOUR PROBLEMS IN STABLE MARRIAGE*

DAN GUSFIELD†

Abstract. The stable marriage problem is a well-known problem of matching n men to n women to achieve a certain type of "stability;" the $O(n^2)$ time Gale-Shapley [GS] algorithm for finding two particular, but extreme, stable marriages (out of a possibly exponential number of stable marriages) is also well known. In this paper we consider four problems concerned with finding information about the set of all stable marriages, and with finding stable marriages other than those obtained by the Gale-Shapley algorithm. In particular, we give an $O(n^2)$ time algorithm which, for any problem instance of n men and n women, finds every man-woman pair that is contained in at least one stable marriage; we show that the same algorithm finds all the "rotations" for the problem instance in $O(n^2)$ time; we give an $O(n^2+n|S|)$ time and $O(n^2)$ space bounded algorithm (which is time and space optimal) to enumerate all stable marriages, where S is the set of them; and we give an $O(n^2)$ time algorithm to find the minimum regret stable marriage (the best marriage, as measured by the person who is worst off in it). We believe the previous best time bounds for these problems are respectively $O(n^4)$ (from a related problem in [K]), $O(n^3)$ [ILG], $O(n^3|S|)$ [K] and $O(n^4)$ [K]. The basic idea leading to the improved running times is to exploit theorems about the structure of stable marriages in order to avoid back-up and duplicated work inherent in earlier algorithms.

Key words. stable marriage, matching, combinational algorithm, partial order

AMS(MOS) subject classifications. 90C27, 68Q25

1. Introduction. The stable marriage problem is a well-known problem of matching n men to n women to achieve a certain type of "stability"; the Gale-Shapley [GS] algorithm for finding two particular, but extreme, stable marriages (out of a possibly exponential number of stable marriages) is also well known, and is the basis of a national system for matching hospitals to resident doctors [R]. In this paper we consider four problems concerned with finding information about the set of all stable marriages, and with finding stable marriages other than those obtained by the Gale-Shapley algorithm. In particular, we give an $O(n^2)$ time algorithm which, for any problem instance of n men and n women, finds every man-women pair that is contained in at least one stable marriage; we show that the same algorithm finds all the "rotations" for the problem instance in $O(n^2)$ time (rotations are central in efficiently finding the optimal or most "egalitarian" stable marriage [ILG], and in the efficient enumeration of all stable marriages); we give an $O(n^2+n|S|)$ time and $O(n^2)$ space bounded algorithm (which is time and space optimal) to enumerate all stable marriages, where S is the set of them; and we give an $O(n^2)$ time algorithm to find the minimum regret stable marriage (the best marriage, as measured by the person who is worst off in it). We believe the previous best time bounds for these problems are respectively $O(n^4)$ (from a related problem in [K]), $O(n^3)$ [ILG], $O(n^3|S|)$ [K] and $O(n^4)$ [K]. The basic idea leading to the improved running times is to exploit theorems (from [K], [MW], [IL] and this paper) about the structure of stable marriages in order to avoid back-up and duplicated work inherent in earlier algorithms.

* Received by the editors September 9, 1985; accepted for publication (in revised form) March 19, 1986. This work was partially funded by the National Science Foundation under grant MCS-81/05894.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520. Present address, Department of Electrical and Computer Engineering, Computer Science Division, University of California, Davis, California 95616.

2. Definitions and background results. An instance of the stable marriage problem consists of n men and n women, each of whom has a rank-ordered preference list of the n people of the opposite sex. A marriage M is a one-to-one matching of the men and the women. Marriage M is said to be *unstable* if there is a man m and a woman w who are not matched to each other in M, but who both prefer each other to their respective mates given in M. Such a pair is said to *block* M. A marriage that is not unstable is called *stable*. The fundamental theorem [GS] is that there is a stable marriage for any problem instance. It is known [K] that there can be an exponential number of stable marriages, and the problem of counting then is $\#P$-complete [IL].

The lattice of stable marriages. Let M and M' be two stable marriages, and let $\max_i(M, M')$ be the woman man i most prefers between his two assigned mates in M and M'. Let $\min_i(M, M')$ denote the other woman. Then $\max(M, M')$ is the mapping of each man i to $\max_i(M, M')$, and $\min(M, M')$ is the opposite mapping. We say that marriage M *dominates* marriage M' (from the perspective of the men) if and only if $M = \max(M, M')$, and a marriage X is *between* M and M' if and only if M dominates X and X dominates M', and X differs from both M and M'. It is surprising, but easy to show ([K], [GS84]) that $\max(M, M')$ and $\min(M, M')$ are both stable marriages. Hence, under the relation of dominance, the set of all stable marriages forms a lattice L where the join and union operations are the max and min operations above. The unique maximum (most dominant) element of L is called the *man optimal* marriage, and the unique (most dominated) element of L is called the *woman optimal* marriage. The man optimal marriage has the very strong property that for *every* man m, there is no stable marriage in which m is married to a woman he prefers to his mate in the man optimal marriage.

If dominance is defined from the women's point of view, and wmax and wmin are the max and min operations with respect to the women, then $\max(M, M') = \text{wmin}(M, M')$and $\min(M, M') = \text{wmax}(M, M')$, so the lattice obtained using wmax is an inverted copy of the lattice obtained using max. Hence the man optimal marriage is woman *pessimal* i.e. for every woman w, there is no stable marriage in which w is married to a man she prefers less than her mate in the man optimal marriage. Similarly, the woman-optimal marriage is man-pessimal. In this paper, dominance will always be from the men's point of view unless explicitly stated otherwise. The following is an immediate consequence of the above facts.

LEMMA 0 [K], [GS84]. *For any two stable marriages M and M', M dominates M' from the men's point of view, if and only if M' dominates M from the women's point of view.*

Gale–Shapley algorithm. The algorithm of [GS] finds the man optimal marriage, although it can, by relabeling, also find the woman optimal marriage. This algorithm will be the basis of two of the algorithms given in this paper, so we briefly review it here.

ALGORITHM GS

At the start of the algorithm, each person is *free* and becomes *engaged* during the execution of the algorithm. Once a woman is engaged she never becomes free again (although to whom she is engaged may change), but men can alternate between being free and being engaged. The following step is iterated until all men are engaged:

Choose a free man m, and have m *propose* to the highest (most preferred) woman w on his list, such that w has not already rejected m. If w is free, then w and m become engaged. If w is engaged to man m', then she rejects the man (m or m') that she least prefers, and becomes, or remains, engaged to the other man. The rejected man becomes, or remains, free.

When all men are engaged, the engaged pairs are said to be *mated* or *paired* and form the man optimal stable marriage.

Proof of the correctness of this method appears in many places [K], [GS], [L]. There are specializations of the above algorithm derived by imposing rules specifying which free man makes the next proposal. In the remainder of the paper, we will use the rule that if a man m has just been rejected, then man m makes the next proposal. With this rule, it is clear that the Gale-Shapley algorithm can be implemented to run in time $O(n^2)$.

Breakmarriage.

DEFINITION. Let M be a stable marriage. Let man m be married (paired) in M to woman w. The operation *breakmarriage* (M, m), developed in [MW], is defined as follows:

With the men and woman paired as in M, restart the Gale-Shapley algorithm by "breaking" the marriage of m and w. Man m is now free, and woman w is "semi-free"; she will only accept a new proposal from a man she prefers to m. Operation breakmarriage (M, m) begins with m proposing to the woman following w in his list, and this initiates a sequence of proposals, rejections and acceptances as given by the Gale-Shapley algorithm. Operation brakmarriage (M, m) terminates either when some man has been rejected by all women, or when w receives (and accepts) a proposal from a man m' she prefers to m; at that point w is engaged to m', and there are no free men. Note that during the entire running of breakmarriage (M, m) there is exactly one free man at any time, hence (unlike the Gale-Shapley algorithm above) the sequence of proposals is completely determined: the next proposal is always made by the unique free man.

The following is simple to prove:

LEMMA 1 [MW]. *If breakmarriage (M, m) terminates with all men engaged, then the engaged pairs form a stable marriage.*

We omit the proof, but the key observation is that no man gets a mate higher in his list as a result of breakmarriage (M, m), and no woman gets a mate lower in her list.

The following is the central theorem in [MW].

THEOREM 1 [MW]. *Every stable marriage M' can be obtained by a series of breakmarriage operations starting from the man optimal marriage M_0.*

The proof is simple, and we will not repeat it. However, the key point in the proof is that if M is any stable marriage which dominates M', and man m's mate in M' is different from his mate in M, then breakmarriage (M, m) either results in M' or in a stable marriage between M and M' (i.e. breakmarriage (M, m) does not move any man to a woman below his proper mate in M'). Hence M' can be derived from M_0 by successively (and arbitrarily) choosing a man who is not yet married to his mate in M', and executing a breakmarriage operation. Each such operation either results in M' or in a stable marriage that dominates M', but which is closer to it.

In order to make the transformation from M_0 to M' completely deterministic, we can impose an ordering on the men so that when a new breakmarriage operation must be started, we select the first man (in the ordering) who is not yet married to his intended mate in M'. Hence the set of proposals and their order is determined by M_0, M' and the order of the men. This will be useful in Algorithm A below.

We note two useful corollaries to Theorem 1.

COROLLARY 1. *If breakmarriage (M, m) results in marriage M', then M' dominates all marriages which are dominated by M and in which m is not married to his mate in M.*

COROLLARY 2. *If m is married in M to a woman other than his mate in the woman optimal marriage, then breakmarriage* (M, m) *terminates with a new stable marriage, i.e. no man is rejected by the all the women.*

Corollary 1 is proven simply by letting M'' be any stable marriage dominated by M where m is not married to his mate in M. The first step in transforming M to M'' is to execute breakmarriage (M, m); hence breakmarriage (M, m) results in a marriage which dominates M'' and in which m has a different mate than in M. This corollary reverses the orientation of Theorem 1: in the theorem, there is a known target marriage which is obtained from M_0 by successive breakmarriage operations, while with Corollary 1 we will do breakmarriage operations without having any target marriage in mind, and we will extract information from the results. Corollary 1 will be central in § 5, and Corollary 2 will permit a simpler exposition.

3. Finding all stable pairs in time $O(n^2)$.

DEFINITION. Given an instance of the stable marriage problem, a man-woman pair (m, w) is said to be a *stable pair* if and only if m is married to w in some stable marriage.

There is a fairly direct method [K] to test whether any given pair (m, w) is a stable pair. The pair is stable if and only if there is a stable marriage omitting m and w, where each man whom w prefers to m is mated to a woman he prefers to w, and where each woman whom m prefers to w is mated to a man she prefers to m. This can be tested by a direct modification of the preference lists and of the Gale-Shapley algorithm. This approach would then need $\Theta(n^2)$ executions of the modified algorithm, so the best resulting bound would be $O(n^4)$, although it would not only identify the stable pairs, but marriages which contain them.

Knuth [K] mentions the usefulness of knowing the nonstable pairs, and shows how the man optimal and woman optimal marriages can be used to identify some, but not necessarily all, nonstable pairs. In this section we give a method to exactly identify all stable pairs (hence all nonstable pairs) in $O(n^2)$ time (the same time needed to find the man and woman optimal marriages); this is also the best time bound that we know of to determine if even a single pair is stable.

THEOREM 2. *Let* M_0 *and* M_t *be the man optimal and woman optimal marriages respectively. Let* $M_0, M_1, \cdots, M_t$ *be a sequence of stable marriages such that for each* i *from* 0 *to* $t-1$, M_i *dominates* M_{i+1} *and there is no stable marriage between* M_i *and* M_{i+1}. *Then every stable pair appears in at least one of the marriages in the sequence.*

Proof. Let M_i and M_{i+1} be two consecutive stable marriages on the sequence, and let m be a man who is married to w_i in M_i and to $w_{i+1} \neq w_i$ in M_{i+1}. Of course, man i prefers w_i to w_{i+1}. Now let w be a woman who man m prefers to w_{i+1} but not to w_i. If there exists a stable marriage M in which m and w are married, then $M' = \min[M_i, \max(M, M_{i+1})]$ is also a stable marriage in which m and w are married; hence M' is different from both M_i and M_{i+1}. But then, since M_i dominates M' and M' dominates M_{i+1}, M' is between M_i and M_{i+1}, a contradiction. □

COROLLARY 3. *Let H be the Hasse diagram (the graph representing the transitive reduction) of the lattice of all stable marriages. Then the marriages along any path (directed by the dominance relation) in H between the man optimal and woman optimal marriages contain all the stable pairs.*

Figure 1a shows the problem instance given in [MW] and Fig. 1b displays the Hasse diagram of the set of all stable marriages. Corollary 3 and the facts about operations max, min, wmax, and wmin are easy to verify in this example.

We now show how to efficiently find all stable pairs by enumerating a short

Men's lists	Women's lists
1: 57126843	1: 53761284
2: 23754186	2: 86357214
3: 85146237	3: 15624873
4: 32741685	4: 87324156
5: 72513684	5: 64738125
6: 16758423	6: 28546371
7: 25763481	7: 75218643
8: 38457261	8: 74152368

FIG. 1a. *Preference lists of eight men and women.*

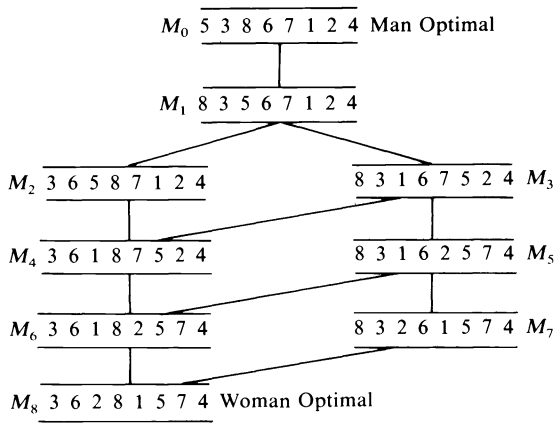


FIG. 1b. *The Hasse diagram of the set of all stable marriages. The number in position i of any list indicates the woman married to man i in that marriage.*

sequence of stable marriages which satisfy the conditions of Theorem 2. The particular marriages are obtained as a by-product of successive breakmarriage operations that transform the man optimal marriage into the woman optimal marriage. This method will later be used in the algorithm to find all stable marriages efficiently.

Algorithm A: Pausing breakmarriage. The following algorithm finds a sequence of stable marriages that satisfy the conditions of Theorem 2. The key algorithmic idea is to modify the breakmarriage operation so that it *pauses* at certain points where the next marriage in the sequence is output. In particular, the algorithm will pause when the proposal sequence generated by going from M_0 to M_t discovers a certain type of cycle called a *p-cycle* (we will see later that these are the rotations in [IL]). At each pause, the p-cycle is output, and the next marriage in the sequence is generated from the previous marriage by making changes dictated by the p-cycle. To more quickly understand the algorithm and its running time, it is helpful to keep in mind that the sequence of proposals, acceptances and rejections, is exactly the same sequence as used in transforming M_0 to M_t by successive breakmarriage operations, as discussed above, without pauses. The additional detail in the algorithm, which is interwoven into the proposal sequence, is used to extract and output p-cycles and the sequence of desired marriages.

ALGORITHM A

0. Set $i=0$; find and output the man optimal marriage M_0; find the woman optimal marriage M_t. All women are *unmarked* at this point.

1. If $M_i = M_t$ then stop. Else unmark any marked women and let m be the first man (in the fixed ordering of men) whose mate in M_i is different from his mate in M_t.

2. Set M to M_i. Let w be m's mate in M. Mark w and initiate breakmarriage (M, m); carry out (or continue, if returning from 3d) the sequence of proposals, acceptances and rejections as determined by breakmarriage (M, m), with the following modifications:

(a) When any unmarked woman *accepts a* proposal, mark that woman.

(b) Whenever a *marked* woman, say w' (which could be w), *receives* a proposal from a man, say m', whom she prefers to her mate in M_i, go to PAUSE breakmarriage (M, m).

Note that the comparison here is to the mate of w' in M_i, not to whom she is presently engaged to. Note also that the pause comes before w' decides to accept or reject the proposal from m'.

3. PAUSE: When breakmarriage (M, m) pauses do:

3a. Let $R(W)$ be the set containing w' and all the women who were marked since the most recent time that w' became marked; let $R(M)$ be the mates of $R(W)$ in M_i; and let *p-cycle* R_i be the ordered set of pairs consisting of $R(W)$ and their respective mates in $R(M)$, where the pairs are ordered in the order that the women in $R(W)$ were most recently marked.

3b. Let M_{i+1} be the marriage where w' is paired with m', where every other woman in $R(W)$ is paired to the man she is currently engaged to (as given in the current status of breakmarriage (M, m)), and all other women are paired to their mates in M_i. Output the p-cycle R_i and marriage M_{i+1}; set $i=i+1$ and unmark all women in $R(W)$.

3c. If $w = w'$, then let w' accept the proposal from m' (which completes breakmarriage (M, m)), and go to step 1.

3d. If $w \neq w'$, then let w' accept or reject the proposal from m' (by comparing m' to the man she is currently engaged to in the current status of breakmarriage (M, m)).

If w' rejects the proposal from m', then mark w'; return to step 2 and continue with breakmarriage (M, m) (i.e. m' next proposes to the woman on his list below w').

If w' accepts the proposal from m', then leave w' unmarked; go to step 2 and continue with breakmarriage (M, m) (i.e. the man who was engaged to w' when m' made his proposal is now free and makes the next proposal).

Note that by Corollary 2, each breakmarriage (M, m) operation, initiated in step 2, will ultimately finish with a new stable marriage (i.e. no man is rejected by all the women). It should also be clear that each M_i in the sequence is a marriage, although we must still demonstrate stability.

The five p-cycles output by Algorithm A on the problem instance given in Fig. 1 are $\pi_1 = \{(1, 5), (3, 8)\}$, $\pi_2 = \{(1, 8), (2, 3), (4, 6)\}$, $\pi_3 = (\{(3, 5), (6,1)\}$, $\pi_4 = \{(5, 7), (7, 2)\}$, $\pi_5 = \{(3, 1), (5, 2)\}$, where the first number in each pair in a p-cycle is a man and the second a woman. The p-cycles are listed in the order that the algorithm finds them, where the algorithm has used the given numerical order of the men. Notice that π_4 and π_5 were found in the running of a single breakmarriage operation, which started with man 3. All the other breakmarriage operations discovered exactly one p-cycle.

Before proving the correctness of Algorithm A, the following interpretation of the algorithm may be helpful, especially in explaining step 3d. Suppose m is mated to w in M_i and breakmarriage (M_i, m) pauses when a marked woman $w' \neq w$ is proposed to. Consider all of the proposals made from the start of breakmarriage (M_i, m) up to and including the first proposal that w' accepts; let the sequence of these proposals be called $P(w')$. Now consider breakmarriage (M_{i+1}, m). The key point to note is that breakmarriage (M_{i+1}, m) initially executes exactly the same sequence of proposals $P(w')$ in exactly the same order. The proposal immediately following $P(w')$ in breakmarriage (M_{i+1}, m) differs from the one in breakmarriage (M_i, m), and is, in fact, the next proposal made by Algorithm A in step 2, after returning from the pause in

breakmarriage (M_i, m) caused by the proposal to w'. Algorithm A can be thought of as an optimized algorithm that successively runs breakmarriages on $M_0, M_1, \cdots, M_{t-1}$, each until a p-cycle is encountered. The optimization makes sure that in each successive pair of breakmarriage operations no proposal in $P(w')$ is repeated. Step 3d of Algorithm A adjusts the mark of woman w' appropriately so that the last proposal in $P(w')$ is not repeated.

The following facts are easy to establish by examining the actions of Algorithm A. They will be needed in the proof of correctness and time.

Fact 1. The men and women who have different mates in M_k than in M_{k+1} are exactly the men and women in the pairs of R_k. The men in R_k (strictly) prefer their mates in M_k to their mates in M_{k+1}, and the women in R_k (strictly) prefer their mates in M_{k+1} to their mates in M_k.

Fact 2. If m_1 and m_2 are the men in any two consecutive pairs (in the circular order) of R_k, and they are married to w_1 and w_2 respectively in M_k, then m_1 is married to w_2 in M_{k+1}.

Fact 3. If m_1 and m_2 are as above, then w_2 is the first woman below w_1 on m_1's list such that w_2 prefers m_1 to m_2, her mate in M_k.

Time analysis and correctness of Algorithm A. With the exception of the time needed to output the marriages, Algorithm A runs in time $O(n^2)$, since step 1 is within this time bound, and since no man proposes to the same woman twice, and all other work is proportional to the number of proposals. In more detail: at any point in the algorithm the unique free man makes the next proposal which is to the next woman on his list; a linked list connecting the women in the order that they are marked allows R_i to be found in constant time per pair; and the total number of pairs in all the p-cycles is $O(n^2)$ since no pair is in more than one p-cycle (this follows from Fact 1). For the purpose of efficiently outputting the pairs which appear in the sequence of marriages, we can simply output the p-cycles and marriage M_t; each pair that appears in any of the marriages in the sequence is then output exactly once, and hence $O(n^2)$ time suffices to output these pairs.

In order to show that the output pairs are in fact stable, we need the following two lemmas:

LEMMA 2. *Each M_i found by the algorithm is a stable marriage.*

Proof. We prove this by induction. M_0 is stable and we assume that all marriages up though M_k are stable. Suppose M_{k+1} is not stable; then there is a man m and a woman w who block M_{k+1}. Since, by Fact 1, each woman either improves in M_{k+1} (over M_k) or keeps her same mate, m must be in a pair in R_k, otherwise m and w would block M_k. For the same reason, m cannot prefer w to his mate in M_k, so w must be strictly between (in order of preference) m's mate in M_k and his mate in M_{k+1}. Let B denote these women. But by Fact 3, none of the women in B prefer m to their mates in M_k, and so by Fact 1, none of these women prefer m to their mates in M_{k+1}. Hence M_{k+1} is stable. □

LEMMA 3. *There is no stable marriage between M_k and M_{k+1}.*

Proof. Suppose, to the contrary, that M is a stable marriage between M_k and M_{k+1}. We claim first that no man m can be married in M to a woman between his mate in M_k and his mate in M_{k+1}. Let w be such a woman between m's respective mates, and let m_w be the mate of w in M_k. By Fact 3, w prefers m_w to m, and since M_k dominates M and m_w is not married to w in M, m_w prefers w to his mate in M. Hence m_w and w block M. So if a stable M exists between M_k and M_{k+1}, then every man is either mated to his mate in M_k or to his mate in M_{k+1}, and there must be at

least one man of each type (else M is either M_k or M_{k+1}). Now in the circular order of pairs given in R_k, let m and m' be any two consecutive men in R_k, and let w and w' be their respective mates in M_k. Recall that w' becomes the mate of m in M_{k+1}. Hence it is not possible that in M, m marries his mate in M_{k+1} and m' marries his mate in M_k, since they both would then marry w'. Similarly, it is not possible that in M, m marries his mate in M_k and m' marries his mate in M_{k+1}, for then w' would be unmarried in M. But then either $M = M_k$ or $M = M_{k+1}$, and hence there is no marriage M between M_k and M_{k+1}. □

Hence by Lemmas 2 and 3, and Theorem 2:

THEOREM 3. *The marriages found by Algorithm A contain all the stable pairs, and all stable pairs can be found and output in time* $O(n^2)$.

3.1. Rotations and Algorithm A. Theorems 2 and 3 along with Algorithm A above showed that a small set of stable marriages contain all stable pairs. This fact, given by direct proofs above, also follows from a much deeper analysis of the structure of the set of all stable marriages, given in a powerful paper by Irving and Leather [IL]. Here we define rotations, the basic object in [IL], and show that Algorithm A finds each of them exactly once. Finding all the rotations is the first step in the algorithm of the next section which enumerates all stable marriages in time $O(n^2+n|S|)$; the $O(n^2)$ term contains the time used in Algorithm A to find the rotations. Finding all the rotations is also the first step in the algorithm in [ILG] to find the optimal or most egalitarian stable marriage.

Rotations.

DEFINITION. Let M be a stable marriage. For any man m let $S(m)$ be the first woman w' on m's list such that (i) m prefers his mate in M to w', and (ii) w' prefers m to her mate in M. Let $S'(m)$ be the man to whom $S(m)$ is married to in M.

DEFINITION [IL]. Let $R = \{(m_1, w_1), (m_2, w_2), \cdots, (m_z, w_z)\}$ be an ordered list of pairs from M such that for each i from 1 to z, $S'(m_i)$ is $m_{i+1(\mathrm{mod}\, z)}$. Then R is called a *rotation* (exposed in M).

Note that for a given marriage there may be many or there may be no exposed rotations.

Given an instance of the stable marriage problem, consider the set of all stable marriages for that instance, and consider the set of all rotations exposed by those marriages (any given rotation may be exposed by many marriages).

It is shown in [IL] that

THEOREM 4 [IL]. *Except for the stable pairs that are in the woman optimal marriage (which are in no rotations), each stable pair is in exactly one rotation, and, of course, each pair in a rotation is stable.*

The proof of Theorem 4 is fairly involved, and we refer the reader to [IL] for its motivation as well as its proof.

Finding all rotations in $O(n^2)$ time. Clearly, each p-cycle R_k is a rotation (exposed in M_k) and hence Algorithm A finds a set of distinct rotations[1] that contain all stable pairs other than those in the woman optimal marriage. Therefore, by Theorem 4, Algorithm A finds all rotations, and outputs each one exactly once, and so

THEOREM 5. *Given an n by n instance of the stable marriage problem, all the rotations can be found and output in* $O(n^2)$ *time.*

[1] We used the terminology "p-cycle" to avoid any confusion between what is derived directly from Algorithm A, and what is known about rotations from [IL].

Combined with Corollary 3, we have

THEOREM 6. *Let P be any path in H from the man optimal to the woman optimal marriage. Then any two consecutive marriages on P differ by a single rotation, and the set of the rotations between marriages along P contain all rotations exactly once.*

It is also now easy to see that any sequence of stable marriages which satisfy the conditions of Theorem 2 must lie on such a path P in H, and so, Algorithm A enumerates the marriages along some path in H from M_0 to M_t.

4. Enumerating all stable marriages in optimal time and space. McVitie and Wilson [MW] give an algorithm for enumerating all stable marriages in a problem instance with n men and n women. Their algorithm can be shown to take at least $\Omega(n^3|S|/(\log|S|^2))$ time, and no more than $O(n^3|S|)$ time, where S is the set of stable marriages. Knuth [K] describes a similar algorithm which has the same complexity.[2] Wirth [W] gives a different enumeration algorithm which is less efficient than these two.

In this section we give an algorithm for enumerating all stable marriages in $O(n^2+n|S|)$ time and $O(n^2)$ space. Considering the time needed just to output the marriages, and the space needed just to store the input preference lists, this time and space use is necessary; it is surprising that it is also sufficient. The algorithm depends critically on results in [IL], so we will first briefly review some of the results in [IL]. We will next modify the central construction given in [IL], and then combine these results with Algorithm A and the modified construction to obtain the enumeration algorithm. We also note that in the same time bound, the Hasse diagram of the lattice of all stable marriages can be explicitly constructed.

Partial orders and precedence graphs.

DEFINITION. For a given instance of the stable marriage problem, let π be a rotation exposed in stable marriage M, and let $M(\pi)$ be the marriage obtained by mating each man m in π with $S(m)$, and mating all men not in π with their mates in M. We say that π *moves* each man and women in π from their mates in M to their mates in $M(\pi)$. Note that a rotation always moves a man "down" his preference list, and always moves a woman "up" her preference list, and that the moves made by a rotation are independent of the marriage it is exposed in.

DEFINITION. A pair (m, w), not necessarily a stable pair, is said to be *eliminated* by rotation π if π moves w from m or below in her preference list to strictly above m.

Note that if (m, w) is a pair in π then π eliminates (m, w), and that if (m, w') is any other pair eliminated by π, then m prefers w to w', for otherwise no marriage in which π is exposed could be stable. Note also that if w is the woman most preferred by m such that (m, w) is eliminated by a rotation, then (m, w) is a stable pair, and is in the eliminating rotation.

LEMMA 4 [IL]. *No pair is eliminated by more than one rotation, and for any pair (m, w), at most one rotation moves m to w.*

This follows directly from the fact that Algorithm A finds all the rotations, as the rotations it finds obey the above claim.

Now we define the following relation between rotations:

DEFINITION [IL]. Let π and ρ be two distinct rotations. Rotation π is said to *explicitly precede* ρ if and only if π eliminates a pair (m, w), and ρ moves m to a woman w' such that m (strictly) prefers w to w'. The relation *precedes* is defined as the transitive closure of the relation "explicitly precedes."

[2] It is reported in [K] that the time is $O(n^2|S|)$, but this is incorrect. Constructions appear in [G] showing that the algorithm can take $\Omega(n^3|S|/[\log|S|^2])$ time.

If π explicitly precedes ρ, then ρ cannot possibly become exposed until π is eliminated. Hence it is easy to verify that the relation "precedes" defines a partial order on the rotations. In order to get some intuition for the importance of this relation, we claim (proofs follow from details in [IL]) that if π precedes ρ, then no matter how the men are ordered, Algorithm A finds π before it finds ρ. Hence in any transformation of M_0 to a marriage M by breakmarriage operations, the moves specified by rotation ρ will be made only if the moves specified by rotation π are made first. These claims are strengthened in the following definitions and theorem.

DEFINITION. Given an instance of the stable marriage problem, let D be a directed acyclic graph, where the nodes of D are in one-to-one correspondence with the set of rotations (we give each node the name of its corresponding rotation) and for any two nodes π and ρ, there is a directed edge from π to ρ if and only if rotation π precedes rotation ρ. Note that D may have $\Theta(n^2)$ nodes and $\Theta(n^4)$ edges.

We will often refer informally to a rotation π in D, instead of the node in D corresponding to rotation π; this should cause no confusion.

DEFINITION. A subset of rotations SN of D is *closed* if and only if SN contains all rotations which precede the rotations in SN.

The following is the central theorem of [IL]:

THEOREM 7 [IL]. *Let S be the set of all stable marriages for a given problem instance, and let D be the corresponding directed graph formed from the set of all rotations. Then there exists a one-to-one correspondence between S and the family of closed subsets in D, i.e. each closed subset in D specifies a distinct stable marriage, and all stable marriages are specified in this way.*

For a closed set SN the corresponding stable marriage is obtained by starting with M_0 and making the moves specified by the rotations in SN, in any order consistent with the precedence relations, i.e. the moves of notation π can be made only after the moves of all rotations which precede π.

Refining D. The algorithm to enumerate all stable marriages will enumerate each closed subset of D exactly once. The style of the enumeration (without concern for time and space complexity) is not new, and a similar method is implicit in [IL], although the question of generating all stable marriages is not explicitly discussed there. Enumeration methods in the same style appear in [TIA], [LLR] and [KS]. The new contributions here are several observations which allow the approach to run fast and in small space. In particular, explicit use of D does not lead to the $O(n^2+n|S|)$ time bound, since construction of D would take more than $O(n^2)$ time ($O(n^5)$ using fast transitive closure is possible), and the enumeration itself would take more than $O(n)$ time per marriage ($O(n^2)$ is possible). Further, D needs $\Theta(n^4)$ space just to store it, hence the $O(n^2)$ space bound could not be achieved using D. The main idea in this part of the paper is to use a sparse subgraph of D which preserves all the closed subsets, and which can be built quickly. It is not difficult to see that any subgraph of D whose transitive closure is D, preserves the closed subsets. We will construct such a subgraph G, in $O(n^2)$ time, with the property that G has $O(n^2)$ edges, and that no node in G has outdegree more than n. The bounded outdegree is the one of the keys to the $n|S|$ term in the time bound, and the sparsity is of course central to the space bound.

DEFINITION. G is a directed acyclic subgraph of D containing all the nodes of D but only edges defined by the following two rules, which are applied for each man m whose mate in M_0 is different than his mate in M_t:

Rule 1. Let $W(m)=\{w_0, w_1, \cdots, w_r\}$ be the set of women, in decreasing order of preference by m, such that for each i from 0 to r, (m, w_i) is a stable pair. For i from

0 to $r-1$, let π_i be the rotation containing pair (m, w_i), and let $\Pi(m)$ be the set of these rotations. Then for i from 0 to $r-2$, G contains an edge from π_i to π_{i+1}.

Rule 2. Suppose (m, w) is a nonstable pair eliminated by a rotation π, such that m prefers w to any other woman w' in any other pair (m, w') eliminated by π. If there are women w_i and w_{i+1} in $W(m)$ such that m prefers w_i to w and m prefers w to w_{i+1}, then G contains an edge from π to π_i.

Note that G is defined to contain only one copy of any edge, even though the same edge may be specified more than once by the above rules.

Figure 2 shows graph G constructed from the problem instance and rotations of Fig. 1; edge $\langle \pi_3, \pi_4 \rangle$ is defined by an application of Rule 2, while all the other edges are defined by Rule 1. The graphs are not always so tree-like as in this example.

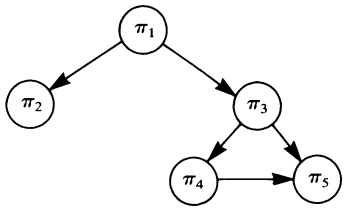


FIG. 2. *Graph G.*

LEMMA 5. *G has only $O(n^2)$ edges; it can be constructed in time $O(n^2)$; and no node in G has outdegree more than n.*

Proof. Given the rotations, which can be found in $O(n^2)$ time, we label each pair that is eliminated by some rotation with the name of the (unique) eliminating rotation. To do this, we examine each rotation π, and for each woman w in a pair in π we note the men that π moves w over; each of these pairs is labeled with π. Since π eliminates a set of pairs corresponding to a contiguous sequence of men in w's preference list, finding these pairs takes constant time per pair. Then, since no pair is eliminated by more than one rotation, these labelings can be done in $O(n^2)$ total time.

Now G can be constructed by processing each man m's list top down, starting from his mate in M_0. We keep a mark on the most recently encountered stable pair in m's list. When a new stable pair is encountered, we create an edge in G from the rotation labeling the marked pair to the rotation labeling the new pair, and we update the mark. When a nonstable pair is encountered, we check (in unit time using a random access list of the rotations) if its label has already been encountered in m's list. If not, then we create an edge in G from the rotation labeling the current nonstable pair, to the rotation labeling the current marked pair (there will be one). Each scan down a man's list takes $O(n)$ time, hence $O(n^2)$ time in total.

Since the total time to build G is $O(n^2)$, it can only have $O(n^2)$ edges. It is also clear from the details above that for any rotation π, the scan down a given man m's list adds at most one edge out of π; hence the outdegree of any node in G is bounded by n, the number of men. □

To complete our claims about G, we need the following lemma.

LEMMA 6. *For any two rotations π and ρ, π precedes ρ if and only if π reaches ρ by a directed path in G, hence the transitive closure of G is D, and so the closed sets of G and D are identical.*

Proof. Clearly, G is a subgraph of D since each edge in G specifies a precedence relation between the rotations at the endpoints of the edge. To prove the other direction,

it suffices to show that if π *explicitly* precedes ρ then π reaches ρ in G. By definition of "explicitly precedes," there must be two women w and w^* such that (m, w) is eliminated by π, and ρ moves m to w^*, and m prefers w to w^*. Then w^* is in $W(m)$, and $\rho \in \Pi(m)$; say $\rho = \pi_{i^*}$, where, by definition, ρ moves m from w_{i^*} to w^*. So in G there is a directed path from π_i to ρ for every $\pi_i \in \Pi(m)$ such that $i < i^*$.

Now let w' be the woman most preferred by m such that (m, w') is eliminated by π. By construction of G, there is an edge (associated with the pair (m, w')) from π to $\pi_{i'}$ for some $\pi_{i'} \in \Pi(m)$; let $w_{i'}$ be the woman that $\pi_{i'}$ moves m from. So if $i' \leqq i^*$ (i.e. $w_{i'}$ is equal to or is preferred to w_{i^*}), then there is a directed path in G from π to ρ. But m prefers $w_{i'}$ to w, and w to w^*, and since, by the actions of algorithm A, man m is moved over any particular woman by at most one rotation, w_{i^*} cannot be preferred to w'; hence $w_{i'}$ must either be w_{i^*} or be preferred to w_{i^*}, and the lemma follows. □

Note that G is not necessarily the transitive reduction of D. As stated above, any subgraph of D whose transitive closure is D, preserves the closed sets, and since we want a sparse subgraph, the transitive reduction of D would be the best. However, general algorithms to produce the transitive reduction of D (even assuming D is given) would take much more than the $O(n^2)$ time to construct G. Perhaps the transitive reduction of D can be computed from the rotations in $O(n^2)$ time for this special problem, but G is sufficient for the needs of this paper.

4.1. The enumeration algorithm. We will first describe how to use G to build a tree T with root r, where every edge in T is labeled with a rotation, such that the path from the root to any node in T enumerates a distinct closed set SN of rotations in G (and D), and such that each closed set in G is enumerated in this way. Hence by Theorem 7, there is a one-to-one correspondence between the nodes of T and the set of all stable marriages. Further, the order of the rotations along any path will be such that if π is a rotation on a given edge $e = \langle x, y \rangle$, then all rotations that precede (in the partial order of rotations) π will be on the path from the root to x. It follows inductively that the stable marriage corresponding to any node x can be explicitly constructed by starting at the root and successively executing the moves dictated by each rotation on the path to x. Since each such change takes $O(n)$ time, and each node in T corresponds to a distinct stable marriage, it follows that all the stable marriages can be output in $O(n)$ time per marriage, once T has been constructed. In obtaining the output, if T is traversed depth first, then only one complete marriage must be known at any time (the previous, as well as the next, marriage can be obtained from the marriage and the relevant rotation), hence only $O(n)$ additional space is needed for the traversal of T.

Building *T*. First, we label the rotations numerically according to a topological ordering of G, i.e. every node has a larger label than any of its predecessors. It is well known that these labels can be found in linear time in the number of edges of G, hence in $O(n^2)$ time. There are ways to avoid topological labeling, but the exposition becomes more complex.

To build T, we start at the root r and successively expand from any unexpanded node y in T as follows: Let $R(y)$ be the rotations along the path from r to y in T, and let $e = \langle x, y \rangle$ be the last edge on this path. Let $MR(y)$ be the set of maximal rotations (nodes in G with indegree zero) when all the rotations in $R(y)$ are removed from G, and let $LR(y)$ be those rotations in $MR(y)$ whose label is larger than the label on edge e. Then y is expanded by adding $|LR(y)|$ edges out of node y, each labeled with a distinct rotation in $LR(y)$.

LEMMA 7. *Given G, T can be constructed in $O(n)$ time per node.*

Proof. We give here more implementation detail on expanding a node. Let $e = \langle x, y \rangle$ be the last edge on the path to y, and let the rotation on e be π. We will assume, for now, that at node x in T, there is a graph $G(x)$, obtained from G by deleting all nodes in $R(x)$, and all incident edges. We also assume that the indegree of each node in $G(x)$ is known. Then $LR(y)$ is the set of all neighbors of π in $G(x)$ which have indegree 1 (note that these all have larger label than π due to the topological labeling of G), together with the set of rotations in $LR(x)$ whose label is larger than π. The first set can clearly be found in $O(n)$ time since no node in G (hence in $G(x)$) has outdegree more than n, i.e. there are at most n neighbors of π in $G(x)$. For the second set, we claim that $|LR(x)| \leqq n$, hence we can simply scan $LR(x)$ to find those rotations with label larger than π. To see that $|LR(x)| \leqq n$, note first that for any fixed m, if (m, w) and (m, w') are two pairs in (necessarily) distinct rotations, then one of these two rotations must precede (in the partial order) the other. But, by construction or induction, each pair of rotations in $LR(x)$ must be incomparable, and so for any man m, m is in a pair in at most one rotation in $LR(x)$.

So far, we have seen that if $G(x)$ is given at node x, then the edges out of x can be determined and labeled in time $O(n)$. However, constructing the graphs at each of the endpoints of these edges must be done with some care. For example, if T is built in a breath first manner, then $|LR(x)|$ graphs have to be constructed and stored. In addition to the enormous space this would require, it also would need more than $O(n)$ time per node, since the graphs can have $\Theta(n^2)$ nodes and edges. The solution is to expand T depth first: to expand a given node x in T, we find all the maximal elements in $G(x)$ and store them (essentially, constructing all the edges out of x), but we construct a new graph $G(y)$ for only one edge $\langle x, y \rangle$, labeled π, out of x; node y is the next node in T to be expanded. Graph $G(x)$ can be transformed into $G(y)$ in $O(n)$ time, by deleting node π and all incident edges from $G(x)$; the indegree in $G(y)$ of each neighbor of π is one less than its indegree in $G(x)$, and all other indegrees remain as in $G(x)$, so the indegrees are also maintained in $O(n)$ time. Backing up from y to x, we use $G(y)$ and the rotation on edge $\langle x, y \rangle$ to reconstruct $G(x)$ in time $O(n)$. Knowing $G(x)$ and the untraversed edges out of node x, we choose an unexpanded child y' of x, transform $G(x)$ into $G(y')$, and then expand y'. So T can be built in $O(n)$ time per node. □

COROLLARY 4. *Given G, the set of all stable marriage can be enumerated in $O(n)$ time per marriage, and $O(n^2)$ total space.*

Proof. The approach above was to first build T, and then to traverse it depth first to explicitly construct the stable marriages. However, T was built depth first in order to obtain the $O(n)$ time bound, so we can construct the stable marriages as we build T. But then, we never need to know the complete T at any given time. What is sufficient at any one time, staying close to the above details, is the path from r to the current node being expanded, and the edges (with their rotations) which directly hang off of that path. The depth of T is at most $O(n^2)$ since each edge on a path corresponds to a distinct rotation, and the outdegree of each node in T is $O(n)$, hence if we construct and output the stable marriages as T is (implicitly) being built, depth first, then we need only $O(n^3)$ space for T. However, all other space use is $O(n^2)$, and the total space bound could be reduced to that if we did not store the maximal elements of $G(x)$ at each node x. These had been stored to faciliate the backup to x, and the next traversal out of x, and to simplify the exposition. But, when backing up from node y to x, where edge $\langle x, y \rangle$ is labeled with rotation π, the maximal elements of $G(x)$ can be found from $G(y)$ in $O(n)$ time, since they are the maximal elements of $G(y)$, plus π, minus the neighbors of π in $G(x)$. So both $G(x)$ and its maximal elements can be recomputed

in $O(n)$ time on backup. However, we must be careful that no edge out of x is traversed more than once. There are several ways to do this. One simple way is to traverse the edges out of x in increasing order of their labels; each time we enter x we scan the maximal elements of $G(x)$ and choose the one with the smallest label larger than the label on the edge just used to enter x either backup or first entry). In this way, only a single path of T needs to be kept at any one time, hence the total space is $O(n^2)$, and the time remains $O(n)$ per node. □

Figure 3 shows the tree T built from graph G of Fig. 2. Each node in T labeled with the corresponding stable marriage from Fig. 1.

We still need to show that the nodes in T correspond one-to-one to the closed sets of G, and that the order of the rotations along a path in G has the desired properties claimed above. This is done in the following lemmas.

LEMMA 8. *Let x be an arbitrary node in T. Then $R(x)$ is a closed set of rotations in D (hence G).*

Proof. By induction on the length of the path to x. The lemma is clearly true for the root, which corresponds to the empty set, and for nodes at distance one from the root, for each of these corresponds to a maximal rotation in D. Now let x be a node at distance k from r, and let $\langle x, y\rangle$ be an edge out of x with label π. By inductive hypothesis, $R(x)$ is a closed set, and, by construction, π is maximal in $G(x)$, so all the predecessors of π are in $R(x)$, and hence $R(x)+\{\pi\}$ is a closed set in D, and this set corresponds to node y. □

COROLLARY 5. *The stable marriage corresponding to node y can be constructed by traversing the path from r to y, successively making the moves dictated by the rotations on the path.*

LEMMA 9. *Every closed set in D is $R(x)$ for some node x in T.*

Proof. By induction on the size of the set. The lemma is clearly true for size zero and one, since these sets are the empty set and the maximal elements in D. Now suppose the lemma holds for sets of size k, and let SN be a closed set of size $k+1$. SN must have a minimal element with respect to the partial order D; let π be the minimal element of SN with the largest label. By the induction hypothesis, $SN-\{\pi\}$

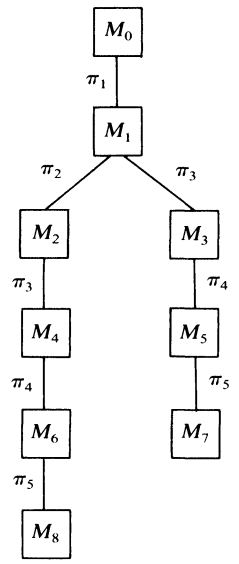


FIG. 3. *Tree T with rotations on edges, and stable marriages at nodes.*

is $R(x)$ for some node x in T. But then, π is a maximal node in $G(x)$, and since it has the largest label of the rotations in SN, it will label an edge $\langle x, y\rangle$ out of x. Hence SN is $R(y)$. □

LEMMA 10. *Let x and x' be two distinct nodes in T, then $R(x) \neq R(x')$; hence no closed set is enumerated twice in T.*

Proof. Consider a node x and two edges $\langle x, y\rangle$ and $\langle x, y'\rangle$ out of x labeled π and ρ, where π has a smaller label than ρ. Note that the labels along any path from r are in increasing order, hence π cannot appear in the subtree of T rooted at y'. The lemma follows by applying this observation inductively on the length of the paths. □

Constructing the Hasse diagram. We will not give any details here, but we claim that the Hasse diagram of the set of all stable marriages can also be explicitly constructed in $O(n^2 + n|S|)$ time, and (excluding the space for the lattice itself) with modest space. To see that this is plausible, note that each node in the Hasse diagram can have outdegree of at most n, since each node is associated with a stable marriage, and each edge out of the node is associated with a rotation exposed in that marriage, and there clearly can be no more than $n/2$ rotations exposed in any stable marriage. Hence the size of the lattice itself is at most $O(n|S|)$.

5. The minimum regret stable marriage problem.

DEFINITION. For stable marriage M containing the pair (m, w), the *regret* of m is the position of woman w in m's list, and the regret of w is the position of man m in w's list. The regret of a marriage M, denoted $r(M)$, is defined to be the maximum regret of any person, given the pairing in M, i.e. M is measured by the person who is worst off in it.

Knuth [K] discusses the problem of finding a stable marriage which minimizes $r(M)$ over all stable marriages. The solution given in [K] is attributed to Alan Selkow and naive analysis of it gives a running time of $O(n^4)$. Here we use the breakmarriage operation and Corollary 1 to obtain a method that runs in time $O(n^2)$. It is Corollary 1 that allows this speed up by avoiding duplicated proposals and rejections.

For ease of exposition, we will break the problem into two problems: find, if one exists, a marriage minimizing $r(M)$ over all stable marriages in which at least one woman is a person of maximum regret in the marriage, and find, if one exists, a marriage minimizing $r(M)$ over all stable marriages in which at least one man is a person of maximum regret in the marriage. Let the first type of marriage be called *woman regret* minimum, and the second type be called *man regret* minimum. Note that for a given problem instance, it is possible that in every stable marriage, all people with maximum regret are women (men) and hence there is no man (woman) regret minimum. This happens if and only if all people of maximum regret in the woman (man) optimal marriage are women (men). These cases are easy to check and adjust for in the algorithm so, for ease of the exposition, we will assume that neither of these two cases occur, and hence both a woman regret and a man regret minimum marriage exist. The minimum regret stable marriage is obtained from these two marriages. The following algorithm finds a woman regret minimum, assuming both a woman and a man regret minimum exist.

ALGORITHM B

0. Find the man optimal stable marriage M_0, and find the woman optimal stable marriage. Set $i = 0$.

1. Let w be a woman with regret $r(M_i)$ in M_i, and let m be her mate in M_i. If m and w are a pair in the woman optimal marriage, then stop and output M_i; M_i is a woman regret minimum. Else perform operation breakmarriage (M_i, m) and let M_{i+1} be the resulting stable marriage.

2. If there are no women with regret $r(M_{i+1})$ in M_{i+1}, then stop and output M_i; marriage M_i is a woman regret minimum. Else set $i = i+1$, and go to step 1.

Correctness of Algorithm B. First, the algorithm must terminate since, after each breakmarriage operation, any woman with a new mate prefers him to her previous mate, and any man with a new mate prefers his previous mate. So unless the conditions in step 1 apply, ultimately there will be no women with the maximum regret, and then the conditions of step 2 will apply. Note that by Corollary 2, each breakmarriage operation in the algorithm results in a stable marriage. Now if the algorithm terminates in step 1, then M_i is woman regret minimum, for there is no stable marriage in which w has a better mate than m. Hence we will assume that the algorithm terminates in step 2.

Let $M_0, M_1, \cdots, M_z$ be the sequence of marriages produced by the algorithm (hence the algorithm outputs M_{z-1}). To prove that M_{z-1} is woman regret minimum, we need the following two lemmas.

LEMMA 11. *For i from 0 to $z-2$, if M_i is woman regret minimum then so is M_{i+1}.*

Proof. Each woman either has the same mate in both marriages, or prefers her mate in M_{i+1}, and, by the algorithm, for i from 0 to $z-2$, there is a woman in M_{i+1} with regret $r(M_{i+1})$. □

LEMMA 12. *For every i from 0 to $z-1$, either M_i is woman regret minimum or it dominates all marriages which are.*

Proof. This is clearly true for M_0 since it dominates all marriages. Suppose the claim holds through M_k; let w be a woman with regret $r(M_k)$ in M_k, and let m be her mate in M_k. Now if M_k is not woman regret minimum, then w cannot be married to m in any marriage which is. Hence we know that M_k dominates all woman regret minimum marriages, and that m and w are not paired to each other in any of these marriages. But by Corollary 1, breakmarriage (M_k, m) results in a marriage which dominates all marriages which are dominated by M_k and in which m is not married to his mate in M_k. Hence M_{k+1} dominates all woman regret minimum marriages. Hence the lemma follows by inductively applying this argument until either $i = z-1$, or until M_i is woman regret minimum, where Lemma 11 applies. □

Now we can prove correctness of the algorithm.

THEOREM 8. *The marriage M_{z-1} is woman regret minimum.*

Proof. Let MW be a woman regret minimum marriage. Suppose M_{z-1} is not woman regret minimum and let (m, w) be a pair in M_{z-1} where w is a woman with regret $r(M_{z-1})$ in M_{z-1}. Then we know that M_{z-1} dominates MW (from Lemma 12), and that m is not married to w in MW (by the assumption that M_{z-1} is not woman regret minimum), so, by Corollary 1, M_z dominates MW, and, by Lemma 0, MW dominates M_z from the women's point of view. But, by the algorithm, there are no women among the people of maximum regret in M_z, and so (again by Lemma 0) there can be no women among the people of maximum regret in MW, since it dominates (from the women's view) M_z. But this is a contradiction, since at least one woman in MW must have regret $r(MW)$, or else MW is not even in the set of stable marriages over which the woman regret minimum is defined. Hence either M_{z-1} is woman regret minimum, or none exist, so, given our assumptions, the algorithm is correct. □

Implementation and time analysis of Algorithm B. The algorithm moves M_0 towards M_t using breakmarriage operations; hence the total number of proposals is $O(n^2)$. Step 0 of the algorithm clearly requires only time $O(n^2)$, but steps 1 and 2 must be implemented with some care in order to obtain an overall $O(n^2)$ time bound. In each

iteration of steps 1 and 2 the maximum regret of the men and of the women must be determined and compared, and a woman with overall maximum regret, if one exists, must be found. Simple scanning of the men and women at each iteration would lead to a bound of $O(n^3)(O(n)$ time per iteration, and $O(n^2)$ iterations). Below we sketch the details that give a time bound of $O(n^2)$ and a space bound of $O(n)$ (not counting the space for the preference lists).

At the start of each step 1, the status of the women in the current marriage will be represented by n linked lists, one for each level of regret, where each list i links together (in no particular order) all of the women with regret i in the current marriage. For each i, we let $c(i)$ be the number of women in list i; variable wr keeps the largest i such that $c(i) \neq 0$. We also need two n length vectors of pointers, one to point to the current location of each woman in the list that she is presently in, and one to point to the head of each list. Clearly all the lists and pointers take $O(n)$ space and can be initiated in $O(n)$ time. An identical data structure is kept for the men; $K(i)$ is the number of men in list i, and mr is the largest i such that $K(i) \neq 0$. Of course we also need to record who the pairs are in the current marriage, and other information needed to efficiently execute breakmarriage operations, but these details are assumed, since they are trivial and were needed in Algorithm A.

Given the above data structures, a woman of regret $r(M)$ in the current marriage M is found at the head of the women's list wr. After a breakmarriage operation, the women with new mates are removed from their current lists (in constant time per woman using the vector of pointers), and inserted at the heads of the appropriate new lists, and the variables $c(i)$ are adjusted. If $c(wr)$ is now zero, then i is decremented from wr until $c(i) \neq 0$ is found, and wr is updated. Since wr only decreases during Algorithm B, the overall time for this search is $O(n)$. The men's lists are similarly updated after the breakmarriage, but for any $k(i)$ which changes from zero to a positive count, mr is set to max (mr, i). Step 2 is implemented by comparing wr to mr. When computing the man regret minimum, the roles of men and women and their respective data structures are interchanged.

6. Open question. Consider the following problem: Find a fast algorithm to determine if an input marriage is stable. One obvious way to test for stability is to examine each man m to see if there is a woman w whom m prefers to his mate in M, such that w also prefers m to her mate in M. With the obvious storage of the preference lists, each check takes unit time. One can also check each woman's preferences in the above way, but note that we need only check from the perspective either of the men or from the perspective of the women, but not both. If we define $r(M, i)$ to be the number of people whom person i prefers to their mate in M, then stability can be checked from the perspective of the men in time $\sum_{\text{man } m} r(M, m)$. Naively, this could be as many as $n(n-1) = O(n^2)$ checks (table look-ups). Is there a way to beat this bound? Particularly, if preprocessing is allowed, say to build D, or some other "reasonable" work, can the $O(n^2)$ bound per marriage be reduced?

The worst case bound of $n(n-1)$ is, in fact, not optimal: no more than $n(n-1)/2$ checks are needed, and these can be found and done in that time.

LEMMA 13. *If M is a stable marriage, then* $\sum_{\text{man } m}[r(M, m)] + \sum_{\text{woman } w}[r(M, w)] \leqq n(n-1)$. *Hence one of* $\Sigma_m[r(M, m)]$ *or* $\sum_w[r(M, w)]$ *is less than or equal to* $n(n-1)/2$.

Proof. If M is a stable marriage and man m prefers woman w to his current mate, then woman w must not prefer man m to her current mate, and similarly, if woman w prefers m to her mate, them m must not prefer w to his mate. Hence the pair (m, w) can contribute at most one to $r(M, m) + r(M, w)$ and the lemma follows. □

Hence we can test for stability by first computing $\sum_m r(M, m)$ and $\sum_w r(M, w)$ (preprocessing permits us to construct the correct data structure so that this sum can be done in $O(n)$ time). If they sum to more than $n(n-1)$, then M isn't stable. If the sum is less than or equal to $n(n-1)$, then we check for stability from the perspective of the sex with smallest sum.

Acknowledgments. I would like to thank Dana Angluin, Gregory Sullivan and Robert Irving for listening to parts of this paper as it developed. I would like to especially thank Carrie Shepard for helping me to read [K], and for joining me in converting the theory of stable marriage into practice.

REFERENCES

[GS] D. GALE AND L. SHAPLEY, *College admissions and the stability of marriage*, Amer. Math. Monthly, 69 (1962).

[GS84] D. GALE AND M. SOTOMAYOR, *Some remarks on the stable matching problem*, Discrete Appl. Math., to appear.

[G] D. GUSFIELD, *Three fast algorithms for four problems in stable marriage*, Yale Computer Science Technical Report #407, July 1985.

[IL] R. IRVING AND P. LEATHER, *The complexity of counting stable marriages*, this Journal, 15 (1986), pp. 655-667.

[ILG] R. IRVING, P. LEATHER AND D. GUSFIELD, *An efficient algorithm for the optimal stable marriage problem*, July 1985.

[K] D. E. KNUTH, *Marriages Stable*, Les Presses de L'Université de Montreal, 1976. (In French.)

[KS] D. E. KNUTH AND J. SZWARZFITER, *A structural program to generate all topological sorting arrangements*, IPL, 2 (1974), pp. 153-157.

[L] E. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[LLR] E. LAWLER, J. LENSTRA AND A. RINOOY KAN, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, preprint.

[MW] D. MCVITIE AND L. WILSON, *The stable marriage problem*, CACM (July 1971).

[R] A. ROTH, *The evolution of the labor market for medical interns and residents: A case study in game theory*, J. Political Economy, 92 (1984).

[TIA] S. TSUKIYAMA, M. IDE AND M. ARIYOSHI, *A new algorithm for generating all the maximal independent sets*, this Journal, 6 (1977), pp. 505-517.

[W] N. WIRTH, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
011

ON THE COMPUTATIONAL COMPLEXITY OF ALGEBRA ON LATTICES*

H. B. HUNT III†, D. J. ROSENKRANTZ† AND P. A. BLONIARZ†

Abstract. We study the computational complexity of equivalence and minimization problems for expressions on many different lattices including each finite lattice and each distributive lattice. A general efficient expressibility condition C on a lattice is presented such that

1. The equivalence problem is *coNP*-hard for constant-free expressions on any lattice with at least two elements that satisfies condition C.

Each finite or distributive lattice is shown to satisfy condition C. Moreover, if a lattice $\mathcal{L}$ satisfies condition C and $\equiv$ is a congruence relation on $\mathcal{L}$, then $\mathcal{L}/\equiv$ also satisfies condition C.

Several additional results are also presented. These results include the following:

2. In contrast to 1, the equivalence and operator minimization problems are solvable deterministically in polynomial time for disjunctive normal form and conjunctive normal form expressions on any lattice and for constant-free expressions on any free lattice with at least three generators.

3. Let $\mathcal{L}$ be a lattice. Then, the operator minimization problem and various approximate operator minimization problems for expressions on $\mathcal{L}$ are as hard as the problem of determining, for expressions F and G on $\mathcal{L}$, if $F \leqq G$.

Key words. lattice, finite lattice, free lattice, distributive lattice, formula equivalence, formula minimization, computational complexity, *NP*-hardness, deterministic polynomial time

AMS(MOS) subject classifications. 68Q40, 68Q25, 06B, 06B25, 06D, 03G10

1. Introduction.

In this paper we study the computational complexity of equivalence and minimization problems for constant-free expressions on lattices. The lattices studied include each finite lattice and each distributive lattice. The problems studied include the following: letting $\mathcal{L}$ be a lattice,

1. determining, for expressions F and G on $\mathcal{L}$, if F and G are *equivalent*, i.e. denote the same function;

2. determining, for expressions F and G on $\mathcal{L}$, if $F \leqq G$;

3. given expression F on $\mathcal{L}$, calculating an equivalent expression G on $\mathcal{L}$ with minimum number of occurrences of operators.

Let N denote the set of natural numbers. Let $\Psi : N \to N$. We also study the problem of

4. given expression F on $\mathcal{L}$, calculating expression G on $\mathcal{L}$ such that the number of occurrences of operators in G is $\leqq \Psi(\text{minop}(F))$, where

$$\text{minop}(F) = \min \{ i \in N \mid \text{there is an expression } H \text{ on } \mathcal{L} \text{ equivalent to } F \text{ with } i \text{ occurrences of operators} \}.$$

We call problems 1-4 the *Equivalence Problem*, *$\leqq$-Problem*, *Operator Minimization Problem*, and the *Ψ-Bounded Approximate Operator Minimization Problem for $\mathcal{L}$*, respectively.

Our interest in the computational complexity of problems 1-4 for lattices is motivated as follows. First, lattices appear naturally throughout mathematics and computer science [4], [5]. For example, they appear in group theory [1], [5], algebraic

* Received by the editors October 29, 1984; accepted for publication (in revised form) January 13, 1986. A preliminary version of some of these results was presented at the Seventeenth Annual Conference on Information Sciences and Systems, March 1983. This research was partially supported by the National Science Foundation under grants MCS80-03353, DCR84-03014, MCS82-03237 and MCS78-04346.

† Computer Science Department, State University of New York at Albany, Albany, New York 12222.

logic [21], combinatorics and geometry [1], [5], [24], the logical design of digital systems [7] and the theory of data types [23]. Many of these lattices are not distributive, e.g. the *geometric* or *matroid lattices* [5], [24]. Second, lattices are some of the simplest algebraic structures defined in terms of binary operations. Their simplicity enables us to prove complexity results for problems 1-4 that hold for many different lattices. For example, we show that problems 1, 2 and 3 are *coNP*-hard for the *same* reason for constant-free expressions on any finite or distributive lattice with at least two elements. Third, lattices can be embedded into other algebraic structures such as rings in such a way as to extend lower bounds on the complexity of lattice computations into lower bounds on the complexity of computations on these other structures [16], [17]. Finally, our results on the complexity of problems 1-4 for lattices suggest inherent limitations on the practical feasibility of automated systems for manipulating and/or simplifying algebraic expressions on algebraic structures more general than the real or complex numbers [8], [19].

Previous work on the computational complexity of problems 1-4 for lattices has shown the following:

1. Problem 1 is *coNP*-complete for the two-element Boolean algebra [9].
2. Problems 1, 2 and 3 are *coNP*-hard, for any distributive lattice with at least two elements [6].
3. Problems 1, 2 and 3 are effectively solvable for the free lattice with 3 generators [5], [25], [26].
4. Problem 1 is undecidable for expressions with constants for several modular lattices [11], [18], [20].

New results presented here include the following:

(i) A general efficient expressibility condition C on a lattice is presented such that problems 1 and 2 are *coNP*-hard for constant-free expressions on any lattice with at least two elements satisfying C. Each finite or distributive lattice is shown to satisfy C. Moreover, if a lattice $\mathscr{L}$ satisfies C and $\equiv$ is a congruence relation on L, then the lattice $\mathscr{L}/\equiv$ also satisfies C.

(ii) Problems 1, 2 and 3 are solvable deterministically in polynomial time for disjunctive normal form and for conjunctive normal form expressions on any lattice and for constant-free expressions on any free lattice with $\geqq 3$ generators.

(iii) Let $\mathscr{L}$ be a lattice. Then, the $\leqq$-Problem for expressions on $\mathscr{L}$ is efficiently reducible to the Operator Minimization Problem and to various Approximate Operator Minimization Problems for expressions on $\mathscr{L}$. In particular, we show the following:

(a) Let $\mathscr{L}$ be a finite or distributive lattice with at least two elements; and let p be a polynomial. Then, there exists a deterministic or a nondeterministic polynomially time-bounded algorithm to solve the p-Bounded Approximate Operator Minimization Problem for expressions on $\mathscr{L}$ only if $P = NP$ or $NP = coNP$, respectively.

(b) Let $\mathscr{L}$ be a lattice for which the Equivalence Problem is undecidable; and let f be a recursive function. Then, there is no algorithm to solve the f-Bounded Approximate Operator Minimization Problem for expressions on $\mathscr{L}$.

The rest of this section consists of notation, definitions and basic results about strings, lattices, expressions and computational complexity.

We denote the *length* of a string S or the *cardinality* of a set S by $|S|$.

DEFINITION 1.1. Following [5], a *lattice* $\mathscr{L} = (S, \vee, \wedge)$ where S is a nonempty set and $\vee$ and $\wedge$ are binary operations on S that are commutative, associative, idempotent,

and such that for all $x, y \in S$,

$$x \vee (x \wedge y) = x \quad \text{and} \quad x \wedge (x \vee y) = x.$$

The set S is called the *domain* of the lattice $\mathcal{L}$. The binary operations $\vee$ and $\wedge$ are called the *join* and *meet*, respectively, of $\mathcal{L}$. A lattice $\mathcal{L}$ with domain S is said to be

(i) *nondegenerate* if $|S| > 1$,

(ii) *finite* if the set S is finite,

(iii) *distributive* if, for all $x, y, z \in S$, $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

In this paper, we will assume that all lattices are nondegenerate. For a degenerate lattice, the problems we consider are trivial, since all expressions are equivalent.

It is well known that lattices are closely related to *partially ordered sets* as follows. If $\mathcal{L}$ is a lattice with domain S, then $(S, \leqq)$ is a partially ordered set, where the binary relation $\leqq$ on S is defined by

$$\text{for all } a, b \in S, \quad a \leqq b \quad \text{if and only if } a \vee b = b$$

or, equivalently, is defined by

$$\text{for all } a, b \in S, \quad a \leqq b \quad \text{if and only if } a \wedge b = a.$$

If a lattice $\mathcal{L}$ has a *maximal* or a *minimal* element under the partial order $\leqq$, we denote this element by 1 or by 0, respectively.

DEFINITION 1.2. Let $\mathcal{L} = (S, \vee, \wedge)$ be a lattice. A subset T of S such that, for all $x, y \in T$, either $x \leqq y$ or $y \leqq x$ is called a *chain* (*of* $\mathcal{L}$). The *length* of a chain T equals $|T| - 1$ if T is finite, and equals ∞ otherwise. If there exists $n \in N$ such that some chain of $\mathcal{L}$ has length n and no chain of $\mathcal{L}$ has length $> n$, then the lattice $\mathcal{L}$ is said to be of *length n*. Otherwise, the lattice $\mathcal{L}$ is said to be of *length* ∞.

PROPOSITION 1.3 [1], [5]. *Let $\mathcal{L}$ be a lattice with domain S. Then for all $x, y, z \in S$,*

$$(x \wedge y) \vee (x \wedge z) \leqq x \wedge (y \vee z) \quad \textit{and} \quad (x \wedge z) \vee (y \wedge z) \leqq (x \vee y) \wedge z.$$

DEFINITION 1.4. Let $\mathcal{L}$ be a lattice with domain S. We denote the set of all expressions on $\mathcal{L}$ consisting only of variables, parentheses, and the binary operators $\vee$ and $\wedge$ by EXPR ($\mathcal{L}$). An expression $F \in \text{EXPR}(\mathcal{L})$ is said to be *constant-free.*

Constant-free expressions represent functions in the standard manner.

DEFINITION 1.5. Let $\mathcal{L}$ be a lattice with domain S. Let $F, G \in \text{EXPR}(\mathcal{L})$. We denote the function represented by the expression F by $\hat{F}$. Let α be an assignment of values from S to the variable symbols. We denote the value in S taken on by $\hat{F}$, given α, by $\alpha[F]$. We say that F and G are *equivalent on* $\mathcal{L}$, denoted by $F = G$ on $\mathcal{L}$, if $\hat{F} = \hat{G}$. We say that F is *less than or equal to* G on $\mathcal{L}$, denoted by $F \leqq G$ on $\mathcal{L}$, if for all assignments α of values from S to the variable symbols, $\alpha[F] \leqq \alpha[G]$.

It is easily seen that, for all lattices $\mathcal{L}$ and for all expressions F and G on $\mathcal{L}$,

(i) $F = G$ on $\mathcal{L}$ if and only if $F \leqq G$ and $G \leqq F$ on $\mathcal{L}$ if and only if $(F \vee G) \leqq (F \wedge G)$ on $\mathcal{L}$,

(ii) $F \leqq G$ on $\mathcal{L}$ if and only if $F \vee G = G$ on $\mathcal{L}$ if and only if $F \wedge G = F$ on $\mathcal{L}$.

Thus for all lattices $\mathcal{L}$, the Equivalence and $\leqq$-Problem for expressions on $\mathcal{L}$ are linearly reducible to each other.

DEFINITION 1.6. Let $\mathcal{L} = (S, \vee, \wedge)$ be a lattice. If $\equiv$ is an equivalence relation on S such that, for all $x, y \in S$,

(i) $x \equiv y$ implies $x \vee z \equiv y \vee z$ for all $z \in S$, and

(ii) $x \equiv y$ implies $x \wedge z \equiv y \wedge z$ for all $z \in S$,

then the relation $\equiv$ is said to be a *congruence* (*relation*) *on* $\mathcal{L}$. If a congruence on a lattice has at least two distinct equivalence classes, we say that the congruence is *nontrivial.*

Let $\mathscr{L}$ be a lattice. Let $\equiv$ be a congruence on $\mathscr{L}$. Let $F, G \in \mathrm{EXPR}(\mathscr{L})$. We write $F \equiv G$ if, for all assignments of values α from the domain of $\mathscr{L}$ to the variables,

$$\alpha[F] \equiv \alpha[G].$$

We assume that the reader is familiar with the computational complexity classes *P*, *NP* and *coNP*, otherwise see [2] or [13]. The concepts from computational complexity theory needed in this paper are *polynomial reducibility*, *NP-hardness* and *coNP-hardness* defined as follows.

DEFINITION 1.7. Let Σ and Δ be finite nonempty sets. Let L and M be languages such that $L \subset \Sigma^*$ and $M \subset \Delta^*$. We say that L is *polynomially reducible to* M if and only if there exists a function f from Σ^* to Δ^* computable deterministically in polynomial time (on a Turing machine) such that

$$\text{for all } x \in \Sigma^*, \quad x \in L \quad \text{if and only if } f(x) \in M.$$

We say that L is *linearly reducible* to M if L is polynomially reducible to M via a function f that is computable deterministically in linear time. We say that a language L is *NP-hard* or is *coNP-hard* if each language in *NP* or in *coNP*, respectively, is polynomially reducible to L. We say that a language is *NP-complete* or is *coNP-complete* if and only if

$$L \text{ is } NP\text{-hard and is an element of } NP,$$

or

$$L \text{ is } coNP\text{-hard and is an element of } coNP,$$

respectively.

One well-known observation about the complexity classes *P*, *NP* and *coNP* is the following.

PROPOSITION 1.8. *Let L be a coNP-hard language. Then L is an element of P only if P = NP = coNP; and L is an element of NP only if NP = coNP.*

It is generally assumed that each *NP*- or *coNP*-hard problem is likely to require deterministic exponential time, and thus, is likely to be computationally intractible. It is also generally assumed that $NP \neq coNP$, and thus, that no *coNP*-hard problem can be solved nondeterministically in polynomial time. This last assumption, if true, implies that no complete and soundproof procedure π exists for any *coNP*-hard set L such that π has the following two properties:

1. there is a polynomial $p(n)$ such that, for all x in L, there is a π-proof that x is in L of length $\leqq p(|x|)$,

2. the set of all π-proofs is recognizable deterministically in polynomial time.

Since most natural proof procedures have property 2, if $NP \neq coNP$ then generally there are no "short" proofs that x is in L, for any *coNP*-hard set L. Most of the problems considered here are shown to be *coNP*-hard. Thus, using term rewriting systems [15] to solve these problems is likely to take an exponential number of steps.

DEFINITION 1.9. A Boolean expression f is said to be a *Disjunctive Normal Form Boolean Expression* or DNF *Boolean Expression* if and only if f is of the form t_1 *or* $\cdots$ *or* $t_k (k \geqq 1)$, where each t_i is of the form t_{i1} *and* $\cdots$ *and* $t_{im} (m \geqq 1)$ and each t_{ij} is a variable or a negated variable. A Boolean expression f is said to be a *Conjunctive Normal Form Boolean expression* or CNF *Boolean Expression* if and only if f is of the form t_1 *and* $\cdots$ *and* $t_k (k \geqq 1)$, where each t_i is of the form t_{i1} *or* $\cdots$ *or* $t_{im} (m \geqq 1)$ and each t_{ij} is a variable or a negated variable.

In §§ 4 and 5, we show, for each lattice $\mathcal{L}$, that the Equivalence and Operator Minimization Problems are solvable deterministically in polynomial time for the following lattice-theoretical analogues of the DNF and CNF Boolean expressions.

DEFINITION 1.10. Let $\mathcal{L}=(S, \wedge, \vee)$ be a lattice. A DNF *expression f on* $\mathcal{L}$ is a constant-free expression on $\mathcal{L}$ that consists of a join of meets of variables. Dually, a CNF *expression f on* $\mathcal{L}$ is a constant-free expression on $\mathcal{L}$ that consists of a meet of joins of variables.

Most of the *NP*- or *coNP*-hard lower bounds in this paper follow from the next proposition from [9].

PROPOSITION 1.11. *The set* $\{f \mid f$ *is a tautological* DNF *Boolean expression*$\}$ *is coNP-complete. The set* $\{f \mid f$ *is a satisfiable* CNF *Boolean expression*$\}$ *is NP-complete.*

Finally in Theorem 5.6, we consider nondeterministic algorithms that *solve* Approximate Operator Minimization Problems. There are several possible reasonable definitions of what it means for a nondeterministic algorithm to *solve* such problems. The proof of Theorem 5.6 applies to the weak definition where each output-producing branch of a nondeterministic computation produces an acceptable equivalent expression, different branches can produce different acceptable expressions, and there is at least one output-producing branch for each valid input.

2. The efficient expressibility condition C. The class of sp_n-expressions ($n \geqq 1$) is introduced and one of its elementary properties is established. This property is used to derive a general efficient expressibility condition C such that, for each lattice $\mathcal{L}$ satisfying C, the Equivalence and $\leqq$-Problems for constant-free expressions on $\mathcal{L}$ are *coNP*-hard. We also show that, for each lattice $\mathcal{L}$ satisfying C and for each nontrivial congruence $\equiv$ on $\mathcal{L}$, the problem of determining for $F, G \in \mathrm{EXPR}(\mathcal{L})$ if $F \equiv G$ is *coNP*-hard. Finally, we observe that each distributive lattice satisfies the condition C.

DEFINITION 2.1. Let $\mathcal{L}=(S, \vee, \wedge)$. Let $n \geqq 1$. An sp_n*-expression* (*on* $\mathcal{L}$) is an expression $F \in \mathrm{EXPR}(\mathcal{L})$ for which there exist $2n$ distinct variables $x_1, y_1, \cdots, x_n$, and y_n such that F is of the form

$$t_1 \vee t_2 \vee \cdots \vee t_k \qquad (k \geqq 1),$$

where each t_i is of the form

$$t_{i1} \wedge t_{i2} \wedge \cdots \wedge t_{in}$$

and each t_{ij} is either x_j, or y_j or $(x_j \vee y_j)$.

DEFINITION 2.2. Let $\mathcal{L}=(S, \vee, \wedge)$ be a lattice. Let $n \geqq 1$. Let F be an sp_n-expression on $\mathcal{L}$. Let $x_1, y_1, \cdots, x_n$, and y_n be the $2n$ variables appearing in F as in Definition 2.1. Let

$$\tau[F]=\{s \in \mathrm{EXPR}(\mathcal{L}) \mid s \text{ is of the form } s_1 \wedge s_2 \wedge \cdots \wedge s_n,\ s_i \in \{x_i, y_i\},$$
$$\text{and there is a term } t_j = t_{j1} \wedge t_{j2} \wedge \cdots \wedge t_{jn} \text{ of } F \text{ such that,}$$
$$\text{for } 1 \leqq i \leqq n, \text{ either } t_{ji} \text{ is } s_i \text{ or } t_{ji} \text{ is } (x_i \vee y_i)\}.$$

The expression $E[F]$ is $\bigvee_{s \in \tau(F)} s$.

We note that

(i) the expression $E[F]$ is also an sp_n-expression,

(ii) the size of $E[F]$ can be exponential in the size of F.

LEMMA 2.3. *Let* $\mathcal{L}=(S, \vee, \wedge)$ *be a lattice. Let* $n \geqq 1$. *Let* F *be an* sp_n *– expression on* $\mathcal{L}$. *Then* $E[F] \leqq F$.

Proof. The lemma follows from the properties of the lattice operations $\vee$ and $\wedge$ by repeated applications of Proposition 1.3.

DEFINITION 2.4. Let $\mathcal{L}=(S, \vee, \wedge)$ be a lattice. Let $X=\{x_i \mid i \in N\}$ and $Y=\{y_i \mid i \in N\}$ be disjoint countably infinite sets of variables. Let $n \geqq 1$. Let G_n be the sp_n-expression

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n).$$

The expression SP_n is the expression $E[G_n]$.

For example, the expression SP_2 is

$$(x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \wedge y_2).$$

We note that, for all $n \geqq 1$, the expression SP_n is an sp_n-expression.

The significance of the SP_n expressions is that, as will be shown, any DNF Boolean expression f with n variables can be transformed in polynomial time into a constant-free expression on $\mathcal{L}$, $F(f)$, such that f is a tautology if and only if $F(f) \wedge SP_n$ is equivalent to SP_n. Consequently, if SP_n can be expressed by sufficiently succinct expressions on $\mathcal{L}$, the tautology problem for DNF Boolean expressions can be polynomially reduced to the equivalence problem on $\mathcal{L}$.

THEOREM 2.5. *Let $\mathcal{L}$ be any lattice satisfying the condition* C:

C: *there exists a deterministic polynomially time-bounded algorithm (i.e., Turing machine) A that, given input 1^n ($n \geqq 1$), outputs an expression $A_n \in \mathrm{EXPR}(\mathcal{L})$ equivalent on $\mathcal{L}$ to the expression SP_n.*

Then the Equivalence and $\leqq$-Problems for the constant-free expressions on $\mathcal{L}$ are coNP-hard.

Proof. As noted above, the Equivalence and $\leqq$-Problems for constant-free expressions on $\mathcal{L}$ are linearly reducible to each other. Thus, it suffices to show that the Equivalence Problem for constant-free expressions on $\mathcal{L}$ is *coNP*-hard. The proof consists of a deterministic polynomially time-bounded reduction of the set of tautological DNF Boolean expressions to the set of equivalent constant-free expressions on $\mathcal{L}$. See [6], [16] for related reductions.

The reduction proceeds as follows. Let f be an n-variable DNF Boolean expression having m clauses $c_1, \cdots$, and c_m. Thus f is of the form c_1 *or* c_2 *or* $\cdots$ *or* c_m. Without loss of generality, we assume that the variables appearing in f are $x_1, x_2, \cdots$, and x_n and that no variable appears more than once in any clause of f. Let $y_1, y_2, \cdots$, and y_n be n distinct variables not appearing in f. For each clause c_j of f, we construct the expression

$$t_j = t_{j1} \wedge t_{j2} \wedge \cdots \wedge t_{jn} \quad \text{on } \mathcal{L} \text{ where}$$

$$t_{ji} = \begin{cases} (x_i \vee y_i) & \text{if neither } x_i \text{ nor } \bar{x}_i \text{ appears in } c_j, \\ x_i & \text{if } x_i \text{ appears in } c_j, \\ y_i & \text{if } \bar{x}_i \text{ appears in } c_j. \end{cases}$$

Let $F(f)$ be the sp_n-expression $t_1 \vee t_2 \vee \cdots \vee t_m$. Let A_n be the expression output by the algorithm A that is equivalent on $\mathcal{L}$ to the expression SP_n. Let $G(f)$ be the expression $(F(f) \wedge A_n)$. We claim that

CLAIM 2.5.1. *The expressions $G(f)$ and A_n are equivalent on $\mathcal{L}$ if and only if f is a tautology.*

Since the set of tautological DNF Boolean expressions is *coNP*-complete, the expressions $G(f)$ and A_n are constant-free expressions on $\mathcal{L}$, and the expressions $G(f)$ and A_n can be constructed from f in deterministic polynomial time, the correctness of Claim 2.5.1 implies that the Equivalence Problem for constant-free expressions on $\mathcal{L}$ is *coNP*-hard.

Proof of Claim 2.5.1. Since $G(f) = (F(f) \wedge A_n) \leqq A_n$, we need only show that $A_n \leqq G(f)$ if and only if f is a tautology.

If f is a tautology, then $\tau[F(f)] = \{s \in \text{EXPR}(\mathscr{L}) \mid s \text{ is of the form } s_1 \wedge s_2 \wedge \cdots \wedge s_n, \text{ and } s_i \in \{x_i, y_i\}\}$. Thus $A_n = SP_n = E[F(f)] \leqq F(f)$ by Lemma 2.3. Thus $A_n \leqq (F(f) \wedge A_n) = G(f)$.

If f is not a tautology, then some expression s of the form $\tilde{s}_1 \wedge \tilde{s}_2 \wedge \cdots \wedge \tilde{s}_n$, where $\tilde{s}_i \in \{x_i, y_i\}$, does not appear in $\tau[F(f)]$. Let a and b be elements of the domain of $\mathscr{L}$ such that $a \leqq b$ but $a \neq b$. Since $\mathscr{L}$ is nondegenerate by assumption, such elements a and b always exist. Let α be any assignment of values from the domain of $\mathscr{L}$ to the variables such that under α:

$$\text{if } \tilde{s}_i = x_i, \text{ then } \alpha[x_i] = b \quad \text{and} \quad \alpha[y_i] = a, \text{ or}$$

$$\text{if } \tilde{s}_i = y_i, \text{ then } \alpha[x_i] = a \quad \text{and} \quad \alpha[y_i] = b.$$

By inspection $\alpha[SP_n] = \alpha[A_n] = b$. Also by inspection $\alpha[F(f)] = a$ since each expression t_j appearing in $F[f]$ has an occurrence of a variable y in it such that the value of the variable equals a under α. Thus $\alpha[G(f)] = a \wedge b = a$. Consequently $\sim[\alpha[A_n] \leqq \alpha[G(f)]]$, and hence, $\sim[A_n \leqq G(f)]$.

The conclusion of Theorem 2.5 also applies to each nontrivial congruence $\equiv$ on $\mathscr{L}$.

THEOREM 2.6. *Let $\mathscr{L}$ be any lattice satisfying the condition* C *of Theorem* 2.5. *Let $\equiv$ be any nontrivial congruence on $\mathscr{L}$. Then the problem of determining, for constant-free expressions F and G on $\mathscr{L}$, if $F \equiv G$ is coNP-hard.*

Proof. Let $\mathscr{L} = (S, \vee, \wedge)$. Let $S/\equiv$ be the set of equivalence classes of $\equiv$. For all $x \in S$, let $\lfloor x \rfloor$ denote the equivalence class of x. Let $\vee'$ and $\wedge'$ be the binary operations on $S/\equiv$ defined by, for all $x, y \in S$, $\lfloor x \rfloor \vee' \lfloor y \rfloor = \lfloor x \vee y \rfloor$ and $\lfloor x \rfloor \wedge' \lfloor y \rfloor = \lfloor x \wedge y \rfloor$. Since $\equiv$ is a congruence on $\mathscr{L}$, the operations $\vee'$ and $\wedge'$ are well defined and the algebraic structure $\mathscr{L}' = (S/\equiv, \vee', \wedge')$ is a nontrivial lattice [1], [5]. It can easily be verified that, for all $F, G \in \text{EXPR}(\mathscr{L})$, $F \equiv G$ on $\mathscr{L}$ if and only if $F = G$ on $\mathscr{L}'$. Finally, the equivalence problem for constant-free expressions on $\mathscr{L}'$ is *coNP*-hard since L' also satisfies condition C. This last claim follows since, for all $n \geqq 1$, $A_n = SP_n$ on $\mathscr{L}$ implies $A_n \equiv SP_n$ on $\mathscr{L}$, and hence, implies $A_n = SP_n$ on L'.

Finally, each distributive lattice satisfies condition C.

COROLLARY 2.7. *Let $\mathscr{L}$ be a distributive lattice. Then*

1. *$\mathscr{L}$ satisfies condition C,*

2. *the Equivalence Problem for constant-free expressions on $\mathscr{L}$ is coNP-hard* [6].

In addition,

3. *the Equivalence Problem for sp_n-expressions ($n \geqq 1$) on $\mathscr{L}$ is coNP-hard.*

Proof. Let PP_n^1 be the expression

$$(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n).$$

Then for all $n \geqq 1$, PP_n^1 is equivalent on $\mathscr{L}$ to SP_n, and is constructible deterministically in polynomial time in n. Thus $\mathscr{L}$ satisfies condition C, and hence by Theorem 2.5, the Equivalence Problem for the constant-free expressions on $\mathscr{L}$ is *coNP*-hard. Finally let f, $F(f)$ and n be as in the proof of Theorem 2.5. Then both $F(f)$ and PP_n^1 are sp_n-expressions and $F(f) = PP_n^1$ on $\mathscr{L}$ if and only if f is a tautology.

3. Finite lattices and the expressibility condition C. We show that each finite lattice also satisfies condition C. To accomplish this, we introduce a countably infinite collection of families of expressions $\{\{PP_n^k \mid n \geqq 1\} \mid k \geqq 1\}$ as possible candidates for the family of expressions $\{A_n \mid n \geqq 1\}$ of the statement of condition C. For each finite lattice $\mathscr{L}$, we show that there exists $k_0 \geqq 1$ such that, for all $n \geqq 1$, $PP_n^{k_0} = SP_n$ on $\mathscr{L}$.

DEFINITION 3.1. Let $\mathcal{L}=(S, \vee, \wedge)$ be a lattice. Let $X=\{x_i \mid i \in N\}$ and $Y=\{y_i \mid i \in N\}$ be disjoint countably infinite sets of variables. Let k, n be nonnegative integers. Let $[1, n]=\{1, 2, \cdots, n\}$. Let $C(n, k)$ be the family of all subsets of $[1, n]$ with k elements. For any $k \leqq n$ and $I \in C(n, k)$, the set of expressions $\sigma[I]$ and the expression F_I are defined as follows:

(i) $\sigma[I]=\{\bigwedge_{i \in I} w_i \mid w_i \in \{x_i, y_i\} \text{ for } i \in I\}$,

(ii) F_I is the expression $\bigvee_{w \in \sigma[I]} w$.

For all n, $k \geqq 1$, the expression PP_n^k is defined as follows. For $k > n$, the expression PP_n^k is the same as SP_n, otherwise it is $\bigwedge_{I \in C(n,k)} F_I$.

For example the expression PP_2^2 is

$$(x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \wedge y_2),$$

and the expression PP_3^2 is

$$\begin{aligned}((x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \wedge y_2)) \\ \wedge ((x_1 \wedge x_3) \vee (x_1 \wedge y_3) \vee (y_1 \wedge x_3) \vee (y_1 \wedge y_3)) \\ \wedge ((x_2 \wedge x_3) \vee (x_2 \wedge y_3) \vee (y_2 \wedge x_3) \vee (y_2 \wedge y_3)).\end{aligned}$$

We note that, for each fixed k, the expression PP_n^k is constructible deterministically in polynomial time in n.

LEMMA 3.2. *Let $k \geqq 1$. Then for all lattices $\mathcal{L}$ and for all $n \geqq 1$,*

1. *$PP_n^k \geqq SP_n$ on $\mathcal{L}$,*
2. *$PP_n^k \geqq PP_n^{k+1}$ on $\mathcal{L}$.*

Proof. Since PP_n^k is $\bigwedge_{I \in C(n,k)} F_I$, PP_n^n is $F_{[1,n]}$, which is the same as SP_n. Therefore 1 follows from 2, and 2 need only be proved for the case that $k < n$.

Proof of 2. Let $I \in C(n, k)$. Let $j \in [1, n]-I$. Let $I'=I \cup \{j\}$. Then

$$F_I \geqq F_I \wedge (x_j \vee y_j) \geqq (F_I \wedge x_j) \vee (F_I \wedge y_j) \geqq F_{I'}$$

by repeated application of Proposition 1.3. Thus

$$PP_n^k = \bigwedge_{I \in C(n,k)} F_I \geqq \bigwedge_{I' \in C(n,k+1)} F_{I'} = PP_n^{k+1}.$$

THEOREM 3.3. *Let $\mathcal{L}=(S, \vee, \wedge)$ be a finite lattice where the cardinality of the set S equals $m \geqq 2$. Then for all $n \geqq 1$, the constant-free expressions $PP_n^{m(m+1)/2}$ and SP_n are equivalent on $\mathcal{L}$. Thus the lattice $\mathcal{L}$ satisfies condition* C *of Theorem* 2.5, *and hence, the Equivalence and $\leqq$-Problems for constant-free formulas on $\mathcal{L}$ are coNP-hard.*

Proof. Let $\mathcal{L}=(S, \vee, \wedge)$ be a finite lattice of cardinality $m \geqq 2$. By Lemma 3.2, we need only show that, for all $n \geqq 1$, $PP_n^{m(m+2)/2} \leqq SP_n$ on $\mathcal{L}$. By Definition 3.1, we can assume that $m(m+1)/2 < n$.

Let α be an assignment of values to the variables. Let the values taken on by the variables x_i and y_i under α be a_i and b_i, respectively, for $1 \leqq i \leqq n$. The set of unordered pairs $\{\{a_1, b_1\}, \cdots, \{a_n, b_n\}\}$ has at most $\binom{m}{2}+\binom{m}{1}=m(m+1)/2=k$ distinct elements. Thus, there exists $I_\alpha \in C(n, k)$ with $I_\alpha=\{n_1, n_2, \cdots, n_k\}$ where $1 \leqq n_1 < n_2 < \cdots < n_k \leqq n$ such that

$$\{\{a_1, b_1\}, \cdots, \{a_n, b_n\}\}=\{\{a_{n_1}, b_{n_1}\}, \cdots, \{a_{n_k}, b_{n_k}\}\}. \tag{3.3.1}$$

Let $w \in \sigma[I_\alpha]$. Thus, w is an expression of the form

$$w_{n_1} \wedge w_{n_2} \wedge \cdots \wedge w_{n_k}$$

where $w_{n_i} \in \{x_{n_i}, y_{n_i}\}$. Let $W = \{s \in S \mid$ under α, s is taken on by one of the variables w_{n_1}, $w_{n_2}, \cdots,$ or $w_{n_k}\}$. For $1 \leqq j \leqq n$, let z_j be the variable

$$w_j \quad \text{if } j \in I_\alpha,$$

$$x_j \quad \text{if } j \notin I_\alpha \text{ and } a_j \in W,$$

$$y_j \quad \text{if } j \notin I_\alpha \text{ and } a_j \notin W.$$

By (3.3.1) for $1 \leqq j \leqq n$, there exists $n_i \in I_\alpha$ such that $\{a_j, b_j\} = \{a_{n_i}, b_{n_i}\}$. Hence for $1 \leqq j \leqq n$, $\{a_j, b_j\} \cap W \neq \phi$. Thus for $1 \leqq j \leqq n$, if $j \notin I_\alpha$ and $z_j = y_j$, then $b_j \in W$.

Let z be the expression $z_1 \wedge z_2 \wedge \cdots \wedge z_n$. Let $Z = \{s \in S \mid$ under α, s is taken on by one of the variables $z_1, \cdots,$ or $z_n\}$. Then $W = Z$, and hence, the expressions w and z take on the same value under α. Thus for all $w \in \sigma[I_\alpha]$, there exists a meet z of the form $z_1 \wedge z_2 \wedge \cdots \wedge z_n$ where each $z_i \in \{x_i, y_i\}$ such that

$$\alpha[w] \leqq \alpha[z] \leqq \alpha[SP_n].$$

Thus $\alpha[F_{I_\alpha}] \leqq \alpha[SP_n]$ for all α, and hence, $\alpha[PP_n^k] \leqq \alpha[SP_n]$ for all α.

Next, we show that there is no finite set $\{i_1, \cdots, i_m\}$ of nonnegative integers such that, for each finite lattice $\mathscr{L}$ and for all $n \geqq 1$,

$$PP_n^{i_1} = SP_n \text{ on } \mathscr{L}, \text{ or } PP_n^{i_2} = SP_n \text{ on } \mathscr{L}, \text{ or } \cdots; \text{ or } PP_n^{i_m} = SP_n \text{ on } \mathscr{L}.$$

THEOREM 3.4. *For all $k \geqq 1$, there exists a finite lattice $\mathscr{L}_k$ of length $k+1$ such that, for all $n \geqq k+1$, $PP_n^k \neq SP_n$ on $\mathscr{L}_k$.*

Proof. For all $k \geqq 1$, the lattices $\mathscr{L}_k$ are presented diagramatically in Fig. 1.

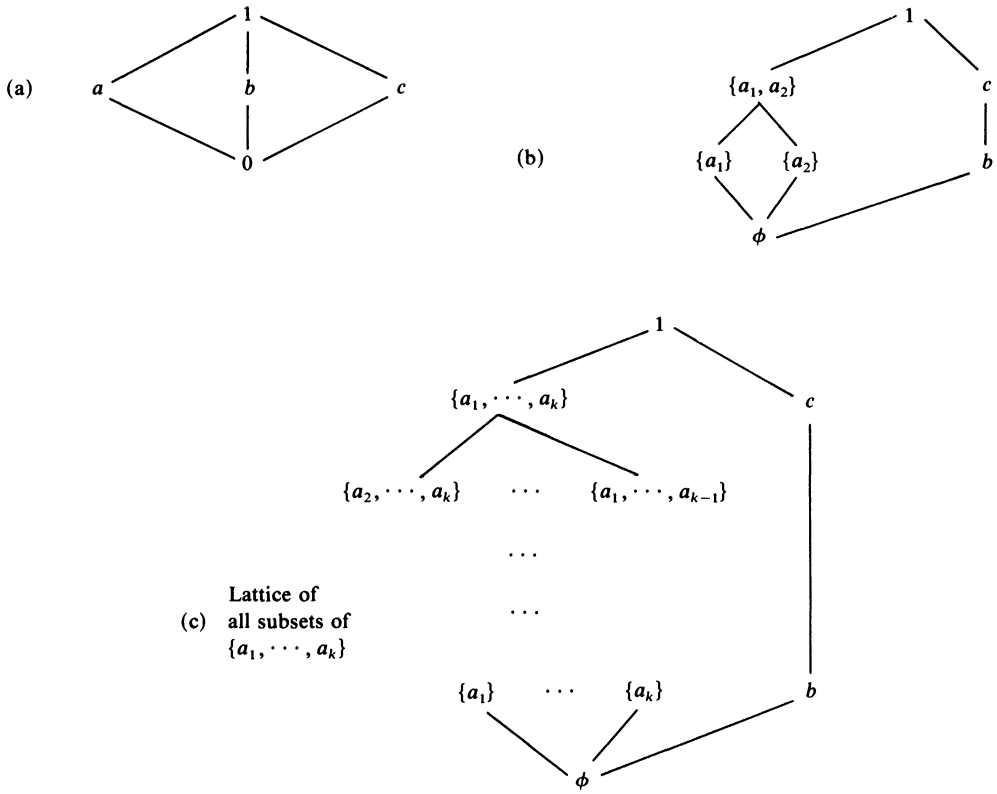


FIG. 1. (a) *The lattice L_1 of length* 2. (b) *The lattice L_2 of length* 3. (c) *The lattice L_k of length $k+1$ ($k \geqq 2$).*

Let $k=1$. Thus $n \geqq 2$. Let α be any assignment of values to the variables such that

$$\alpha[x_1]=a, \qquad \alpha[x_2]=b,$$
$$\alpha[x_i]=a \quad \text{for } 3 \leqq i \leqq n,$$
$$\alpha[y_i]=c \quad \text{for } 1 \leqq i \leqq n.$$

Then by direct inspection, $\alpha[PP_n^1]=1$ and $\alpha[SP_n]=c$.

Let $k=2$. Then $n \geqq 3$. Let α be any assignment of values to the variables such that

$$\alpha[x_1]=\{a_1\}, \qquad \alpha[x_2]=\{a_2\},$$
$$\alpha[x_i]=\{a_1, a_2\} \quad \text{for } 3 \leqq i \leqq n,$$
$$\alpha[y_i]=c \quad \text{for } 1 \leqq i \leqq n-1,$$
$$\alpha[y_n]=b.$$

Then $\alpha[PP_n^2]=c$ and $\alpha[SP_n]=b$. This follows since

(i) any meet of variables containing both an x_i and a y_j takes the value ϕ under α;

(ii) for all $I \in C(n, 2)$ where $I=\{i_1, i_2\}$ and $1 \leqq i_1 < i_2 \leqq n$,

$$\alpha[F_I]=[\alpha[x_{i_1}] \wedge \alpha[x_{i_2}]] \vee [\alpha[y_{i_1}] \wedge \alpha[y_{i_2}]]$$
$$=\begin{cases} c & \text{if } I=\{1,2\}, \\ 1 & \text{otherwise;} \end{cases}$$

(iii) $\alpha[SP_n]=[\alpha[x_1] \wedge \cdots \wedge \alpha[x_n]] \vee [\alpha[y_1] \wedge \cdots \wedge \alpha[y_n]]=\phi \vee b=b$.

Let $k>2$. The proof for the case when $k=2$ is generalized as follows. Let α be any assignment of values to the variables such that

$$\alpha[x_i]=\{a_1, \cdots, a_k\}-\{a_i\} \quad \text{for } 1 \leqq i \leqq k,$$
$$\alpha[x_i]=\{a_1, \cdots, a_k\} \quad \text{for } k+1 \leqq i \leqq n,$$
$$\alpha[y_i]=c \quad \text{for } 1 \leqq i \leqq n-1,$$
$$\alpha[y_n]=b.$$

Then $\alpha[PP_n^k]=c$ and $\alpha[SP_n]=b$. This follows since

(iv) any meet of variables containing both an x_i and a y_j takes the value ϕ under α;

(v) for all $I \in C(n, k)$ where $I=\{i_1, \cdots, i_k\}$ and $1 \leqq i_1 < \cdots < i_k \leqq n$,

$$\alpha[F_I]=[\alpha[x_{i_1}] \wedge \cdots \wedge \alpha[x_{i_k}]] \vee [\alpha[y_{i_1}] \wedge \cdots \wedge \alpha[y_{i_k}]]$$
$$=\begin{cases} c & \text{if } I=\{1, 2, \cdots, k\}, \\ 1 & \text{otherwise;} \end{cases}$$

(vi) $\alpha[SP_n]=[\alpha[x_1] \wedge \cdots \wedge \alpha[x_n]] \vee [\alpha[y_1] \wedge \cdots \wedge \alpha[y_n]]=\phi \vee b=b$.

4. Expression equivalence problems solvable in polynomial time. Deterministic polynomial time-bounded algorithms are described for several Expression Equivalence Problems. The results obtained include the following:

4a. the Equivalence Problems for the DNF and for the CNF expressions on any lattice are solvable deterministically in polynomial time;

4b. the Equivalence Problem for constant-free expressions on the free lattice with 3 generators is solvable deterministically in polynomial time.

Our proof of 4a also shows that

4c. two DNF or two CNF expressions are equivalent on some lattice if and only if they are equivalent on every lattice, and hence, if and only if they are equivalent on the free lattice with 3 generators.

Thus, result 4b implies result 4a, once result 4c is known. We prove 4a separately, however, because our proof of 4a directly implies 4c and, in addition, provides a simple characterization of DNF and of CNF expressions with minimal numbers of occurrences of operators (see Theorem 5.1).

To prove 4a, we need some additional notation and properties of DNF and of CNF expressions.

DEFINITION 4.1. Let $\mathscr{L}$ be a lattice. Let $f \in \text{EXPR}(\mathscr{L})$ such that f is a meet or a join of variables. Then $\text{Var}(f)$ is the set of all variables appearing in f. Let g be a DNF or a CNF expression on $\mathscr{L}$. Thus g is a join, or a meet, respectively, of expressions $g_1, \cdots$, and g_k $(k \geqq 1)$. The subexpressions $g_1, \cdots$, and g_k are called the *terms of* g. The set Min-Var (g) is the set of all $\text{Var}(g_i)$, $1 \leqq i \leqq k$, for which there exists *no* j not equal to i for which $\text{Var}(g_j) \subsetneq \text{Var}(g_i)$. If g is a DNF expression, then let g' be of the form

$$g_{i_1} \vee \cdots \vee g_{i_l} \qquad (1 \leqq l \leqq k)$$

and if g is a CNF expression, then let g' be of the form

$$g_{i_1} \wedge \cdots \wedge g_{i_l} \qquad (1 \leqq l \leqq k)$$

where

1. $1 \leqq i_1 < \cdots < i_l \leqq k$,
2. $\text{Var}(g_{i_j}) \in \text{Min-Var}(g)$ for $1 \leqq j \leqq l$,
3. for each g_i, $1 \leqq i \leqq k$, if $\text{Var}(g_i)$ is in Min-Var (g), then there is exactly one j, $1 \leqq j \leqq k$, such that $\text{Var}(g_{i_j}) = \text{Var}(g_i)$.

Thus if g is a DNF or a CNF expression on a lattice $\mathscr{L}$, then g' is also a DNF or a CNF expression, respectively, on $\mathscr{L}$. We note that given g, g' is unique up to commutativity of $\vee$ and $\wedge$. For example, if g is the expression

$$(x \wedge y) \vee (x \wedge z \wedge w) \vee z \vee (y \wedge z),$$

then Min-Var (g) is $\{\{x, y\}, \{z\}\}$ and g' is the expression

$$(x \wedge y) \vee z.$$

The next proposition presents elementary but important properties of Min-Var (g) and of g'.

PROPOSITION 4.2. *Let $\mathscr{L}$ be a lattice. Let f and g both be* DNF *or both be* CNF *expressions on $\mathscr{L}$. Then,*

1. *$f = f'$ and $g = g'$ on $\mathscr{L}$,*
2. *$f = g$ on $\mathscr{L}$ if and only if* Min-Var $(f) =$ Min-Var(g).

Proof. 1. Let f be a DNF expression on $\mathscr{L}$. Clearly, $f' \leqq f$ on $\mathscr{L}$. Let f_i be a term of f. If $\text{Var}(f_i) \in \text{Min-Var}(f)$, then $f_i \leqq f'$ on $\mathscr{L}$. If $\text{Var}(f_i) \notin \text{Min-Var}(f)$, then there is a term f_j of f such that $\text{Var}(f_j) \subsetneq \text{Var}(f_i)$ and $\text{Var}(f_j) \in \text{Min-Var}(f)$. In which case, $f_i \leqq f_j \leqq f'$ on $\mathscr{L}$. Hence $f_i \leqq f'$ on $\mathscr{L}$, and hence, $f \leqq f'$ on $\mathscr{L}$. The case when f is a CNF expression on $\mathscr{L}$ follows by a dual argument.

2. Let f and g be DNF expressions on $\mathcal{L}$. By 1, $f = f'$ and $g = g'$ on $\mathcal{L}$. Suppose $f = g$ on $\mathcal{L}$ but Min-Var $(f) \neq$ Min-Var (g). Let a and b be elements of $\mathcal{L}$ such that $a \leqq b$ but $a \neq b$. Since $\mathcal{L}$ is nondegenerate, such elements a and b always exist. Let w be an element of the symmetric difference of the sets Min-Var (f) and Min-Var (g) of *minimal* cardinality. Then $w = \{v_{i_1}, \cdots, v_{i_k}\}$ where the v_{i_j} are distinct variables. Let w be the expression $v_{i_1} \wedge \cdots \wedge v_{i_k}$. Let α be the assignment of values from the domain of $\mathcal{L}$ to the variables such that $\alpha[v_{i_1}] = \cdots \alpha[v_{i_k}] = b$ and such that $\alpha[x] = a$ for all other variables x. If $w \in$ Min-Var (f), then $\alpha[f] = b$ and $\alpha[g] = a$. If $w \in$ Min-Var (g), then $\alpha[f] = a$ and $\alpha[g] = b$. Thus in either case $\alpha[f] \neq \alpha[g]$, and hence, $f \neq g$ on $\mathcal{L}$, a contradiction. The case when f and g are CNF expressions on $\mathcal{L}$ follows by a dual argument.

THEOREM 4.3. *Let $\mathcal{L}$ be a lattice. Then the Equivalence Problems for the* DNF *expressions on $\mathcal{L}$ and for the* CNF *expressions on $\mathcal{L}$ are solvable deterministically in polynomial time.*

Proof. By Proposition 4.2 if f and g are both DNF or are both CNF expressions on $\mathcal{L}$, then $f = g$ on $\mathcal{L}$ if and only if Min-Var $(f) =$ Min-Var (g). The sets Min-Var (f) and Min-Var (g) can be computed from f and g and can be tested for equality deterministically in polynomial time.

We note as claimed above that the proof of Theorem 4.3 also shows 4c.

One immediate corollary of Theorem 4.3 is the following.

COROLLARY 4.4. *Let $\mathcal{L}$ be a lattice. Then, the $\leqq$-Problems for* DNF *expressions and for* CNF *expressions on $\mathcal{L}$ are solvable deterministically in polynomial time.*

Proof. As noted above, $F \leqq G$ if and only if $F \vee G = G$ if and only if $F \wedge G = F$, for all expressions F and G. If F and G are DNF expressions, then so is $F \vee G$; and if F and G are CNF expressions, then so is $F \wedge G$.

Next, we consider the equivalence problem for constant-free expressions on various free lattices. A precise definition of the *free lattice on $n \geqq 2$ generators*, denoted by $FL(n)$, will not be presented here and can be found in [5], [14]. Intuitively, the free lattice on $n \geqq 2$ generators $\{g_1, \cdots, g_n\}$ is the lattice generated by $g_1, \cdots,$ and g_n in which there are *no* laws of equality except those derivable from the associative, commutative, and indempotent properties of the operations $\vee$ and $\wedge$ and from the identities

$$x \vee (x \wedge y) = x \quad \text{and} \quad x \wedge (x \vee y) = x \quad \text{for all } x \text{ and } y.$$

The lattice $FL(2)$ with generators g_1 and g_2 is presented in Fig. 2. It is a nondegenerate distributive lattice, and thus by Corollary 2.7, the equivalence problem for constant-free expressions on $FL(2)$ is *coNP*-hard. For all $n \geqq 3$, the equivalence problem for

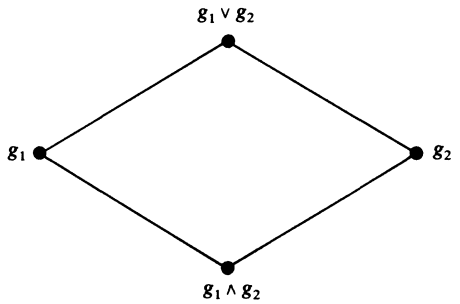


FIG. 2

constant-free expressions on $FL(n)$ is known to be decidable [5], [14]. In addition for $n \geqq 3$ Birkhoff in [5] has observed that the equivalence problem for constant-free expressions on $FL(n)$ requires no more than an exponential number of steps. Freese [12] has shown that there are expressions for which an unmodified use of Whitman's approach [25], [26] to equivalence testing requires exponential time. In the rest of this section we show that a polynomial number of steps suffice. To do this, we need some additional notation and several results from [25], [26]. We note that Freese [12] and Cosmadakis et al. [10] have also extended Whitman's work so as to obtain polynomial time algorithms for this problem.

Henceforth, for all $n \geqq 1$, we denote the generators of $FL(n)$ by $g_1, \cdots$, and g_n. Also, let F be an expression on $FL(n)$ involving only parentheses, the operators $\vee$ and $\wedge$, and the symbols $g_1, \cdots$, and g_n. We denote the element of $FL(n)$ represented in the obvious way by the expression F by $\mu(F)$.

PROPOSITION 4.5. *Let n, $m \geqq 3$. Let $k \geqq 1$. Let F and G be expressions involving only parentheses, the operators $\vee$ and $\wedge$, and the variables symbols $x_1, \cdots$, and x_k. Then the following statements are equivalent*:

1. $F \leqq G$ *on* $FL(n)$,
2. $F \leqq G$ *on* $FL(m)$,
3. *if α is any assignment of values from $FL(k)$ to the variables such that $\alpha[x_1] = g_1, \cdots$, and $\alpha[x_k] = g_k$, then $\alpha[F] \leqq \alpha[G]$ on $FL(k)$.*

Proof. The proposition follows easily from [26] and the properties of free lattices.

PROPOSITION 4.6 [5], [25], [26]. *Let $n \geqq 3$. Let F and G be two expressions involving only parentheses, the operators $\vee$ and $\wedge$, and the symbols $g_1, \cdots$, and g_n. Let $f = \mu(F)$ and let $g = \mu(G)$. Then $f \leqq g$ on $FL(n)$ if and only if*

(i) *F is g_i, G is g_j, and $i = j$, or*

one or more of the following hold:

(ii) *F is $(F_1 \vee F_2)$, and $\mu(F_1) \leqq g$ and $\mu(F_2) \leqq g$,*

(iii) *F is $(F_1 \wedge F_2)$, and $\mu(F_1) \leqq g$ or $\mu(F_2) \leqq g$,*

(iv) *G is $(G_1 \vee G_2)$, and $f \leqq \mu(G_1)$ or $f \leqq (G_2)$,*

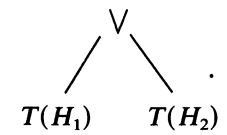
(v) *G is $(G_1 \wedge G_2)$, and $f \leqq \mu(G_1)$ and $f \leqq \mu(G_2)$.*

THEOREM 4.7. *Let $n \geqq 3$. Let $\mathscr{L}$ be the lattice $FL(n)$. Then, the problem of determining, given expressions F and G in* EXPR $(\mathscr{L})$, *if $F \leqq G$ on $\mathscr{L}$ is decidable deterministically in polynomial time.*

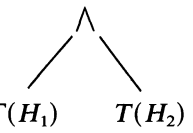
Proof. Let F and G be elements of EXPR $(\mathscr{L})$. Let the variable symbols appearing in F or in G be $x_1, \cdots$, and x_k. By Proposition 4.5 $F \leqq G$ on $\mathscr{L}$ if and only if $\alpha[F] \leqq \alpha[G]$ on $FL(k)$, where α is any assignment of values from $FL(k)$ to the variables such that $\alpha[x_1] = g_1, \cdots$, and $\alpha[x_k] = g_k$. We use Proposition 4.6, together with the ideas of *dynamic programming* [3], to determine if $\alpha[F] \leqq \alpha[G]$ on $FL(k)$, where α is an assignment of values from $FL(k)$ to the variables as described immediately above.

To each constant-free expression H on $\mathscr{L}$, and thus, to F and to G, we can associate the labeled binary tree $T(H)$ constructed recursively as follows:

1. If H is a variable x_i, then $T(H)$ is the single node g_i.
2. IF H is of the form $(H_1 \vee H_2)$, then $T(H)$ is the tree



3. If H is of the form $(H_1 \wedge H_2)$, then $T(H)$ is the tree



Clearly, the trees $T(F)$ and $T(G)$ can be constructed deterministically from F and G, respectively, in $O(|F|+|G|)$ operations. We note that the number of interior nodes of $T(F)$ and of $T(G)$ equals the number of occurrences of operators in F and in G, respectively. For each node n in such a tree, we denote the expression associated to the subtree with root n by f_n. For each node n of $T(F)$ and m of $T(G)$, we define the predicate $R(n, m)$ as follows:

$$R(n, m) = \begin{cases} \text{True} & \text{if } \mu(f_n) \leqq \mu(f_m), \\ \text{False} & \text{otherwise.} \end{cases}$$

Proposition 4.6 provides the basis for a bottom-up method of computing $R(n, m)$ as follows.

4.7a. If both n and m are leaf nodes, then $R(n, m) = \text{True}$ if and only if both nodes have the same label.

4.7b. If n is an interior node with children n_1 and n_2 and m is a leaf node, then

$$R(n, m) = \begin{cases} R(n_1, m) \quad \textit{or } R(n_2, m) & \text{if the label of } n \text{ is } \wedge, \\ R(n_1, m) \quad \textit{and } R(n_2, m) & \text{if the label } n \text{ is } \vee. \end{cases}$$

4.7c. If n is a leaf node and m is an interior node with children m_1 and m_2, then

$$R(n, m) = \begin{cases} R(n, m_1) \quad \textit{and } R(n, m_2) & \text{if the label of } m \text{ is } \wedge, \\ R(n, m_1) \quad \textit{or } R(n, m_2) & \text{if the label of } m \text{ is } \vee. \end{cases}$$

4.7d. If both n and m are interior nodes with children n_1 and n_2, and m_1 and m_2, respectively, then

$$\begin{aligned} R(n, m) = {} & [R(n_1, m) \textit{ and } R(n_2, m)] \textit{ or } [R(n, m_1) \textit{ or } R(n, m_2)], \\ & \quad \text{if the label of } n \text{ is } \vee \text{ and the label of } m \text{ is } \vee, \\ & [R(n_1, m) \textit{ and } R(n_2, m)] \textit{ or } [R(n, m_1) \textit{ and } R(n, m_2)], \\ & \quad \text{if the label of } n \text{ is } \vee \text{ and the label of } m \text{ is } \wedge, \\ & [R(n_1, m) \textit{ or } R(n_2, m)] \textit{ or } [R(n, m_1) \textit{ or } R(n, m_2)], \\ & \quad \text{if the label of } n \text{ is } \wedge \text{ and the label of } m \text{ is } \vee, \text{ and} \\ & [R(n_1, m) \textit{ or } R(n_2, m)] \textit{ or } [R(n, m_1) \textit{ and } R(n, m_2)], \\ & \quad \text{if the label of } n \text{ is } \wedge \text{ and the label of } m \text{ is } \wedge. \end{aligned}$$

A polynomially time-bounded algorithm $\leqq$-*Test*, to determine given F, $G \in$ EXPR $(\mathscr{L})$, if $F \leqq G$ on $\mathscr{L}$, is presented in Fig. 3. To verify the correctness of this algorithm, it suffices to observe that,

> for each pair (v, w) of vertices of $T(F)$ and of $T(G)$, if R is computed in the order specified by step 3 of the algorithm, then all values of R on pairs of vertices needed to compute $R(v, w)$ are computed before $R(v, w)$ is computed.

Thus, $R(v, w)$ can be computed using only a single application of 4.6a–4.6d. Finally to verify that this algorithm is executable deterministically in polynomial time, it suffices to verify that

(i) $\|\textit{Pairs}(F, G)| = O(|F| \cdot |G|)$;
(ii) for all $(v, w, h(v), h(w)) \in \textit{Pairs}(F, G)$, $0 \leqq h(v),\ h(w) \leqq \max(|F|, |G|)$;
(iii) the sort at step 3 is executable deterministically using only $O(|\textit{Pairs}(F, G)|)$ operations;
(iv) the number of operations needed to execute step 4 is $O(|\textit{Pairs}(F, G)|)$.

Input: Let $n \geqq 3$. Two constant-free expressions F and G on $FL(n)$.
Output: "Yes", if $F \leqq G$ on $FL(n)$, and "No", otherwise.
Body:
1: Construct the labeled trees $T(F)$ and $T(G)$.
2: Construct the set
$Pairs(F, G) = \{(v, w, h(v), h(w)) \mid v$ is a node of $T(F)$; w is a node of $T(G)$; and $h(v)$ and $h(w)$ are the lengths of maximal length paths from v to a leaf of $T(F)$ and from w to a leaf of $T(G)$, respectively$\}$.
3: Sort the set $Pairs(F, G)$ under the partial order ρ given by
$(v, w, h(v), h(w))\rho(v', w', h(v'), h(w'))$ if and only if $[h(v) < h(v')]$ *or* $[h(v) = h(v')$ *and* $h(w) < h(w')]$.
4: Let the result of *step* 3 be $Pairs(F, G) = \{p_1, \cdots, p_m\}$ where $p_i = (v_i, w_i, h(v_i), h(w_i))$. Compute the predicate R in the order $R(v_1, w_1), R(v_2, w_2), \cdots, R(v_m, w_m)$ using 4.7a–4.7d on previously computed values of R.
5: If $R(v_m, w_m) =$ True, then *return* ("Yes"). Otherwise, *return* ("No").

FIG. 3. *The algorithm* $\leqq$*-Test.*

COROLLARY 4.8. *Let* $n \geqq 3$. *Let* $\mathcal{L}$ *be the lattice* $FL(n)$. *Then, the Equivalence Problem for constant-free expressions on* $\mathcal{L}$ *is solvable deterministically in polynomial time.*

Proof. The corollary follows immediately from Theorem 4.7 since $F = G$ on $\mathcal{L}$ if and only if $F \leqq G$ and $G \leqq F$ on $\mathcal{L}$.

5. The complexity of operator minimization problems. We study the computational complexity of Operator Minimization Problems and of Approximate Operator Minimization Problems for various classes of expressions and for a variety of lattices. These lattices include each of the lattices studied in §§ 2–4. Results obtained include the following:

5a. The Operator Minimization Problems are solvable deterministically in polynomial time for the DNF and for the CNF expressions on any lattice.

5b. The Operator Minimization Problem is solvable deterministically in polynomial time for constant-free expressions on the free lattice with $\geqq 3$ generators.

5c. Let $\mathcal{L}$ be a lattice. Then, the $\leqq$-Problem for expressions on $\mathcal{L}$ is linearly reducible to the Operator Minimization Problem for expressions on $\mathcal{L}$.

THEOREM 5.1. *Let* $\mathcal{L}$ *be a lattice. Then, there exists a deterministic polynomially time-bounded algorithm to calculate, given a* DNF *or a* CNF *expression on* $\mathcal{L}$ *as input, an equivalent* DNF *or* CNF *expression, respectively, on* $\mathcal{L}$ *with a minimal number of occurrences of operators.*

Proof. Let f be a DNF or a CNF expression on $\mathcal{L}$. Let $n[f] = \sum_{\text{Var}(f_i) \in \text{Min-Var}(f)} |\text{Var}(f_i)|$. By Proposition 4.2 any DNF or CNF expression, respectively, equivalent to f on $\mathcal{L}$ must have at least $n[f]$ occurrences of variables, and thus, must have at least $n[f]-1$ occurrences of operators. The expression f' is equivalent to f on $\mathcal{L}$, and has exactly $n[f]-1$ occurrences of operators. Clearly, f' can be computed from f deterministically in polynomial time.

To prove 5b, we need two preliminary results together with some additional notation. The first result is Corollary 1.1 in [26] restated in terms of EXPR $(FL(3))$ by means of Proposition 4.5.

PROPOSITION 5.2. *Let* $\mathcal{L}$ *be the lattice* $FL(3)$. *Let* $F \in \text{EXPR}(\mathcal{L})$. *Then,* F *has a minimal number of occurrences of operators of any equivalent constant-free expression on* $\mathcal{L}$ *if and only if either*

F *is of the form* $A_1 \vee \cdots \vee A_k$ $(k \geqq 2)$ *where each expression* A_i *is of the form* $A_{i1} \wedge \cdots \wedge A_{im}$ $(m \geqq 2)$ *or is a variable and*

(a) *for no expression* A_{ij} *is* $A_{ij} \leqq F$ *on* $\mathcal{L}$,

(b) *for* $1 \leqq i, i' \leqq k$, $A_{i'} \leqq A_i$ *on* $\mathscr{L}$ *implies* $i' = i$, *and*
(c) *each expression* A_i *has a minimal number of occurrences of operators of any equivalent expression on* $\mathscr{L}$, *or*

F is of the form $A_1 \wedge \cdots \wedge A_k$ $(k \geqq 2)$ *where each expression* A_i *is of the form* $A_{i1} \vee \cdots \vee A_{im}$ $(m \geqq 2)$ *or is a variable and*
(d) *for no expression* A_{ij} *is* $F \leqq A_{ij}$ *on* $\mathscr{L}$,
(e) *for* $1 \leqq i, i' \leqq k$, $A_{i'} \leqq A_i$ *on* $\mathscr{L}$ *implies* $i' = i$, *and*
(f) *each expression* A_i *has a minimal number of occurrences of operators of any equivalent expression on* $\mathscr{L}$, *or*
F is a variable symbol.

DEFINITION 5.3. Let $\mathscr{L}$ be the lattice $FL(3)$. Let $F \in \mathrm{EXPR}(\mathscr{L})$ be of the form $A_1 \vee \cdots \vee A_k$ $(k \geqq 2)$ where each expression A_i is of the form $A_{i1} \wedge \cdots \wedge A_{im}$ $(m \geqq 2)$ or is a variable, or be of the form $A_1 \wedge \cdots \wedge A_k$ $(k \geqq 2)$ where each expression A_i is of the form $A_{i1} \vee \cdots \vee A_{im}$ $(m \geqq 2)$ or is a variable. Let $B \in \mathrm{EXPR}(\mathscr{L})$. Then $F[A_i \leftarrow B]$ is the expression that results from F by replacing the expression A_i by B.

PROPOSITION 5.4. *Let* $n \geqq 3$. *Let* $\mathscr{L}$ *be the lattice* $FL(n)$. *The algorithm* Min-Test *of Fig.* 4, *given input* $F \in \mathrm{EXPR}(\mathscr{L})$, *either*

returns "F is minimal." if F has a minimal number of occurrences of operators of any equivalent constant-free expresssion on $\mathscr{L}$, *or*

returns an expression $G \in \mathrm{EXPR}(\mathscr{L})$ *equivalent to F on* $\mathscr{L}$ *such that G has at least one fewer occurrence of operators than F.*

Moreover, the algorithm Min-Test *is executable deterministically in polynomial time.*

Proof. The proof that Min-Test returns as claimed follows from Proposition 5.2 by a straightforward induction argument on the maximal depth of nesting of *alternations* of meets and joins in the expression F. Let the number of subexpressions of F be m. To prove that Min-Test is executable deterministically in polynomial time, it suffices to observe that

(i) the number of recursive calls of Min-Test before termination is $\leqq m$;
(ii) the number of operations required to execute steps 0, 1, 2, 3 and 4 or to execute steps 0, 1, 6, 7 and 8 (excluding recursive calls of Min-Test) during any call of Min-Test is bounded by a polynomial in $|F|$.

Observation (i) follows by noting that each recursive call of Min-Test occurs on a subexpression of F. Observation (ii) follows from Theorem 4.7 and Corollary 4.8 since each of the tests of the form $f \leqq g$ on $\mathscr{L}$ involves F or subexpressions of F.

THEOREM 5.5. *Let* $n \geqq 3$. *Let* $\mathscr{L}$ *be the lattice* $FL(n)$. *Then, the Operator Minimization Problem for constant-free expressions on* $\mathscr{L}$ *is solvable deterministically in polynomial time.*

Proof. By Proposition 5.4 the algorithm Op-Min of Fig. 5 solves the Operator Minimization Problem for constant-free expressions on $\mathscr{L}$ and is executable deterministically in polynomial time.

Next, we show that the $\leqq$-Problem for expressions on $\mathscr{L}$ is linearly reducible to the Operator Minimization Problem for expressions on $\mathscr{L}$.

THEOREM 5.6. *Let* $\mathscr{L}$ *be a lattice. Then for any class of expressions on* $\mathscr{L}$ *that includes the constant-free expressions and is closed under* $\vee$ *and* $\wedge$, *the problem of determining, for expressions F and G, whether* $F \leqq G$, *is linearly reducible to the problem of minimizing the number of operators in an expression.*

Proof. Let F and G be expressions in the class. Let z be a new variable occurring in neither F nor G. Let H be the expression $(F \wedge z) \vee G$. Then $F \leqq G$ if and only if

Input: A constant-free expression F on $FL(n)$ $(n \geqq 3)$.
Output:
"F is minimal.", if F has a minimal number of occurrences of operators of any equivalent constant-free expression on $FL(n)$; and
a constant-free expression G on $FL(n)$ equivalent to F on $FL(n)$ with at least one fewer occurrence of operators, otherwise.
Body:
0: If F is a single variable, *return* ("F is minimal.").
1: If F is of the form $A_1 \vee \cdots \vee A_k$ $(k \geqq 2)$ where each A_i is of the form $A_{i1} \wedge \cdots \wedge A_{im}$ $(m \geqq 2)$ or is a variable, goto 2 else goto 6.
2: For $1 \leqq i \leqq k$ do
if $\exists j$ such that $A_{ij} \leqq F$ on $FL(3)$ then
return $\left(A_{ij} \vee \bigvee_{l \neq i} A_l\right)$.
3: For $1 \leqq i, i' \leqq k$ with $i \neq i'$ do
if $A_{i'} \leqq A_i$ on $FL(3)$ then *return* $\left(\bigvee_{l \neq i'} A_l\right)$.
4: For $1 \leqq i \leqq k$ do
begin
Call Min-Test (A_i).
If Min-Test (A_i) returns an expression B,
return $(F[A_i \leftarrow B])$.
end.
5: *return* ("F is minimal.").
6: *Comment.* In this case, F is of the form $A_1 \wedge \cdots \wedge A_k$ $(k \geqq 2)$ where each A_i is of the form $A_{i1} \vee \cdots \vee A_{im}$ $(m \geqq 2)$ or is a variable.
For $1 \leqq i \leqq k$ do
if $\exists j$ such that $F \leqq A_{ij}$ on $FL(3)$ then
return $\left(A_{ij} \wedge \bigwedge_{l \neq i} A_l\right)$.
7: For $1 \leqq i, i' \leqq k$ with $i \neq i'$ do
if $A_i \leqq A_{i'}$ on $FL(3)$ then *return* $\left(\bigwedge_{l \neq i'} A_l\right)$.
8: For $1 \leqq i \leqq k$ do
begin
Call Min-Test (A_i).
If Min-Test (A_i) returns an expression B,
return $(F[A_i \leftarrow B])$.
end.
9: *return* ("F is minimal.").

FIG. 4. *The algorithm* Min-Test.

any expression equivalent to H and with a minimum number of operators does not contain an occurrence of z.

To see this, first assume that $F \leqq G$. Then H is equivalent to G, and a minimum version of H does not contain z. Now suppose that $F \leqq G$ does not hold. Then there is an assignment α to the variables occurring in F and G such that $\alpha[F] \leqq \alpha[G]$ does not hold. Let β be the extension of α whereby $\beta[z] = \alpha[F]$, and let γ be the extension

Input: A constant-free expression F on $FL(n)$ $(n \geqq 3)$.
Output: An equivalent expression $G \in \text{EXPR}(FL(n))$ with a minimal number of occurrences of operators of any equivalent expression on $FL(n)$.
Body:
0: While F is not the same expression as Min-Test (F) do $F \leftarrow$ Min-Test (F).
1: *Return* (F).

FIG. 5. *The algorithm* Op-Min.

of α whereby $\gamma[z]=\alpha[G]$. Then $\gamma[H]=\alpha[G]$. Also, $\beta[H]=\alpha[F]\vee\alpha[G]$, which is not equal to $\alpha[G]$. Thus the value of H depends on variable z, and every expression equivalent to H contains an occurrence of z.

The idea of the proof of Theorem 5.6 can be easily generalized to obtain hardness results for a variety of Approximate Operator Minimization Problems. In particular for lattices $\mathscr{L}$ satisfying condition C of Theorem 2.5, the following holds.

THEOREM 5.7. *Let $\mathscr{L}$ be any lattice satisfying condition* C *of Theorem* 2.5. *Let p be any polynomial function from N to N. Then*

1. *there exists a deterministic polynomially time-bounded algorithm to solve the p-Bounded Approximate Operator Minimization Problem for constant-free expressions on $\mathscr{L}$ only if $P=NP$;*

2. *there exists a nondeterministic polynomially time-bounded algorithm to solve the p-Bounded Approximate Operator Minimization Problem for constant-free expressions on $\mathscr{L}$ only if $NP=coNP$.*

Proof. Informally, the proof consists of a deterministic polynomially time-bounded reduction of the $\leqq$-Problem for constant-free expressions on $\mathscr{L}$ to the p-Bounded Approximate Operator Minimization Problem for constant-free expressions on $\mathscr{L}$.

Let F and G be constant-free expressions on $\mathscr{L}$. Let $\mathrm{op}(G)$ be the number of occurrences of operators in G. Let $n_0=\max\{p(i)\,|\,1\leqq i\leqq \mathrm{op}(G)\}$. Let $z_1,\cdots$, and z_{n_0+2} be distinct variable symbols not appearing in F or in G. Let H be the constant-free expression

$$(F\wedge z_1\wedge\cdots\wedge z_{n_0+2})\vee G.$$

As in the proof of Theorem 5.6, the following hold:

1. If $F\leqq G$, then $H=G$. Let H' be any constant-free expression equivalent to H on $\mathscr{L}$ having a minimum number $\mathrm{op}(H)$ of occurrences of operators. Then $p(\mathrm{op}(H))=p(\mathrm{minop}(G))\leqq n_0$.

2. If $\sim[F\leqq G]$, then any expression equivalent to H on $\mathscr{L}$ has at least one occurrence of each of the variables $z_1,\cdots$, and z_{n_0+2}. Thus, each such expression has at least n_0+1 occurrences of operators.

Thus, $F\leqq G$ if and only if the algorithm that solves the p-Bounded Approximate Operator Minimization Problem for constant-free expressions on $\mathscr{L}$, given input H, outputs an expression having fewer than n_0+1 occurrences of operators. Hence, assertations 1 and 2 of the theorem follow from Theorem 2.5.

Finally for lattices for which the $\leqq$-Problem or, equivalently, the Equivalence Problem is undecidable, the following analogue of Theorem 5.7 holds.

THEOREM 5.8. *Let $\mathscr{L}$ be any lattice for which the Equivalence Problem is undecidable. Let f be any recursive function from N to N. Then, there is no algorithm to solve the f-Bounded Approximate Operator Minimization Problem for expressions on $\mathscr{L}$.*

Proof. The proof is the same as that of Theorem 5.7 except that $n_0=\max\{f(i)\,|\,1\leqq i\leqq \mathrm{op}(G)\}$.

Conclusion. We conclude with several additional observations on the results presented above. First, the Equivalence Problem for constant-free expressions on any finite or distributive lattice $\mathscr{L}$ is *coNP*-complete. Thus, many of the lower bounds presented above are fairly "tight". For finite lattices $\mathscr{L}$, this is obvious. For distributive lattices $\mathscr{L}$ this follows since, for all $F, G\in \mathrm{EXPR}(\mathscr{L})$

$F=G$ on $\mathscr{L}$ if and only if $F=G$ on the two element lattice ($\{0,1\}$, *or*, *and*).

Second by Proposition 5.2, for all $n\geqq 1$ and all $m\geqq 3$, the expression SP_n has the minimal number of occurrences of operators of any equivalent expression on $FL(m)$.

Thus, for all $n \geqq 1$ and all $m \geqq 3$, any constant-free expression equivalent to SP_n on $FL(m)$ has at least $2^n \cdot (n-1)+2^n-1 = n \cdot 2^n - 1$ occurrences of operators. Hence unsurprisingly in the light of Corollary 4.8, for all $m \geqq 3$, the lattice $FL(m)$ does not satisfy the condition C of § 2.

Third, there is *no* fixed family of expressions $\{A_n \mid n \geqq 1\}$ such that, for all finite lattices $\mathscr{L}$ and for all $n \geqq 1$,

1. $A_n = SP_n$ on $\mathscr{L}$,
2. A_n is computable deterministically in time-bounded by a polynomial in n.

This follows from observation 2 immediately above, together with the result from [22] that

> two constant-free expressions are equivalent on all lattices if and only if they are equivalent on all finite lattices.

Fourth and last, there exist infinite nondistributive lattices that satisfy the condition C of § 2. For example, any direct product of a countably infinite number of copies of a finite lattice also satisfies the condition C of § 2.

Acknowledgments. We wish to acknowledge helpful comments from Ralph Freese and from the two referees.

REFERENCES

[1] J. C. ABBOTT, *Sets, Lattices, and Boolean Algebras*, Allyn and Bacon, Boston, MA, 1969.
[2] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[3] R. E. BELLMAN, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
[4] G. BIRKHOFF, *Lattices in applied mathematics*, in Lattice Theory: Proceedings of Symposia in Pure Mathematics, Vol. 2, Amer. Math. Soc., Providence, RI, 1961.
[5] G. BIRKHOFF, *Lattice Theory (3rd edition)*, Amer. Math. Soc., Providence, RI, 1967.
[6] 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.
[7] J. A. BRZOZOWSKI AND M. YOELI, *Digital Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1976.
[8] J. A. CANNON, *The basis of a computer system for modern algebra*, Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, 1981, pp. 1-5.
[9] S. A. COOK, *The complexity of theorem-proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.
[10] S. S. COSMADAKIS, P. C. KANELLAKIS AND N. SPYRATOS, *Partition semantics for relations*, Proceedings of the Fourth ACM Symposium on Principles of Database Systems, 1985.
[11] R. FREESE, *Free modular lattices*, Trans. AMS, 261 (1980), pp. 81-91.
[12] ———, private communication.
[13] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.
[14] G. GRATZER, *General Lattice Theory*, Academic Press, New York, 1978.
[15] G. HUET AND D. 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.
[16] H. B. HUNT III, *Finite languages and the computational complexity of algebraic structures*, in Combinatorics on Words: Progress and Perspectives, L. J. Cummings, ed., Academic Press Canada, Toronto, Canada, 1983, pp. 333-352.
[17] H. B. HUNT III AND R. E. STEARNS, *Nonlinear algebra is "hard" 1*, in preparation.
[18] G. HUTCHINSON, *Recursively unsolvable word problems of modular lattices and diagram-chasing*, J. Algebra, 26 (1973), pp. 385-399.
[19] R. D. JENKS AND B. M. TRAGER, *A language for computational algebra*, Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, 1981, pp 6-13.
[20] L. LIPSCHITZ, *The undecidability of the word problem for projective geometries and modular lattices*, Trans. AMS, 193 (1974), pp. 171-180.

[21] H. RASIOWA, *An Algebraic Approach to Non-Classical Logics*, North-Holland, Amsterdam, the Netherlands, 1974.
[22] D. SACHS, *Identities in finite partition lattices*, Proc. Amer. Math. Soc., 12 (1961), pp. 944-945.
[23] D. SCOTT, *Data types as lattices*, this Journal, 5 (1976), pp. 522-587.
[24] D. J. A. WELSH, *Matroid Theory*, Academic Press, New York, 1976.
[25] P. M. WHITMAN, *Free lattices*, Annals of Math., 42 (1941), pp. 325-330.
[26] ———, *Free lattices* II, Annals of Math., 43 (1942), pp. 104-115.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
012

AN EFFICIENT APPROXIMATION SCHEME FOR VARIABLE-SIZED BIN PACKING*

FRANK D. MURGOLO†

Abstract. In the classical bin packing problem one is required to pack a given list of items into the smallest possible number of unit-sized bins. Because this problem is NP-complete, researchers have tried to find efficient approximation algorithms that solve this problem in a reasonable amount of time. If we let $A(I)$ be the number of bins used by algorithm A to pack a list of items I, and $OPT(I)$ be the minimum number of bins necessary to pack I, one defines the asymptotic performance ratio of A to be $A(I)/OPT(I)$, as $OPT(I)$ tends to infinity. De la Vega and Lueker presented an approximation scheme, which for any $\varepsilon > 0$, yields an approximation algorithm with performance ratio $1+\varepsilon$. This scheme has time complexity polynomial in n, the number of items, but exponential in $1/\varepsilon$. Karmarkar and Karp extended this scheme to one that has time complexity polynomial in n and $1/\varepsilon$. In the variable-sized bin packing problem, one is given a list of items to pack and a set of different bin sizes to be used in packing the items. For an algorithm A, $A(I)$ is the sum of the sizes of the bins used to pack the list I. This sum is the quantity one wishes to minimize. Friesen and Langston presented an approximation algorithm with performance ratio $\frac{4}{3}$ with time complexity polynomial in the number of items and the number of different bin sizes. In this paper we present an approximation scheme which for any given positive ε produces a scheme with performance ratio $1+\varepsilon$. This algorithm has time complexity polynomial in the number of items, the number of bins, and $1/\varepsilon$.

Key words. approximation schemes, bin packing, ellipsoid method

AMS(MOS) subject classifications. 90C05, 90B99, 68Q25

1. Introduction. In the standard bin packing problem, one is given a list of items $a_1, a_2, a_3, \cdots, a_n$ and asked to pack them into the minimum number of bins so that the total size of the items in any single bin does not exceed 1. Because this problem is NP-complete, researchers have spent considerable effort in devising time efficient approximation algorithms. For an excellent survey of these efforts see [CGJ84]. Following the definitions given in [GJ79], we define some of the terms we will be using in this paper. For a given approximation algorithm, A, and an instance I of the bin packing problem we define r_I as the ratio $A(I)/OPT(I)$, where $A(I)$ is the number of bins used by A to pack I and $OPT(I)$ is the minimum number of bins required to pack I. The (asymptotic) performance ratio is the smallest positive number r such that for some positive integer N, $r_I \leqq r$ for all instances I satisfying $OPT(I) \geqq N$. Until recently, the best value for r was $1.183\cdots$ [GJ].

A polynomial approximation scheme for the bin packing problem takes as input an instance I of the problem and an error bound ε and yields, for each fixed ε, a polynomial time algorithm with $r \leqq 1+\varepsilon$. The solution may require time exponential in $1/\varepsilon$. A fully polynomial approximation scheme is one which is also polynomial in $1/\varepsilon$. Reference [DL81] provides a polynomial approximation scheme, which was extended in [KK82] to a fully polynomial approximation scheme.

In the variable-size bin packing problem, an instance I consists of a list L of items to be packed along with the set of bin sizes available for the packing. The objective is to minimize the sum of the sizes of the bins used. For an algorithm A that provides a packing for I, we let $A(I)$ denote the sum of the bin sizes used by A to pack L. We use $OPT(I)$ to denote the minimum sum of bin sizes required to pack I.

* Received by the editors June 14, 1985; accepted for publication (in revised form) May 14, 1986.

† Department of Information and Computer Science, University of California, Irvine, Irvine, California 92717. Present address, Department of Mathematics and Computer Science, University of California, Riverside, California 92521. This work was supported by National Science Foundation grant MCS 79-04997.

An early industrial application of this problem [Ei57] is the cutting stock problem in which one has a set of different sized pieces of stock from which an order for various lengths must be cut. Reference [GG61] examines a linear programming solution to this problem which derives approximate solutions efficiently in practice. The variable-size bin packing problem is NP-complete, leading one to search for efficient approximation algorithms. Reference [FL86] describes three approximation algorithms with performance ratios of 2, $\frac{3}{2}$ and $\frac{4}{3}$. In this paper we present an approximation scheme that, for any given $\varepsilon > 0$, yields an algorithm A such that $A(I) \leqq (1+\varepsilon)OPT(I) + O(1/\varepsilon^4)$. The time complexity of this scheme is linear in the number of items to be packed but exponential in $1/\varepsilon$ and the number of different bin sizes. We employ methods similar to those in [DL81] and [KK82] for handling small items, and new techniques for dealing with variable size bins. As in [KK82], in order to obtain an algorithm with time complexity polynomial in $1/\varepsilon$ and the number of different bin sizes, we describe how we can make use of the *ellipsoid method* [Kh79], [GLS81] to solve the linear programming formulation of our problem.

Our approximation scheme is primarily of theoretical importance except for instances of very large $OPT(I)$. This is in contrast to the algorithm found in [FL86] which works in time $O(n \log n + n \log k)$ with performance ratio $\frac{4}{3}$. (Here n is the number of items and k is the number of different bin sizes.)

This paper is organized as follows: § 2 describes the algorithm, § 3 describes the amount of error caused by the rounding of item sizes, § 4 describes the linear programming formulation as well as the error introduced when padding out the linear solution to integers, and § 5 bounds the amount of error introduced by disregarding small items and small bins. Section 6 describes the time complexity of the polynomial approximation scheme and § 7 describes the extensions necessary to achieve a running time polynomial in $1/\varepsilon$, and the number of different bin sizes.

2. The approximation scheme. The approximation scheme is comprised of the following steps:

1) Defining a minimum item size and bin size which are both functions of ε;
2) Temporarily disregarding small items and permanently discarding small bins;
3) Rounding the remaining item sizes in order to decrease the number of different item sizes to be packed, and formulating the packing problem as a linear programming problem;
4) Taking the packing provided by the linear program solution and creating a proper packing of all the items, i.e., padding out the solution to integers and packing the small items.

We assume we are given an instance I of the variable-size bin packing problem consisting of

1) *BINS*: a set of available bin sizes (as in [FL86] we assume, without loss of generality, the largest bin size is 1);
2) L: a list of items (sizes) $a_1, a_2, \cdots, a_n$;
3) $\varepsilon > 0$ where we desire a performance ratio of $1+\varepsilon$.

We define the performance measure of a variable-size bin packing algorithm A for an instance I as

$$A(L, BINS, \varepsilon) = \sum_{i=1}^{B} s(B_i)$$

where B is the number of bins used by A, B_i is the ith bin used, and $s(B_i)$ is the size of B_i.

As a notational convenience, we will write $A(L, BINS, \varepsilon)$ as $A(L)$. (We note that this is a generalization of the performance measure for the standard bin packing problem where all bins have a size of 1 and one counts the number of bins used as the performance measure.)

We begin by defining: (i) $\min = \varepsilon$; (ii) $\delta = \varepsilon \times \min = \varepsilon^2$. We call items of size less than δ "small items" and bins of size less than min "small bins." These small items and small bins are temporarily excluded from consideration. We will eventually pack the small items, but we never use the small bins.

3. Rounding of item sizes. As we will see later, the number of different item sizes plays a critical role in determining the time complexity of the approximation scheme. For this reason, using methods similar to those in [DL81] and [KK82], we limit the number of distinct sizes by a rounding procedure and show that this rounding does not affect the performance ratio adversely.

At this stage of the approximation scheme the list to be packed contains no small items and *BINS* contains no small bins. Let L_0 be the list of items which are not small and let m be the number of items in L_0. We assume the list $L_0 = a_1, a_2, \cdots, a_m$ has the property: $s(a_1) \geqq s(a_2) \geqq \cdots \geqq s(a_m)$. Let a_0 be a piece of size 0. We create $\lceil m/k \rceil$ groups of items by placing items a_0 through a_{k-1} into group one, a_k through a_{2k-1} into group two, etc. (We determine k in subsequent analysis.) We then increase the size of every item in each group to the size of the largest item in that group. For example, the sizes of the first k items $a_0, \cdots, a_{k-1}$ are set to the size of $a_0 = 1$. The size of items a_k through a_{2k-1} are set to the size of a_k. Calling this modified list L_1 we ask, "How does a packing of L_1 differ from a packing of L_0?" Using a packing of L_1, one can easily construct a packing for L_0 using only the bins used for L_1, yielding

$$OPT(L_0) \leqq OPT(L_1).$$

Next we imagine, for the purposes of analysis, rounding every successive k items down in size to a_{kj} for $j = 1, 2, 3, \cdots, \lceil m/k \rceil$. (For both L_1 and L_2 we pad out L_0 as necessary with items of size 0.) Calling this list L_2, we have

$$OPT(L_2) \leqq OPT(L_0). \tag{1}$$

Now we observe that L_1 is identical to L_2 except L_1 has k items of size 1 and L_2 may have some additional items of size $a_{\lfloor m/k \rfloor k}$. This tells us

$$OPT(L_1) \leqq OPT(L_2) + k. \tag{2}$$

Together (1) and (2) imply

$$OPT(L_1) \leqq OPT(L_0) + k,$$

which implies

$$\frac{OPT(L_1)}{OPT(L_0)} \leqq 1 + \frac{k}{OPT(L_0)}.$$

Since every item being packed has size greater than δ we have $OPT(L_0) \geqq \delta m$. We know $OPT(L) \geqq OPT(L_0)$ because L_0 is a subset of L. These two facts, together with the above, yield

$$\frac{OPT(L_1)}{OPT(L)} \leqq 1 + \frac{k}{\delta m}.$$

We set $k = \lceil \delta^2 m \rceil$ and derive the following.

LEMMA 1. $OPT(L_1)/OPT(L) \leqq 1+\delta$.

Hence we see that by packing L_1 instead of L, the relative error introduced is at most ε^2.

4. An algorithm to pack L_1. Given the list of L_1, as described in the previous section, we have only $S = \lceil m/k \rceil$ different item sizes with at most k items of each size. Note that S is $O(1/\varepsilon^4)$.

Let $OPT_\varepsilon(items)$ be an optimal packing of the list *items* using only bins of size greater than ε. Define a *configuration* to be a packing of a bin of specific size with a specified set of items such that the sum of the item sizes does not exceed the size of the bin. Because we are packing only items of size greater than δ, we know that at most $\lfloor 1/\delta \rfloor$ items can fit into a bin, thereby limiting the number of possible configurations to a number N which is independent of n but dependent on the number of different bin sizes greater than ε. We consider the list L_1 as the multiset

$$L_1 = \{n_1\sigma_1, n_2\sigma_2, \cdots, n_S\sigma_S\}$$

where $n_i\sigma_i$ denotes n_i items of size σ_i.

Define:

1) a_{ij} as the number of items of size σ_i that are used in the jth configuration;
2) c_j as the size of the bin used in configuration j, also called the *cost*;
3) x_j as the number of times configuration j is used in a given packing.

We have the following integer program:

$$\text{Minimize} \quad \sum_{j=1}^{N} c_j x_j$$

subject to

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{iN}x_N \geqq n_i \quad \text{for } 1 \leqq i \leqq S \quad \text{and} \quad x_j \geqq 0, \qquad 1 \leqq j \leqq N.$$

We relax the restriction to integers and solve the above problem as a linear programming problem only constraining the x_i to be nonnegative reals. Call the solution to this problem $LIN(L_1)$. Relaxing the constraints can only lower the minimum so $LIN(L_1) \leqq OPT_\varepsilon(L_1)$. It is a well-known fact that a linear programming problem with N variables and S constraints has an optimal solution with at most S nonzero values for the variables. Therefore, we can obtain a solution for which each x_i is an integer value by adding no more than S bins. Call this $PAD(L_1)$. Therefore, $PAD(L_1) \leqq LIN(L_1) + S \leqq OPT_\varepsilon(L_1) + S$. Hence we have the following:

LEMMA 2. $PAD(L_1) \leqq OPT(L_1) + O(1/\varepsilon^4)$.

5. Packing small items while excluding small bins. We now describe how our algorithm, *ALG*, provided by our scheme for a given $\varepsilon > 0$ deals with small items. Taking the packing produced by the previous steps, it uses a simple Next-Fit-like strategy and packs as many small items as possible into the already allocated bins. If there are any small items that cannot fit into the already allocated bins, it packs them into new unit-size bins, again using a Next-Fit-like strategy.

The Next-Fit-like strategy can be described in the following manner. Assume the packing produced in the previous step uses m bins, which we label $B_1, B_2, \cdots, B_m$. To pack the first small item, find the least j such that the item fits into bin B_j, $1 \leqq j \leqq m$. If no such j exists, allocate a new unit size bin and let $j = m+1$. Pack this item into bin B_j. Pack each subsequent item, if it fits, into the same bin as its predecessor. If it does not fit, examine only bins of higher index, again allocating a new unit size bin

if it does not fit into a nonempty bin. We have the following lemma concerning this strategy.

LEMMA 3. *Packing the small items is done in time* $O(n)$.

Proof. The total number of comparisons, C, used to pack these small items can be bounded from above.

$$C = O(\text{number of small items} + \text{the number of bins used}).$$

Observe that the number of bins used equals m plus the number of newly allocated unit size bins. The quantity m is clearly independent of the number of small items and the number of newly allocated bins is $O(\text{number of small items})$. □

This Next-Fit-like strategy differs from the standard Next-Fit algorithm because in our case we start with nonempty bins. We will show that this simple packing of the small items does not significantly harm the bound guaranteed by *PAD*. We also show we can exclude the small bins from the packing and still achieve the desired approximation. L denotes the original list of items.

THEOREM 1. *For a given instance* I, $ALG(L) \leqq (1+\varepsilon+4\varepsilon^2)OPT(L) + O(1/\varepsilon^4)$.

Proof. We have two cases to consider.

Case 1. No new bins are allocated when packing small items.

In order to analyze this case, we introduce another packing, $ALG'(L)$, which is not part of our algorithm but is helpful in deriving the relationship between $OPT(L)$ and $PAD(L)$.

Given an optimal packing of L using all the bin sizes of I, we create a new packing $ALG'(L)$, using only bins of size greater than ε, by taking all the items that were packed by $OPT(L)$ into small bins and repacking them into unit size bins using a Next-Fit strategy. Before we describe how $ALG'(L)$ relates to $OPT(L)$ and $PAD(L)$, we give some definitions:

1) $OPT(L)$ is the optimal packing of the list L using all bin sizes.
2) B_l is the set of bins of size greater than ε used in $OPT(L)$.
3) B_s is the set of bins of size less than or equal to ε used in $OPT(L)$.
4) L_l is L restricted to those items packed into B_l in $OPT(L)$.
5) $L_s = L - L_l$.
6) $s(L_l) = \sum_{a_i \in L_l} s(a_i)$. $s(L_s)$ is defined similarly.
7) $s(B_l) = \sum_{B_i \in B_l} s(B_i)$. $s(B_s)$ is defined similarly.

Letting $k+1$ be the number of new unit size bins created by $ALG'(L)$ when repacking L_s, we can write

$$ALG'(L) \leqq s(B_l) + k + 1. \tag{3}$$

We observe that at least k of the new bins are filled to within ε of their capacity, which allows us to bound k:

$$S(L_s) \geqq (1-\varepsilon)k,$$

which together with (3) yields

$$\begin{aligned} ALG'(L) &\leqq s(B_l) + \frac{s(L_s)}{1-\varepsilon} + 1 \\ &\leqq s(B_l) + \frac{s(B_s)}{1-\varepsilon} + 1 \\ &\leqq \frac{s(B_l)}{1-\varepsilon} + \frac{s(B_s)}{1-\varepsilon} + 1 \quad \text{for } 0 < \varepsilon < 1 \end{aligned} \tag{4}$$

$$= \frac{1}{1-\varepsilon} OPT(L) + 1$$

or

$$ALG'(L) \leqq (1+\varepsilon+2\varepsilon^2)OPT(L)+1 \tag{5}$$

for sufficiently small ε.

We now show how $ALG'(L)$ compares to $PAD(L)$. By Lemmas 1 and 2 we have

$$PAD(L_1) \leqq (1+\varepsilon^2)OPT_\varepsilon(L) + O(1/\varepsilon^4). \tag{6}$$

Because $ALG'(L)$ is an algorithm for packing L into bins of size greater than ε, we know $OPT_\varepsilon(L) \leqq ALG'(L)$ which together with (6) gives us

$$PAD(L_1) \leqq (1+\varepsilon^2)ALG'(L) + O(1/\varepsilon^4). \tag{7}$$

Since L_1 has no small pieces, $ALG(L_1) = PAD(L_1)$, which together with (7) yields

$$ALG(L_1) \leqq (1+\varepsilon^2)ALG'(L) + O(1/\varepsilon^4). \tag{8}$$

Since every item of L_1 is at least as large as its corresponding item in L_0, our algorithm uses the ALG packing of L_1 as the packing for L_0. By the assumption of this case, that packing the small items requires no new bins, we have $ALG(L_0) = ALG(L)$. These facts, together with (5) and (6), yield

$$ALG(L) \leqq (1+\varepsilon+4\varepsilon^2)OPT(L) + O(1/\varepsilon^4)$$

for sufficiently small ε.

Case 2. New bins are allocated when packing small items.

In this case we have

$$\sum_{i=1}^{n} s(a_i) \geqq (ALG(L) - s(\text{last bin})) - \delta(B-1)$$

where B is the number of bins used by ALG. Clearly, $B \leqq ALG(L)/\min$ which then gives

$$\begin{aligned}\sum_{i=1}^{n} s(a_i) &\geqq (ALG(L) - s(\text{last bin})) - \delta\frac{ALG(L)}{\min} \\ &= \left(1 - \frac{\delta}{\min}\right) ALG(L) - s(\text{last bin}) \\ &= (1-\varepsilon)ALG(L) - s(\text{last bin})\end{aligned}$$

by the definition of δ. Certainly, $OPT(L) \geqq \sum s(a_i)$, so we get

$$ALG(L) \leqq \frac{OPT(L)}{1-\varepsilon} + \frac{s(\text{last bin})}{1-\varepsilon}$$

or

$$ALG(L) \leqq (1+\varepsilon+2\varepsilon^2)OPT(L) + O(1)$$

for sufficiently small ε.

Hence we know that for any $\varepsilon > 0$ there exists an approximation algorithm ALG with the property that for any instance I of the variable bin packing problem

$$ALG(L) \leqq (1+\varepsilon+2\varepsilon^2)OPT(L) + O(1/\varepsilon^4).$$

Note that the factor of $2\varepsilon^2$ does not affect our result but is present for notational ease. We can choose an ε' as an appropriate function of ε to achieve the stated result since the bound is to hold for arbitrarily small ε.

6. Time complexity. Defining the minimum bin and item size takes a constant amount of time. To define the item sizes for the list L_1, we can sort the list in $O(n \log n)$ time or as suggested in [DL81] we can use the solution for the "selection problem" and find the special sizes $a_0, a_k, a_{2k}, \cdots, a_{\lfloor n/k \rfloor k}$ in time $O(n/\varepsilon^4)$. The linear program can be solved in time independent of n (but exponential in $1/\varepsilon$ and the number of bin sizes). As shown in Lemma 3, the next-fit-like packing of the small items is also done in linear time. Therefore, we have a polynomial approximation scheme which runs in time linear in n but exponential in $1/\varepsilon$ and the number of bin sizes.

7. Achieving a running time polynomial in $1/\varepsilon$ and the number of bin sizes. To extend this scheme to one which is polynomial in $1/\varepsilon$ and the number of bin sizes, we employ the following techniques. In order to round the item sizes, we first sort them taking time $O(n \log n)$. To solve the linear programming formulation of our problem in time polynomial in $1/\varepsilon$ and the number of bin sizes, we employ results, found in [GLS81] and [KK82], which are based upon ideas found in [Kh79]. In [GLS81] the authors prove the equivalence of two problems, the *weak optimization problem* and the *weak separation problem.* They show that a polynomial time solution for one implies a polynomial time solution for the other. Unfortunately, their definition of polynomial time requires one to solve these problems in time polynomial in $|\log \varepsilon|$. Although we are unable to achieve this running time, we show we can solve a modified separation problem in time polynomial in $1/\varepsilon$ and the number of bin sizes. This modified separation problem is very similar to the one in [KK82]. In our problem, however, we must deal with variable bin sizes. We also use a different method to deal with rounding errors than that used in [KK82]. We show that solving this modified separation problem in time polynomial in $1/\varepsilon$ and the number of bin sizes allows us to solve the desired optimization problem with the same running time and efficacy as the algorithm in [GLS81]. We use this polynomial time algorithm to solve the dual linear program for the packing problem under consideration. We introduce a method which allows us to convert a potentially infeasible solution vector to a feasible solution vector which is still within the desired optimality bound. The principle of duality then tells us this is a solution for the primal problem of original interest. The details are as follows.

7.1. The dual problem. The linear programming formulation of our problem is

$$\text{Minimize} \quad \sum_{i=1}^{N} c_i x_i$$

$$\text{subject to} \quad \sum_{j=1}^{N} a_{ij} x_j \geqq b_i, \qquad 1 \leqq i \leqq S.$$

We wish to examine the dual of this problem:

$$\text{Maximize} \quad \sum_{i=1}^{S} b_i u_i$$

$$\text{subject to} \quad \sum_{i=1}^{S} a_{ij} u_i \leqq c_j, \qquad 1 \leqq j \leqq N,$$

$$u_i \geqq 0, \qquad 1 \leqq i \leqq S.$$

(We refer to this formulation as *DUAL.*)

The dual program has the following economic interpretation [KK82]. The dual variable u_i can be considered as the price of each piece of type i. The dual program seeks to maximize the sum of the prices of all the items, subject to the constraints that

(i) the sum of the prices of all the pieces in any single bin be less than or equal to the size of that bin, and (ii) each price be nonnegative.

7.2. Definitions and assumptions. We first give the pertinent definitions and assumptions from [GLS81] and describe how and why our separation problem differs from theirs. In what follows K is a nonempty convex compact set in R^n, Q is the field of rational numbers, $d(y, K)$ denotes the Euclidean distance of y from K, and $\|c\|$ denotes the Euclidean norm of the vector c.

DEFINITION. (Weak) Optimization Problem. Given a vector $c \in Q^n$ and a number $\varepsilon > 0$, find a vector $y \in Q^n$ such that $d(y, K) \leqq \varepsilon$ and for every $x \in K$, $c^T x \leqq c^T y + \varepsilon$.

DEFINITION. (Weak) Separation Problem. Given a vector $y \in Q^n$ and a number $\varepsilon > 0$, either conclude that $d(y, K) \leqq \varepsilon$, or find a vector $c \in Q^n$ such that $\|c\| \geqq 1$ and for every $x \in K$, $c^T x \leqq c^T y + \varepsilon$.

EQUIVALENCE THEOREM [GLS81]. *Let K be a class of convex bodies. There is a polynomial time algorithm to solve the separation problem for the members of K, if and only if there is a polynomial time algorithm to solve the optimization problem for the members of K.*

One can describe their separation problem in the following way. Given a vector $y \in Q^n$ and a number $\varepsilon > 0$, determine either (i) y is within a distance ε of being in K, which of course is satisfied if $y \in K$, or (ii) y is within a distance ε of not being in K. This says we might (i) consider y feasible when it is actually infeasible, or (ii) consider y infeasible when it is actually feasible. Our modified separation problem allows for Part (i) of the GLS separation problem but prohibits Part (ii). Thus, we might consider an infeasible vector as feasible but we *never* consider a feasible vector infeasible. If we consider an infeasible vector as feasible, we take care that its objective function value is within the allowable error from the optimum.

The existence of a polynomial time algorithm for the approximate solution of a linear program with the feasible region being a convex set K is based partially on the assumption that there exist a point u_0 and $0 \leqq r \leqq R$ such that $u_0 \in K$ and

$$S(u_0, r) \leqq K \leqq S(u_0, R)$$

where $S(u_0, r)$ denotes the Euclidean ball of radius r and center u_0. The second inclusion is a natural one for most problems, namely that the set be explicitly bounded. The first inclusion is for technical reasons. In [GLS81] it is shown that without this assumption there is no algorithm to solve the optimization problem. We will show there exist appropriate u_0, r and R for our region K after we specify what it means in [GLS81] for an algorithm to possess a polynomial running time.

In [GLS81] an algorithm to solve the optimization problem over a convex region K is *polynomial* if its running time is bounded by some polynomial of the size of its input. The size of the input is defined, as usual, as the length of its binary encoding. The input to an optimization problem is a vector $c \in Q^n$, a number $\varepsilon > 0$, and u_0, r and R satisfying (11). Therefore, if we wish to invoke the equivalence theorem, we must solve their separation problem in time bounded by poly($|\log \varepsilon|$). Following [KK82], we instead solve our modified version of the separation problem. We use time polynomial in $1/\varepsilon$ and later show this is sufficient for achieving our desired results.

7.3. Finding u_0, r, R with $S(u_0, r) \leqq K \leqq S(u_0, R)$. In order to use the ellipsoid method in the manner we wish, we need to exhibit a vector u_0, and $0 < r \leqq R$ such that $S(u_0, r) \leqq K \leqq S(u_0, R)$. Our K is a convex region defined by inequalities of the form

$$a_{1j}u_1 + a_{2j}u_2 + \cdots + a_{Sj}u_S \leqq c_j, \qquad 1 \leqq j \leqq N.$$

This says that in any configuration the sum of the sizes of the items in a bin is less than or equal to the bin's size. When our approximation scheme is formulating the problem as a linear program all items have size bounded below by ε^2. So, if we let the price of each item be ε^2, then

$$a_{1j}\varepsilon^2 + a_{2j}\varepsilon^2 + \cdots + a_{Sj}\varepsilon^2 \leqq c_j, \qquad 1 \leqq j \leqq N. \tag{9}$$

We wish to have every vector within a distance r of our vector u_0 be a feasible vector. In light of (9) we let $u_0 = r = (\varepsilon^2/2)\,\vec{1}$, and have a ball satisfying

$$S(u_0, r) \leqq K.$$

We now find an R such that K is contained in $S(u_0, R)$. All vectors in K have nonnegative values in all components because the problem requires that all items have a nonnegative price. In addition we know that all bins have size less than or equal to 1 which implies any single price must be no greater than one. Letting x be an arbitrary feasible vector, we have

$$\sum_{i=1}^{S} u_i^2 \leqq S. \tag{10}$$

CLAIM. *For an arbitrary feasible vector* u, $\sum_{i=1}^{S} (u_i - \varepsilon^2/2)^2 \leqq S+1$.

Proof.

$$\begin{aligned}
\sum_{i=1}^{S} (u_i - \varepsilon^2/2)^2 &= \sum_{i=1}^{S} (u_i^2 - u_i\varepsilon^2 + \varepsilon^4/4) \\
&\leqq \sum_{i=1}^{S} (u_i^2 + \varepsilon^4/4) \\
&\leqq \sum_{i=1}^{S} u_i^2 + S\varepsilon^4/4 \\
&\leqq \sum_{i=1}^{S} u_i^2 + 1 \quad \text{since } S \leqq 1/\varepsilon^4 + 2, \\
&\leqq S+1 \quad \text{by equation (10).}
\end{aligned}$$

□

This tells us that the distance between any feasible vector u and u_0 is no greater than $\sqrt{S+1}$.

We therefore leave u_0 and r as before and set $R = \sqrt{S+1}$ and have

$$S(u_0, r) \leqq K \leqq S(u_0, R).$$

7.4. Solving the modified separation problem for *DUAL*. We now give the details of the approximate solution to our modified separation problem. (In what follows the separation problem refers to our modified separation problem.) The separation problem associated with the optimization problem *DUAL* is: given a price vector u, a number $\rho > 0$, and the convex region K defined by the constraints of *DUAL* conclude either that u is infeasible or feasible. If u is considered to be feasible, then ensure that $b^T u \leqq b^T y + \rho$ where $b^T y$ is the optimal value achieved at the solution vector y. If u is considered infeasible then provide a separating hyperplane. We can solve this separation problem in the following way.

For each bin size β solve the following knapsack problem:

$$\text{Maximize} \quad \sum_{i=1}^{S} u_i a_i$$

subject to $$\sum_{i=1}^{S} \sigma_i a_i \leqq \beta$$

where σ_i is the size of items of type i and a_i are positive integers.

If the solution vector, $\bar{a}$, is such that the maximum is greater than β, then the vector u is not feasible and $\bar{a}$ provides the necessary separating hyperplane.

In examining the statement of the optimization problem we see that care must be taken in dealing with the allowable tolerances. The vector u which we decide upon as the approximate solution of the linear program may not actually be a feasible vector. As we will soon see, what we are guaranteed is that it is almost feasible. In order to find an approximate solution which is a feasible vector, we define ρ to be $\varepsilon/(n+1)$. We show later that this is a proper choice.

We wish to solve the above knapsack problem with the necessary accuracy in time polynomial in $1/\varepsilon$. In order to achieve this, we round each component of the price vector u down so that each of its components is an integer multiple of ρn^{-1}. As in [GLS81], the calculations of this algorithm are carried out with an accuracy of $p = 20n^2\lceil \log(2R^2\|c\|/(r\varepsilon))\rceil$ binary places to the right of the radix point. We therefore compute ρn^{-1} and round that quantity to p binary places. After performing the rounding we might be left with a value that is slightly larger than the true value of ρn^{-1}, and this could have an adverse affect on the accuracy of our approximation. To compensate for this, after we round to p binary places to the right of the radix point we subtract 2^{-p}. Reducing the rounded version by 2^{-p} ensures that the version used in calculation is less than or equal to the true value of ρn^{-1}. This small reduction (recall the value of p) does not affect the time complexity of the following dynamic programming solution by an unacceptable amount. We will denote this rounded and reduced version of ρn^{-1} as λ.

We round each component, forming the new vector $\bar{u}$:

$$\bar{u}_i = \lambda \lfloor u_i/\lambda \rfloor.$$

Since λ was already rounded to p binary points, the value of $\bar{u}_i$ is accurate to within p binary points. The computation of $\bar{u}_i$ does not introduce any further precision error.

We now solve the following knapsack problem with the vector $\bar{u}$ by dynamic programming as follows. Let $F(k, U)$ be the solution to

Minimize $$\sum_{i=1}^{k} \sigma_i a_i$$

subject to $$\sum_{i=1}^{k} \bar{u}_i a_i \geqq U\lambda$$

for all integers k and U such that $1 \leqq k \leqq S$, $1 \leqq U \leqq \beta/\lambda$. We set $F(0, 0) = 0$ and let

$$F(k, U) = \min\{F(k-1, U), F(k, U - \bar{u}_k/\lambda) + \sigma_k\}.$$

The solution to the original maximization knapsack problem can be recovered from the tabulation of $F(k, U)$ by computing

$$\lambda \max\{U | F(k, U) \leqq \beta\}.$$

The tabulation of F and the recovery of the solution of the original knapsack problem can be done in time $O(S\beta/\lambda) = O(S\beta n^2/\varepsilon)$, which is polynomial in β, n and $1/\varepsilon$.

If the value of the objective function at the solution vector $\bar{a}$ is less than or equal to β, then $\bar{u}$ is not feasible; otherwise $\bar{u}$ is feasible for the bin size β. There are two cases:

(i) $\bar{u}$ is not feasible. By construction $\bar{u}_i \leqq u_i$, so we know u is not feasible. The vector $\bar{a}$ provides us the necessary separating hyperplane.

(ii) $\bar{u}$ is feasible. In this case we make an optimality cut even though u may not be feasible. However, because of the definition of $\bar{u}$ we have $u_i \leqq \bar{u}_i + \rho n^{-1}$. We also know $\sum b_i \leqq n$ by the definition of b. Therefore $b^T u \leqq b^T \bar{u} + \rho$, which insures that we do not treat a vector as feasible if its objective function value is larger than the solution value by more than ρ.

7.5. The accuracy and complexity of the modified algorithm. The main theorem of [GLS81] we are concerned with when showing that our modified algorithm has the same time complexity and accuracy as the algorithm of [GLS81] is the following.

THEOREM 2 [GLS81]. *Let j be a feasible index for which*

$$c^T x_j = \max \{c^T x_k | 0 \leqq k < N, k \text{ feasible}\}.$$

Then $c^T x_j \geqq \max \{c^T x | x \in K\} - \varepsilon$.

We only require that our modified separation problem be solved within a δ that is polynomial in the problem size whereas [GLS81] require that their separation problem be solved within a δ that is *exponentially* small in the problem size. We examine their analysis to see how these changes affect the relative performances of our algorithms.

In order to make the following discussion somewhat more self-contained, we first remind the reader of the structure of the ellipsoid method used in [GLS81]. Given a convex region K, a linear objective function $c^T x$ and u_c, r and R as described earlier, it begins by defining the initial ellipsoid E_0 as $S(u_0, R)$. At each step k there will be an ellipsoid, E_k, centered at x_k which includes the set K_k of points of K, for which $c^T x$ is at least as large as the largest solution found so far which corresponds to a point considered feasible. The center x_k is then tested for feasibility by solving the separation problem for vector x_k. If x_k is considered infeasible, we form the next ellipsoid, E_{k+1}, taking the ellipsoid of least volume, which includes the portion of E_k which included K_k. If x_k is considered feasible the next ellipsoid, E_{k+1}, is formed by excluding those vectors whose objective function value is less than that of x_k.

At each stage, the algorithm uses a subroutine to test the feasibility of the center of the current ellipsoid. The algorithm's action is based upon whether the subroutine considers the center feasible or infeasible. The algorithm does not know if the center is truly feasible (or infeasible) if the subroutine says it is feasible (infeasible). It only knows it is approximately feasible (infeasible) to within an a priori error bound. This is the reason we speak of a vector as being *considered* feasible rather than simply stating that the vector is feasible. One of the important features of this method is that we can achieve our desired results with only a knowledge of approximate feasibility at each stage.

The algorithm works with these geometrical concepts in terms of vector spaces. Given any vector x_0 and a nonsingular matrix A, we can denote an ellipsoid E as

$$E = \{x | (x - x_0)^T A^{-1} (x - x_0) \leqq 1\}$$

where x is a real valued vector. The algorithm defines the sequence of x's and A's as follows. Let $x_0 = u_0$ and $A_0 = R^2 I$. At the kth step the separation problem for x_k is solved. If x_k is considered feasible, we say k is a *feasible index* and set $a = 0$. Otherwise k is an *infeasible index* and we set $a = -d$, where d is the vector representing the separating hyperplane. The vector a is referred to as the *auxiliary* vector. Next define

$$b_k = A_k a / (a^T A_k a)^{1/2},$$
$$x_k^* = x_k + b_k (n+1)^{-1},$$
$$A_k^* = C_1 (A_k - C_2 b_k b_k^T)$$

where $C_1 = (2n^2+3)(2n^2)^{-1}$ and $C_2 = 2(n+1)^{-1}$, terms polynomial in the dimension of K. The desired elements for the next ellipsoid are x_{k+1} and A_{k+1} which are derived from x_k^* and A_k^*, respectively, by an appropriate rounding. Letting $N = 4n^2\lceil \log 2R^2\|c\|(r\varepsilon)^{-1}\rceil$, Theorem 2 says if j is the feasible index for which $c^T x_j = \max\{c^T x_k | 0 \leqq k < N,\ k \text{ feasible}\}$ then $c^T x_j \geqq \max\{c^T x | x \in K\} - \varepsilon$.

We can now indicate why Theorem 2 is also true for the modified algorithm. In the modified algorithm we form each x_k and A_k in the same way the GLS algorithm does. Therefore, Lemmas 2.1 and 2.2 of [GLS81] are true of the modified algorithm. In Lemma 2.3 of [GLS81] we encounter an important difference. In that lemma, (25) makes the claim

$$\text{for all } x \in K_{k+1}, \qquad a_k^T x \geqq a_k^T x_k - \delta$$

where a_k is the auxiliary vector for step k and K_{k+1} is that portion of K which contains those vectors x for which $c^T x$ is at least as large as the best feasible solution found through step k. Due to the definition of their separation problem we see this claim is the strongest statement they can make. If k is a feasible index, then they do not need the δ. However, if k is an infeasible index, then their separation problem only guarantees that

$$\max\{d^T x | x \in K\} \leqq d^T x_k + \delta \tag{11}$$

where d is the vector representing the separating hyperplane. In this case it is not hard to see that (11) implies the necessity of the presence of δ in (25).

In the modified algorithm a feasible vector is never considered infeasible; therefore, if k is an infeasible index, we can make the claim

$$\max\{d^T x | x \in K\} \leqq d^T x_k,$$

instead of the weaker claim of [GLS81] given by (11). Our stronger claim allows us to state $a_k^T x \geqq a_k^T x_k$ in place of (25). Using this version of (25), one can show that the t appearing in their proof has the bounds $0 \leqq t \leqq 1$ for the modified algorithm, in contrast to the bounds $-\delta(a_k^T b_k)^{-1} \leqq t \leqq 1$ in the GLS algorithm. In their proof we see the bounds for t play an important role in bounding the term $2t(1-t)(n-1)^{-1}$. Under t's bounds in the GLS algorithm this term achieves a minimum at $t = -\delta(a_k^T b_k)^{-1}$ and it is the value of this minimum that must be used in the appropriate step of the analysis in (30). (This fact is one of the primary reasons GLS require δ to be so small.) In our algorithm the aforementioned term achieves a minimum of 0. In this case we must verify that $2n^4(2n^4+n^2-3)^{-1}$ is no more than $1-1/(12n^2)$, which is true for $n > 1$.

Therefore, we find Lemmas 2.2 and 2.3 of [GLS81] are true of our modified algorithm as well. The remainder of the proof of Theorem 2 relies primarily on the assumption that there exists a vector x_0 and radii r and R such that K is contained in the Euclidean ball $S(x_0, R)$ and K contains the Euclidean ball $S(x_0, r)$. We have previously shown these conditions are satisfied by the region K of our problem. Therefore our modified algorithm will solve the optimization problem within a tolerance ρ in the desired number of iterations.

7.6. Creating a feasible solution. We know that our best feasible vector u for *DUAL* will yield a solution value that will be within ρ of the optimal. We must take into account that what we consider a feasible vector is actually an *approximately feasible* vector. If u is not feasible for *DUAL*, it is not immediately clear how we can use it to obtain a solution vector, exact or approximate, to our original problem.

We create a new solution vector v by letting $v = \max\{0, u-\rho\}$, where the maximum is taken component-wise. The new vector v is such that $d(v, K) \leqq 0$, which is to say v

is feasible. In reducing u to v we may also reduce the value of our objective function. However, due to the definitions of ρ and v we know *for all* $x \in K$,

$$\begin{aligned} b^T x &\leqq b^T u + \rho \\ &\leqq b^T v + b^T \rho + \rho \\ &\leqq b^T v + (n+1)\rho \\ &= b^T v + \varepsilon. \end{aligned}$$

Therefore the vector v is a feasible vector which maximizes the objective function of *DUAL* within a tolerance of ε.

The Principle of Duality tells us that a solution of *DUAL* corresponds to a solution of our problem of original interest. Therefore, we can solve *LIN* within ε in time polynomial in $1/\varepsilon$, and the number of different bin sizes.

8. Conclusions. We have presented an approximation scheme that takes as input an instance of the variable-sized bin packing problem. The instance is composed of a set of bin sizes, a set of items and an $\varepsilon > 0$. The approximation scheme produces as output an algorithm *ALG* which works in time polynomial in the number of items, the number of bin sizes and $1/\varepsilon$. The accuracy of *ALG* is given by $ALG(I) \leqq (1+\varepsilon)OPT(I) + O(1/\varepsilon^4)$ where $ALG(I)$ is the sum of the bin sizes used by *ALG* in the packing and $OPT(I)$ is the minimum sum of bin sizes required.

To do this, we do the following. Given an instance, we round the item sizes and disregard certain bin sizes as a function of ε. We then formulate the problem as a linear programming problem. To solve the linear programming problem within the desired time and error bounds, we employ techniques similar to those found in [GLS81] and [KK82], which are based upon ideas found in [Kh79]. The solution to the linear programming formulation is then used to obtain a solution to the original instance. We show that this solution meets the desired time and accuracy constraints.

Acknowledgment. I wish to thank George Lueker for his careful reading of an earlier version of this paper and for his many helpful suggestions.

REFERENCES

[CGJ84] E. G. COFFMAN JR., M. R. GAREY AND D. S. JOHNSON, *Approximation algorithms for bin packing—an update survey*, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini and P. Serafini, eds., Springer-Verlag, Vienna-New York, 1984, pp. 49-106.

[DL81] W. FERNANDEZ DE LA VEGA AND G. S. LUEKER, *Bin packing can be solved within* $1+\varepsilon$ *in linear time*, Combinatorica, 1 (1981), pp. 349-355.

[Ei57] K. EISEMANN, *The trim problem*, Management Sci., 3 (1957), pp. 279-281.

[FL86] D. K. FRIESEN AND M. A. LANGSTON, *Variable sized bin packing*, this Journal, 15 (1986), pp. 222-230.

[GG61] P. C. GILMORE AND R. E. GOMORY, *A linear programming approach to the cutting stock program*, Oper. Res., 9 (1961), pp. 849-859.

[GJ] M. R. GAREY AND D. S. JOHNSON, in preparation.

[GJ79] ———, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, W. H. Freeman and Co., San Francisco, 1979.

[GLS81] M. GROTSCHEL, L. LOVASZ AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169-198.

[Jo73] D. S. JOHNSON, *Near optimal bin packing algorithms*, Ph.D. thesis, Project MAC, Mass. Inst. of Technology, Cambridge, MA, 1973.

[Kh79] L. G. KHACHIYAN, *A polynomial algorithm in linear programming*, Dokl. Akad. Nauk SSSR, 244 (1979), pp. 1093-1096 (English translation: Soviet Math. Dokl., 20, pp. 191-194).

[KK82] N. KARMARKAR AND R. M. KARP, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proc. 23rd Annual Symposium on Foundations of Computer Science, 1982, pp. 312-320.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987

© 1987 Society for Industrial and Applied Mathematics
013

SCHEDULING FILE TRANSFERS FOR TREES AND ODD CYCLES*

HYEONG-AH CHOI† AND S. LOUIS HAKIMI‡

Abstract. The scheduling of file transfers in networks to minimize the overall finishing time was studied by Coffman, et al. where the schedule does not permit interruption and each communication module can be used as a transmitter and as a receiver. They first presented complexity results under various conditions. Then they showed that the general problem is NP-complete and provided approximation algorithms. This paper first presents more efficient approximation algorithms with better performances than the above authors' algorithms for the cases of trees and multitrees. Furthermore, there are simple distributed implementations of our approximation algorithms. Then this paper provides a polynomial time algorithm for finding an optimum schedule for an odd cycle, whose complexity was left as an open question by the above authors.

Key words. network, scheduling, approximation algorithms, complexity

1. Introduction. Coffman, Garey, Johnson and LaPaugh [1] studied a problem of transferring files between various pairs of nodes in networks. The nodes of the network represent communication centers and each node is capable of communicating directly with every other node. In their model [1] an instance of the problem consists of a weighted undirected multigraph $G=(V, E)$, which is called a *file transfer graph.* To each edge e in E there is assigned a positive number, $w(e)$, representing the amount of time needed to transfer the file corresponding to edge e between the nodes corresponding to the end vertices of e. To each vertex v in V there is assigned a positive integer, $\alpha(v)$, representing the number of communication modules at the node corresponding to v. It is noted that each communication module can be used as a transmitter and as a receiver. They also assumed that once the transfer of a file begins, it continues without interruption until the transfer is complete. The problem is to find a schedule of file transfers so as to minimize the overall finishing time.

Coffman, et al. [1] showed that the problem of minimizing the overall finishing time is NP-complete even if a file transfer graph $G(V, E)$ is a tree and $\alpha(v)=1$ for all $v \in V$. Then they provided an approximation algorithm which produces a schedule whose overall finishing time is no larger than 2.5 times optimum finishing time. They also studied the problem when the file transfer graph is a tree. They showed that if the file transfer graph is a simple tree (i.e., a graph without any cycles), then there exists an $O(|E| \cdot \log |E| \cdot \log (\sum_{e \in E} w(e)))$ algorithm for the schedule whose overall finishing time is at most 2.2 times the optimum finishing time. If the file transfer graph is a multitree (i.e., a graph without any cycles of length >2) and $\alpha(v)=\alpha_0$ with α_0 being a positive integer for all $v \in V$, then they showed that there exists an $O(|E| \cdot \log |E| \cdot \log (\sum_{e \in E} w(e)))$ algorithm for the schedule whose overall finishing time is at most 2.4 times the optimum finishing time. In § 2 we present more efficient approximation algorithms for the above cases with better performances. More precisely, we have approximation algorithms of complexities $O(|E| \cdot \log \alpha_{\max})$ and $O(|E| \log |E|)$, where $\alpha_{\max}=\max \{\alpha(v) | v \in V\}$, which produce schedules whose overall finishing times are at most twice the optimum finishing time for the cases when G is a simple tree or a multitree. Coffman, et al. [1] described other approximation algorithms for the above

* Received by the editors January 27, 1986; accepted for publication (in revised form) June 17, 1986. This work is supported by the National Science Foundation, Division of Electrical, Computer and Systems Engineering, Computer Engineering Program, grant ECS-84-06854.

† Department of Computer Science, Michigan State University, East Lansing, Michigan 48824.

‡ Department of Electrical and Computer Engineering, University of California, Davis, California 95616.

two cases with slightly weaker performances than our algorithms. However, the time complexities of these algorithms are exponential in $\alpha_{\max}$.

Coffman, et al. [1] raised the question of the complexity of scheduling file transfers when the file transfer graph is an odd cycle with multi-edges of equal weights. In § 3, we present an $O(|E|)$ algorithm in this case for finding an optimum schedule.

The same problem when interruption in the schedule is permitted was studied by Hajek [2], Hajek and Sasaki [3] and Choi and Hakimi [4], [5]. Another version of the problem when interruption in the schedule is permitted and transmitting and receiving modules are distinct was studied by Gopal, Bongiovanni, Bonuccelli, Tang and Wong [6] and Choi and Hakimi [7].

2. Approximation algorithms for trees. Let $G(V, E)$ be a weighted graph with $\alpha(v)$ being the number of communication modules at $v \in V$ and $w(e)$ being the amount of time required to transfer the file corresponding to edge $e \in E$. For $v \in V$, let $w(v)$ represent the sum of the weights of the edges incident at v. Let $w'(v) = w(v)/\alpha(v)$, $w'_m = \max\{w'(v) | v \in V\}$, and $w_{\max}(e) = \max\{w(e) | e \in E\}$.

THEOREM 1. *Let $G(V, E)$ be a file transfer graph which is a simple tree with $\alpha(v)$ being any positive integer for all $v \in V$. Then we have an $O(|E| \cdot \log \alpha_{\max})$ algorithm which produces a schedule with finishing time*

$$t^* \leqq w'_m + w_{\max}(e).$$

Remarks. We note that t^* is no larger than twice the optimum finishing time, since the optimum finishing time is at least $\max\{w'_m, w_{\max}(e)\}$.

Proof. In what follows, we will describe a schedule for all modules at each node of the network which satisfies the following "continuity" property: every module at each node with the possible exception of one has at most one idle period and that idle period includes the finishing time; the exceptional module may have up to two idle periods and these idle periods include the starting time and the finishing time. For each $v \in V$, let $C_1(v), C_2(v), \cdots, C_{\alpha(v)}(v)$ be the communication modules at vertex v such that $C_1(v)$ denotes the exceptional module if there is such a module at v. We pick any vertex $r \in V$ as a root vertex. Let $e_1, e_2, \cdots, e_{d(r)}$ be a set of edges incident at r. Assuming edges $e_1, e_2, \cdots, e_{i-1}$ are already scheduled, then edge e_i, $1 \leqq i \leqq d(r)$, is scheduled as follows. Suppose module $C_j(r)$ is busy for the semi-open interval $[0, t(C_j(r)))$ for $1 \leqq j \leqq \alpha(r)$. Then edge e_i is scheduled on $C_{j^*}(r)$ if $t(C_{j^*}(r)) \leqq t(C_j(r))$ for all $j \neq j^*$. We set $t(C_{j^*}(r)) \leftarrow t(C_{j^*}(r)) + w(e_i)$. After we schedule all the edges at the root vertex, we will repeat the following steps until all the remaining edges are schedules. We pick a vertex $v \in V$ such that exactly one of the edges at v was scheduled previously. There exists such a vertex, since G is a simple tree. Let $e_1, e_2, \cdots, e_{d(v)}$ be a set of edges at v and e_1 be the edge previously scheduled. We can assume without loss of generality that e_1 was scheduled on $C_1(v)$ in the interval $[t', t' + w(e_1))$. Edge e_i, $2 \leqq i \leqq d(v)$, is scheduled as follows. Assuming now that edges $e_1 \cdots e_{i-1}$ are already scheduled $(i - 1 \geqq 1)$, then we schedule e_i as follows. Suppose module $C_1(v)$ is busy for the semi-open interval $[t^0(C_1(v)), t(C_1(v)))$, which is initially equal to $[t', t' + w(e_1))$ and the module $C_j(v)$ is busy for the semi-open interval $[0, t(C_j(v)))$ for $2 \leqq j \leqq \alpha(v)$. Then edge e_i is scheduled as follows: if $t^0(C_1(v)) \geqq w(e_i)$, then e_i is scheduled on $C_1(v)$ in the interval $[t^0(C_1(v)) - w(e_i), t^0(C_1(v)))$ and set $t^0(C_1(v)) \leftarrow t^0(C_1(v)) - w(e_i)$; otherwise, e_i is scheduled on $C_{j^*}(v)$, where $t(C_{j^*}(v)) \leqq t(C_j(v))$ for all $j \neq j^*$ and set $t(C_{j^*}(v)) \leftarrow t(C_{j^*}(v)) + w(e_i)$.

We will next show that the finishing time t^* is no larger than $w'_m + w_{\max}(e) \triangleq T$. Let $e_1, e_2, \cdots, e_{d(v)}$ be a set of edges incident at $v \in V$. Suppose when edges $e_1, e_2, \cdots, e_{i-1}$ are scheduled, we have $t(C_j(v)) \leqq T$ for all j, $1 \leqq j \leqq \alpha(v)$ but after edge e_i is scheduled,

we have $t(C_{j_1}(v)) > T$ for some j_1, $1 \leqq j_1 \leqq \alpha(v)$. Let $t(C_j(v))$, for $1 \leqq j \leqq \alpha(v)$, denote the quantity after edges $e_1, e_2, \cdots e_{i-1}$ are scheduled but edge e_i is not scheduled yet. Then we have

$$t^0(C_1(v)) < w(e_i) \tag{1}$$

and

$$T - t(C_j(v)) < w(e_i) \quad \text{for } 1 \leqq j \leqq \alpha(v).$$

Summing up the above inequalities results in

$$t^0(C_1(v)) + \sum_{j=1}^{\alpha(v)} (T - t(C_j(v))) < (\alpha(v)+1) \cdot w(e_i). \tag{2}$$

As we assumed that $e_1, e_2, \cdots, e_{i-1}$ were scheduled such that $t(C_j(v)) \leqq T$ for all j, $1 \leqq j \leqq \alpha(v)$, we have that

$$t^0(C_1(v)) + \sum_{j=1}^{\alpha(v)} (T - t(C_j(v))) = T \cdot \alpha(v) - \sum_{k=1}^{i-1} w(e_k). \tag{3}$$

Substituting (3) into (2) results in

$$T \cdot \alpha(v) - \sum_{k=1}^{i-1} w(e_k) < (\alpha(v)+1) \cdot w(e_i). \tag{4}$$

Substituting $T = w'_m + w_{\max}(e)$ and $\sum_{k=1}^{i-1} w(e_k) = w(v) - \sum_{k=i}^{d(v)} w(e_k)$ into (4), we obtain

$$(w'_m + w_{\max}(e)) \cdot \alpha(v) - w(v) + \sum_{k=i}^{d(v)} w(e_k) < (\alpha(v)+1) \cdot w(e_i), \tag{5}$$

which implies that

$$w'_m \cdot \alpha(v) - w(v) + (w_{\max}(e) - w(e_i)) \cdot \alpha(v) + \sum_{k=i+1}^{d(v)} w(e_k) < 0. \tag{6}$$

This cannot hold because

$$w'_m \cdot \alpha(v) - w(v) \geqq 0, \quad w_{\max}(e) - w(e_i) \geqq 0 \quad \text{and} \quad \sum_{k=i+1}^{d(v)} w(e_k) \geqq 0.$$

This completes the proof that $t^* \leqq w'_m + w_{\max}(e)$.

To find the time complexity of this algorithm, we note that the edges $e_1, e_2, \cdots, e_{|E|}$ are scheduled one by one and by using a special data structure such as a heap one can schedule each edge e in $O(1)$ time and needs $O(\log \alpha(v))$ time to update the heap after edge e at v is scheduled. This implies the desired result.

THEOREM 2. *Let $G(V, E)$ be a file transfer graph which is a multitree with $\alpha(v)$ being any positive integer for all $v \in V$. Then we have an $O(|E| \log |E|)$ algorithm which produces a schedule with finishing time*

$$t^* \leqq 2 \cdot \max \{w_{\max}(e), w'_m\},$$

and this is no larger than twice the optimum finishing time.

Proof. We will first construct a new graph G_1 from G as follows: The vertex set $V(G_1) = V(G)$. The edge set $E(G_1)$ is defined in the following way. Let $e_1, e_2, \cdots, e_p$ be a set of edges joining two vertices u and v in $V(G)$. Let $T(u, v) = \max \{w'(u), w'(v), \max \{w(e_i) | 1 \leqq i \leqq p\}\}$. This number will remain fixed in the following procedure. We combine a pair of edges in $\{e_1, \cdots, e_p\}$ to form a new edge as follows. Select two

edges e_i and e_j such that $w(e_i) \leqq T(u, v)$ and $w(e_j) \leqq T(u, v)$, construct a new edge e_{ij} whose weight is $w(e_i)+w(e_j)(\leqq 2 \cdot T(u, v))$, remove edges e_i and e_j from the list, and put the new edge e_{ij} in the list. Repeat the above procedure until there exists at most one edge in the list whose weight is $\leqq T(u, v)$. Let the edge set $E(G_1)$ be a set of edges constructed by the above procedure. Note that for any pair of vertices u and v in $V(G_1)$ there is at most one edge joining u and v whose weight is $\leqq T(u, v)$. We observe that G_1 can be computed from G in $O(|E| \log |E|)$ time. It is easy to see that a schedule for G can be constructed from a schedule for G_1 with the finishing time for G being the same as that for G_1. We will next show how a schedule for G_1 can be constructed with finishing time at most $2 \cdot \max\{w_{\max}(e), w'_m\}$.

Let $E_1 = \{e' \in E(G_1) |$ if $e' = (u, v)$, then $w(e') > T(u, v)\}$. We note that

$$w(e') \leqq 2 \cdot \max\{w_{\max}(e), w'_m\} \quad \text{for all } e' \in E_1.$$

Each edge in E_1 will be scheduled from time 0 until the schedule for that edge is completed and the communication modules used for the scheduling of these edges will never be used for any other purpose. We need first to show that there are enough communication modules at each vertex to schedule the edges in E_1. Let $e_1, e_2, \cdots, e_q$ be a set of edges in E_1 at v. Then we have

$$\sum_{i=1}^{q} w(e_i) \geqq w'(v) \cdot q. \tag{1}$$

As $\sum_{i=1}^{q} w(e_i) \leqq w(v) = w'(v) \cdot \alpha(v)$, (1) implies that

$$q \leqq \alpha(v). \tag{2}$$

To schedule the remaining edges we proceed as follows. Let G_2 be a graph such that $V(G_2) = V(G_1)$; $\alpha_2(v) = \alpha(v) - b(v)$ for $v \in V(G_2)$, where $b(v)$ represents the number of edges in E_1 at v; and $E(G_2) = E(G_1) - E_1$. Then we have

$$\begin{aligned} w_2(v) &= w(v) - \sum\{w(e) | e \text{ is in } E_1 \text{ at } v\} \\ &\leqq w(v) - b(v) \cdot w'(v) \\ &= \alpha(v) \cdot w'(v) - b(v) \cdot w'(v). \end{aligned} \tag{3}$$

If $w_2(v) = 0$, then the schedule at v is already complete and need not be considered. Thus, we assume that $w_2(v) > 0$ for all $v \in V(G_2)$. As $\alpha_2(v) = \alpha(v) - b(v)$, (3) implies that

$$w'_2(v) = \frac{w_2(v)}{\alpha_2(v)} \leqq \frac{w'(v)(\alpha(v) - b(v))}{\alpha(v) - b(v)} = w'(v). \tag{4}$$

This implies that $\max\{w'_2(v) | v \in V\} \leqq w'_m$. This, together with the facts that G_2 is a simple tree (possibly disconnected) and $\max\{w(e) | e \in E(G_2)\} \leqq \max\{w'_m, w_{\max}(e)\}$, by Theorem 1 implies that there exists a schedule for G_2 with finishing time

$$t_2^* \leqq w'_m + \max\{w'_m, w_{\max}(e)\} \leqq 2 \cdot \max\{w'_m, w_{\max}(e)\}.$$

This completes the proof that G can be scheduled with finishing time

$$t^* \leqq 2 \cdot \max\{w'_m, w_{\max}(e)\}.$$

To find the time complexity of the algorithm we note that $O(|E| \log |E|)$ time is needed to construct a graph G_1 from G and the scheduling of edges in G_1 takes

$O(|E(G_1)|)$ time. By Theorem 1, G_2 can be scheduled in

$$O(|E(G_2)| \cdot \log \alpha_{\max}).$$

This implies the desired result.

Finally it must be noted that we have implicitly assumed that there is a central processor which collects all information and then uses these algorithms to compute the schedules. However, in many applications of the problem, no central processor may exist. Then the schedule must be computed in a distributed manner by the processors available at the communication centers using only locally available information. It is easy to see that the algorithm suggested in Theorem 1 requires no central processor, which implies that the algorithm can be executed in a distributed manner. The distributed version of the algorithm in Theorem 2 requires, besides the local information at each vertex u, $w'(v)$ from each vertex v adjacent to u. Furthermore for each set of parallel edges it must be decided which of the two processors at the end vertices of this set of edges will be engaged in computing G_1. This can be done, for example, by always choosing the processor at the node with the lower index.

3. Optimum algorithm for odd cycles.

THEOREM 3. *Let $G(V, E)$ be a file transfer graph which is an odd cycle with multi-edges of equal weights and with $\alpha(v)$ being any positive integer for all $v \in V$. Then we have an $O(|E|)$ algorithm which produces a schedule with finishing time*

$$w_0 \cdot \max\left\{\max_{v\in V}\left\{\left\lceil \frac{d(v)}{\alpha(v)}\right\rceil\right\}, \left\lceil \frac{|E|}{\lfloor \sum_{v\in V} \alpha(v)/2\rfloor}\right\rceil\right\} \triangleq t^*,$$

where w_0 represents the weight of each edge in G, $d(v)$ represents the number of edges at v, and $|E|$ represents the number of edges in G. There is no such schedule whose overall finishing time is $<t^*$.

Proof. The timespan cannot be less than $w_0 \cdot \lceil d(v)/\alpha(v)\rceil$ for any $v \in V$. Also a pair of modules is required to perform any task, thus we may perform at most $\lfloor \sum_{v\in V} \alpha(v)/2\rfloor$ pairs of tasks simultaneously. As there are $|E|$ tasks with weights being w_0, the timespan cannot be less than $w_0 \cdot \lceil |E|/\lfloor \sum_{v\in V} \alpha(v)/2\rfloor\rceil$. This proves that there exists no schedule whose finishing time is $<t^*$.

We will first construct a new graph G_1 from G as follows: The vertex set $V(G_1) = V(G)$. The edge set $E(G_1)$ is defined in the following way. Let

$$T = \max\left\{\max_{v\in V}\left\lceil \frac{d(v)}{\alpha(v)}\right\rceil, \left\lceil \frac{|E|}{\lfloor \sum_{v\in V} \alpha(v)/2\rfloor}\right\rceil\right\}.$$

This number will remain fixed in the following procedure. Let $E(u, v) = \{e_1, e_2, \cdots, e_p\}$ be a set of edges in $E(G)$ joining two vertices u and v in $V(G)$. We perform the following procedure: if $|E(u, v)| \geqq T$, remove T edges from $E(u, v)$ and place an edge in $E(G_1)$ which joins u and v with weight $w_0 \cdot T$. Repeat the above procedure until there are less than T edges joining any pair of adjacent vertices in the remaining graph. Each edge in $E(G_1)$ will be scheduled from time 0 until the schedule for that edge is complete and the communication modules used for the scheduling of these edges will never be used for any other purpose. We need first to show that there are enough communication modules at each vertex to schedule the edges in $E(G_1)$. Let $e_1, e_2, \cdots, e_q$ be a set of edges in $E(G_1)$ at v. Then we have

$$\sum_{i=1}^{q} w(e_i) = q \cdot T \cdot w_0. \tag{1}$$

As

$$\sum_{i=1}^{q} w(e_i) \leqq d(v) \cdot w_0 \leqq \left\lceil \frac{d(v)}{\alpha(v)} \right\rceil \cdot \alpha(v) \cdot w_0 \leqq \cdot T \cdot \alpha(v) \cdot w_0,$$

(1) implies that

$$q \leqq \alpha(v). \tag{2}$$

To schedule the remaining edges we proceed as follows. Let G_2 be a graph such that $V(G_2) = V$; $\alpha_2(v) = \alpha(v) - b(v)$ for $v \in V(G_2)$, where $b(v)$ represents the number of edges in $E(G_1)$ at v; and $E(G_2) = E - E(G_1)$. Let $d_2(v)$ be degree of vertex v in G_2. If $d_2(v) = 0$, then the schedule at v is already complete and need not be considered. Thus, we assume that $d_2(v) > 0$ for all $v \in V(G_2)$. Then we have

$$\begin{aligned} d_2(v) &= d(v) - b(v) \cdot T \\ &\leqq d(v) - (\alpha(v) - \alpha_2(v)) \cdot \left\lceil \frac{d(v)}{\alpha(v)} \right\rceil. \end{aligned} \tag{3}$$

Equation (3) implies that

$$\left\lceil \frac{d_2(v)}{\alpha_2(v)} \right\rceil \leqq \left\lceil \frac{d(v)}{\alpha(v)} \right\rceil. \tag{4}$$

Suppose $\alpha_2(v') = 0$ for some $v' \in V(G_2)$. Then by (3), $d_2(v') = 0$. As G was a cycle, this means that G_2 must be a path (with multi-edges) and by Theorem 4 of [1], there exists an $O(|E(G_2)|)$ algorithm with finishing time $w_0 \times \max_{v \in V} \{\lceil d_2(v)/\alpha_2(v) \rceil\}$, which is no larger than t^*. Thus assume that $\alpha_2(v) \geqq 1$ for all $v \in V(G_2)$. Note that if G_2 is a path (with multi-edges), then again the above argument would complete our proof. Thus assume that G_2 is a cycle. Suppose $\alpha_2(v') > 1$ for some $v' \in V(G_2)$. Let u_1 and u_2 be two vertices adjacent to v' in $V(G_2)$. Let $E_2(u_1, v')$ and $E_2(v', u_2)$ be a set of edges joining vertices u_1 and v' and, respectively, v' and u_2 in $V(G_2)$. Then we can break the cycle into a path by splitting v' into v_1' and v_2', placing $|E_2(u_1, v')|$ and $|E_2(v', u_2)|$ edges joining u_1 and v_1' and, respectively, v_2' and u_2, and setting $\alpha_2(v_1') = \alpha_2(v_2') = 1$. Now we can apply Theorem 4 of [1] to the resulting path. It is easy to see this process would yield the desired result. Thus, assume that $\alpha_2(v) = 1$ for all $v \in V(G_2)$. Then by Theorem 6 of [1], there exists an $O(|E(G_2)|)$ algorithm with finishing time

$$w_0 \cdot \max\left\{ \max_{v \in V(G_2)} \{d_2(v)\}, \left\lceil \frac{|E(G_2)|}{\lfloor |V(G_2)|/2 \rfloor} \right\rceil \right\} \triangleq t_2.$$

To complete the proof of Theorem 1, we only need to show that $t_2 \leqq t^* = w_0 \cdot T$. As (4), with the fact that $\alpha_2(v) = 1$ for all $v \in V(G_2)$, implies that $\max_{v \in V(G_2)} \{d_2(v)\} \leqq T$, we only need to show that $\lceil |E(G_2)| / \lfloor |V(G_2)|/2 \rfloor \rceil \leqq T$. Note that

$$\begin{aligned} |E(G_2)| &= |E| - \frac{\sum_{v \in V} b(v)}{2} \cdot T \\ &\leqq \left\lfloor \frac{\sum_{v \in V} \alpha(v)}{2} \right\rfloor \cdot \left\lceil \frac{|E|}{\lfloor \sum_{v \in V} \alpha(v)/2 \rfloor} \right\rceil - \frac{\sum_{v \in V} b(v)}{2} \cdot T \\ &\leqq T\left(\left\lfloor \frac{\sum_{v \in V} \alpha(v)}{2} \right\rfloor - \frac{\sum_{v \in V} b(v)}{2} \right) \end{aligned} \tag{5}$$

and

$$|V(G_2)| = \sum_{v \in V(G_2)} \alpha_2(v) = \sum_{v \in V} \alpha(v) - \sum_{v \in V} b(v). \tag{6}$$

From (5) and (6) and the fact that $\sum_{v\in V} b(v) = \text{even}$, we obtain

$$\frac{|E(G_2)|}{\lfloor |V(G_2)|/2 \rfloor} \leqq T\left(\left\lfloor \frac{\sum_{v\in V}\alpha(v)}{2} \right\rfloor - \frac{\sum_{v\in V} b(v)}{2}\right) \Big/ \left(\left\lfloor \frac{\sum_{v\in V}\alpha(v)}{2} \right\rfloor - \frac{\sum_{v\in V} b(v)}{2}\right) \tag{7}$$
$$= T.$$

As T is an integer, (7) implies that

$$\left\lceil \frac{|E(G_2)|}{\lfloor |V(G_2)|/2 \rfloor} \right\rceil \leqq T.$$

This completes the proof that $t_2 \leqq t^*$.

To find the time complexity of this algorithm, we note that $O(|E|)$ time is needed to construct a graph G_1 from G and the scheduling of edges in G_1 takes $O(|E(G_1)|)$ time. By Theorem 6 of [1], G_2 can be scheduled in $O(|E(G_2)|)$ time. This implies the desired result.

It may be worth mentioning that the above procedure may be also applied to "even cycles" with multi-edges of equal weights, but then the time complexity of the resulting algorithm is the same as the algorithm suggested by Coffman, et al. for such even cycles.

REFERENCES

[1] E. G. COFFMAN, JR., M. R. GAREY, D. S. JOHNSON AND A. S. LAPAUGH, *Scheduling file transfers*, this Journal, 14 (1985), pp. 744-780.

[2] B. HAJEK, *Link schedules, flows, and the multichromatic index of graphs*, Proc. 1984 Conf. on Information Sciences and Systems, March 1984, pp. 498-502.

[3] B. HAJEK AND G. SASAKI, *Link schedules in polynomial time*, preprint, Dept. of Elec. and Comput. Engrg. and the Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL.

[4] H.-A. CHOI AND S. L. HAKIMI, *Scheduling data transfers in networks with preemptions*, Proc. of 23rd Allerton Conference on Communication, Control, and Computing, October 1985.

[5] ———, *Data transfers in networks with transceivers*, submitted for publication.

[6] I. S. GOPAL, G. BONGIOVANNI, M. A. BONUCCELLI, D. T. TANG AND C. K. WONG, *An optimal switching algorithm for multibeam satellite systems with variable bandwidth beams*, IEEE Trans. Comm., COM-30, November 11, 1982, pp. 2475-2481.

[7] H.-A. CHOI AND S. L. HAKIMI, *Data transfers in networks*, submitted for publication.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
014

STABILITY OF A QUEUEING SYSTEM WITH CONCURRENT SERVICE AND LOCKING*

C. A. COURCOUBETIS†, M. I. REIMAN† AND B. SIMON‡

Abstract. Resource sharing systems, such as database management systems, utilize various types of locking to maintain consistency. Most locking mechanisms cause some resources to remain idle at certain times when there is work for them to do, inducing a decrease in the system's capacity. This decrease of capacity is reflected in the stability condition for the locking system as compared to the system without locking.

We consider the following locking system. There are N servers operating in parallel and two types of incoming customers. The first type corresponds to simple customers, i.e., customers with no locking requirements, and the second corresponds to customers that have to be processed simultaneously by all N servers. When a server is ready to serve such a customer, it has to wait until all servers are ready to serve that same customer.

We determine a necessary and sufficient condition for stability of this system, which can be expressed in terms of the mean of the maximum of N random variables, each representing the amount of work due to simple customers arriving at a station between successive locking customers. For a particular case we provide an asymptotic analysis which indicates that the wasted capacity in such a system grows as $\log N$.

Key words. database systems, performance, asymptotics

AMS(MOS) subject classifications. 60K, 68M, 90B

1. Introduction and summary.

Locking procedures have been widely proposed to solve inconsistency problems in resource sharing systems, database management systems being the most important candidate. Consider a system consisting of N files, and transactions arriving with the intention to update these files. Such a system can be modeled as an N-server FCFS queueing system, where each server has its own queue. Servers correspond to files and customers in a server's queue correspond to update requests to that file made by some transaction. Locking occurs when the processing of a number of updates to different files *must* be performed simultaneously, and has been introduced to enforce a notion of consistency called "serializability" in database management systems (see Yannakakis, Papadimitriou and Kung (1979)). One way to model locking in the system mentioned above is as follows. The servers are coupled so that if request (customer) r_i at server i must be processed together with request (customer) r_j at server j, and r_i is ready to be processed before r_j can do so, it will block its server and wait until r_j reaches the head of the queue in station j, at which point both will begin processing (service). The obvious effect that locking introduces to the performance of such a system is wasted service capacity, since at times servers will be blocked even if there is work to be done. The goal of this paper is to investigate and quantify this amount of wasted work in terms of the new stability condition that locking induces.

To our knowledge, there is no result in the literature that deals with the performance of locking systems, where the generic property of locking to couple servers in the way mentioned above is modeled. Most models considered so far (see Shum and Spirakis (1981), Goodman, Suir and Tay (1983), Mitra and Weinberger (1984)) make the assumption that a customer requesting a set of servers simultaneously will be rejected or restarted at a later time if not all servers are available at request time. One can

* Received by the editors August 30, 1984; accepted for publication (in revised form) May 29, 1986.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.
‡ AT&T Information Systems, Denver, Colorado 80234.

obtain the expected number of rejected transactions as a measure of performance for such a system. The important characteristic of our model is that an arriving customer is forced to wait (queue) in the system on a FCFS basis and block the servers it acquires while waiting, until all the servers it needs are available.

There are several other papers in the literature which consider queueing systems with some similarity to ours. Our aim is not to survey such work, so we merely mention them here. Coffman, Pollak, Gelenbe and Wood (1981), and Baccelli and Coffman (1982) consider a system with m servers and two types of customers: "Reads" and "Writes". "Writes" uses all m servers, while "Reads" uses only one server each, without concern for its identity. Each type of customer has its own queue, and "Writes" has priority over "Reads". The system analyzed in Green (1980) also has m servers, and has one queue. Customers require a random number of servers, again without distinguishing the exact identity of the servers. One major difference between these models and ours is that in our model customers distinguish between the servers, making it necessary to have a separate queue for each server. This makes the analysis completely different from the above papers.

The queueing model will be discussed in detail in § 2. We now give a brief description of the model. We consider a queueing system with N servers (see Fig. 1), each with its own queue. Each server, serving in FCFS order, has an input stream associated with it, which is assumed to be a Poisson process. These customers, which we call "ordinary" customers, have associated sequences of service times, which for each server are assumed to be i.i.d. Service time distributions can differ from server to server. These customers correspond to update requests with no locking requirements,

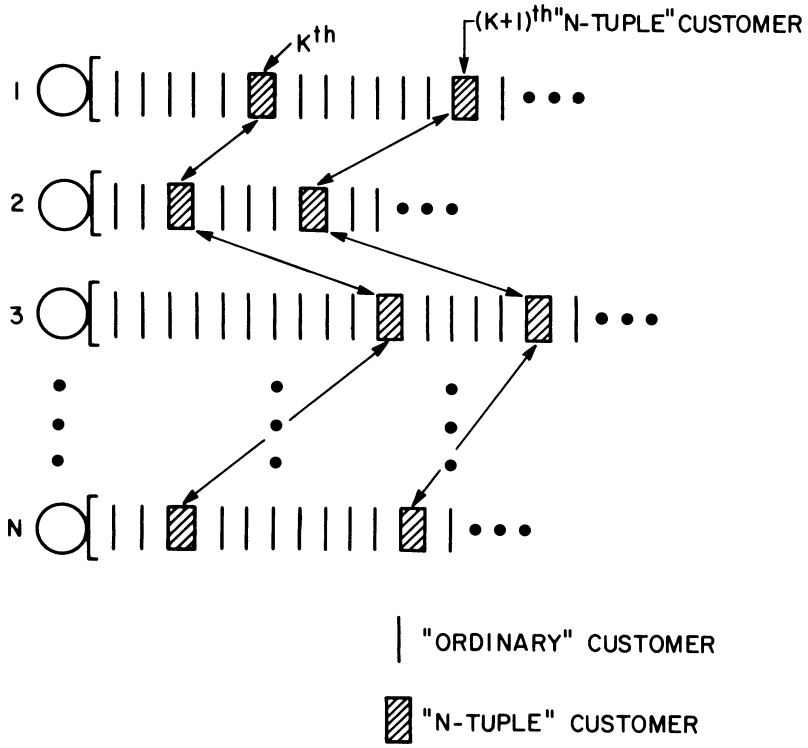


FIG. 1

and there is a stream of such requests for each file in the system. The update requests with locking requirements are modeled as an additional input stream of customers who join all queues simultaneously. These customers, which we call "N-tuple" customers, are assumed to arrive in a renewal process and have an associated i.i.d. sequence of service times. The service time of an "N-tuple" customer is the same at each station, so that an "N-tuple" customer is "done" when all its components are "done," and then all leave the system simultaneously. The locking is modeled with the additional condition that each "N-tuple" customer will be served simultaneously by all N servers. This forces the following scenario: when an "N-tuple" customer is at the head of the queue in some of the stations, and the server becomes idle, it will block (lock-up) the server until all other servers are in this configuration, at which time service begins on the "N-tuple" customer by all servers.

A particular example of this model is a system consisting of N identical copies of a fully duplicated database, each copy being physically separated from the others. The "Ordinary" customers of our model correspond to "read" requests and the "Locking" customers to "write" requests.

In § 3 we determine a necessary and sufficient condition for stability of this system, which can be translated to the maximum arrival rate of work that can be "handled" by the system. By comparing this to the well-known stability condition when no locking is imposed, one can determine the amount of the wasted service capacity due to locking. We consider a system to be stable if the event $\{W^\rho = 0\}$ is positive recurrent, where W^ρ represents the waiting in the queue of the ρth "N-tuple" customer. In other words, we require the busy periods of our system to have finite expectation. It turns out that the stability condition is of the form $\beta + \gamma < \alpha$, where α is the expected interarrival time between two "N-tuple" customers, γ is the expected work of an "N-tuple" customer, and β the expectation of the maximum among the N servers of the work due to "ordinary" customers that arrived between two successive "N-tuple" customers. This definition for β is intuitively plausible, since the waiting of an "N-tuple" customer will be caused by the server with the largest amount of work in queue upon its arrival.

In § 4 we provide an expression for β in terms of the more basic data of the problem. In addition, we derive some asymptotic results which indicate how β grows with N. For the special case we consider, the asymptotics indicate that the wasted service capacity grows as $\log N$.

These results can be interpreted in the case of a system consisting of a large number of files and two types of transactions: those which deal with a single file at a time and those which need more than one file simultaneously. Unfortunately the "locking granule" in our system is very large: it consists of the whole system. Hence even if the later transactions need a subset of the N files, the system treats them as if they needed all N of them. An interesting extension could be the case where "m-tuple" customers are permitted where $m \leqq N$. It would be interesting to determine how much better such a system performs as a function of m.

2. The queueing model. We now give a precise definition of the queueing model used. We assume that there is a probability space which contains all of the random variables we define. There are N servers in the system. Associated with server i $(1 \leqq i \leqq N)$, there is a Poisson process, $A_i = \{A_i(t), t \geqq 0\}$ with rate λ_i. This is the arrival process of "ordinary" customers to station i. The interarrival times for "ordinary" customers at station i, $\{u_i^j, j \geqq 1\}$, thus form an i.i.d. sequence of exponentially distributed random variables with parameter λ_i. The arrival process and interarrival times

are related by

$$A_i(t)=\begin{cases} 0, & u_i^1 > t, \\ \max\left\{k>0: \sum_{j=1}^{k} u_i^j \leqq t\right\}, & u_i^1 \leqq t, \end{cases} \quad t \geqq 0, 1 \leqq i \leqq N.$$

The "ordinary" customers at station i have i.i.d. service times taken from the sequence $\{\nu_i^j, j \geqq 1\}$, with distribution function B_i.

The "N-tuple" customers arrive in the system with i.i.d. interarrival times forming the sequence $\{t^\rho, \rho \geqq 1\}$ and having distribution function G.[1] Upon its arrival, an "N-tuple" customer splits into N components, each joining one of the N queues. Each station serves FCFS. When a component of an "N-tuple" customer reaches its server it will block the server unless all other $N-1$ components are also ready to be served. In this case all N components will be processed simultaneously with equal service times at all servers, corresponding to the service of the "N-tuple" customer. The service times, $\{x^\rho, \rho \geqq 1\}$, of the "N-tuple" customers are i.i.d. with distribution function H.

Another assumption is that all of the sequences of random variables defined above are assumed to be mutually independent. In what follows, we will derive an expression for W^ρ, the waiting time of the ρth "N-tuple" customer (not including service).

Let T^ρ and U_i^n be the arrival time for the ρth "N-tuple" customer and the nth "ordinary" customer, respectively, to station i. Then

$$T^\rho = \sum_{\sigma=1}^{\rho} t^\sigma, \qquad \rho \geqq 1,$$

and

$$U_i^n = \sum_{j=1}^{n} u_i^j, \qquad 1 \leqq i \leqq N, \quad n \geqq 1.$$

We will let K_i^ρ denote the number of "ordinary" customers that arrived before the ρth "N-tuple" customer at station i:

$$K_i^0 = 0, \quad K_i^\rho = A_i(T^\rho), \quad 1 \leqq i \leqq N, \quad \rho \geqq 1.$$

From this it follows that the ρth "N-tuple" customer arrives after the K_i^ρth and before the $(K_i^\rho+1)$th "ordinary" customer at station i (see Fig. 2). Note that, using this

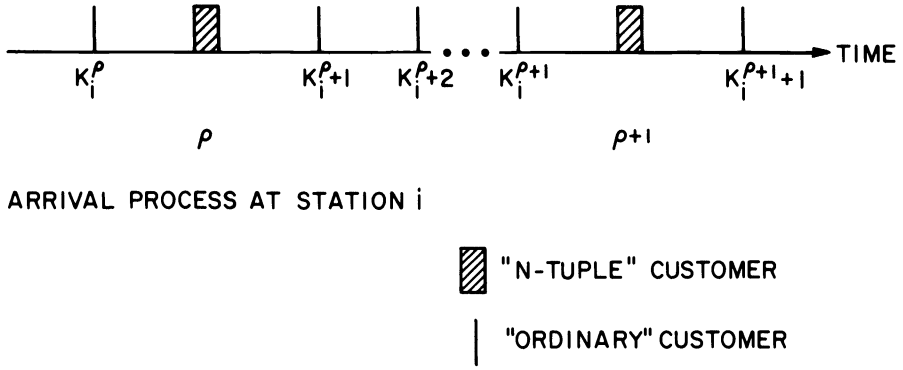


FIG. 2

[1] In what follows, we will use superscripts (subscripts) ρ, σ for random variables associated with "N-tuple" customers, and n, i, j for "ordinary" customers.

definition, if an "N-tuple" customer arrives simultaneously with an "ordinary" customer, the ordinary customer is considered to have arrived first.

In order to construct the waiting times of the "N-tuple" customers, we must also determine the waiting times of "ordinary" customers. Let w_i^n be the waiting time (in the queue) of the nth "ordinary" customer in station i, and z_i^ρ denote the time the ρth "N-tuple" customer will spend in the queue at station i, from the time it arrives until it gets in the front of the queue with an idle server (i.e. all preceding customers have been served). In direct analogy to Lindley (1952), let

$$r_i^n = \nu_i^n - u_i^{n+1}, \tag{2.1}$$

$$\hat{r}_i^\rho = \nu_i^{K_i^\rho} - [T^\rho - U_i^{K_i^\rho}], \qquad \rho \geqq 1 \tag{2.2}$$

where $T^\rho - U_i^{K_i^\rho}$ denotes the interarrival time between the K_i^ρth "ordinary" customer and the ρth "N-tuple" customer in station i. Then for the ρth "N-tuple" customer we have:

$$z_i^\rho = \begin{cases} [w_i^{K_i^\rho} + \hat{r}_i^\rho]^+, & K_i^\rho > K_i^{\rho-1}, \\ [W^{\rho-1} + x^{\rho-1} - t^\rho]^+, & K_i^\rho = K_i^{\rho-1}, \end{cases} \tag{2.3}$$

since it arrives after the K_i^ρth "ordinary" customer (when $K_i^\rho > K_i^{\rho-1}$), or the $(\rho-1)$th "N-tuple" customer (when $K_i^\rho = K_i^{\rho-1}$). The waiting time of the ρth "N-tuple" customer is

$$W^\rho = \max_{1 \leqq i \leqq N} z_i^\rho, \qquad \rho \geqq 1, \tag{2.4}$$

since an "N-tuple" customer can enter service only when all N servers become free. Similarly, for the "ordinary" customers we have:

$$w_i^1 = 0, \qquad K_i^1 \geqq 1, \tag{2.5}$$

$$w_i^{n+1} = [w_i^n + r_i^n]^+, \qquad K_i^\rho + 2 \leqq n+1 \leqq K_i^{\rho+1}, \tag{2.6}$$

and

$$w_i^n = [W^\rho + x^\rho - [U_i^n - T^\rho]]^+, \quad n = K_i^\rho + 1, \quad K_i^{\rho+1} > K_i^\rho. \tag{2.7}$$

Note that (2.6) involves all "ordinary" customers which arrived between the ρth and the $(\rho+1)$th "N-tuple" customer at station i, and wait behind some "ordinary" customer. Since the $(K_i^\rho+1)$th "ordinary" customer waits behind the ρth "N-tuple" customer, its waiting can be expressed in terms of the (total) waiting and service time of this customer, and their interarrival time, as in (2.7).

The following additional definitions will be needed later. Let

$$V_i^\rho = \begin{cases} \sum_{j=K_i^{\rho-1}+1}^{K_i^\rho} \nu_i^j, & K_i^{\rho-1} < K_i^\rho, \\ 0, & K_i^{\rho-1} = K_i^\rho, \end{cases} \qquad \rho \geqq 1,$$

and

$$V^\rho = \max_{1 \leqq i \leqq N} \{V_i^\rho\}, \qquad \rho \geqq 1. \tag{2.8}$$

Intuitively, the station corresponding to V^ρ will cause the waiting of the ρth "N-tuple" customer. Finally, define

$$\alpha = E[t^1], \quad \beta = E[V^1] \quad \text{and} \quad \gamma = E[x^1]. \tag{2.9}$$

3. Stability. The proof of stability consists of constructing two systems on the same probability space as the original system for which the associated waiting time processes $\{\bar{W}^\rho, \rho \geqq 1\}$ and $\{\underline{W}^\rho, \rho \geqq 1\}$ form upper and lower bounds, respectively, for the waiting time process $\{W^\rho, \rho \geqq 1\}$ of the original system. These orderings will be for every sample path. We will show that the condition $\beta + \gamma < \alpha$ is a sufficient condition for stability of the upper bound system and a necessary and sufficient condition for the stability of the lower bound system, and hence a necessary and sufficient condition for stability of the original system.

3.1. The upper bound system. In this system, all ordinary customers arriving (in the original system) between two successive "N-tuple" customers are withheld from entering until the moment immediately prior to the arrival of the latter "N-tuple" customer. The result is that each "N-tuple" customer in this system experiences a larger waiting time than in the original system. The random variables of this system can be defined in direct analogy to the previous section (an overbar will characterize random variables of the upper bound system):

$$\bar{w}_i^1 = 0, \qquad K_i^1 \geqq 1, \tag{3.1}$$

$$\bar{w}_i^n = \bar{w}_i^{K_i^\rho + 1} + \sum_{j=K_i^\rho+1}^{n-1} \nu_i^j, \quad K_i^\rho + 1 < n \leqq K_i^{\rho+1}, \quad \rho \geqq 0, \tag{3.2}$$

$$\bar{z}_i^\rho = \begin{cases} \bar{w}_i^{K_i^\rho} + \nu_i^{K_i^\rho}, & K_i^\rho > K_i^{\rho-1}, \\ [\bar{W}^{\rho-1} + x^{\rho-1} - t^\rho]^+, & K_i^\rho = K_i^{\rho-1}, \end{cases} \tag{3.3}$$

$$\bar{W}^\rho = \max_{1 \leqq i \leqq N} \bar{z}_i^\rho, \quad \text{and} \tag{3.4}$$

$$\bar{w}_i^{K_i^\rho + 1} = [\bar{W}^\rho + x^\rho - t^{\rho+1}]^+, \quad \rho \geqq 1, \quad K_i^{\rho+1} > K_i^\rho. \tag{3.5}$$

3.2. The lower bound system. In this system all ordinary customers arriving (in the original system) between two successive "N-tuple" customers are put in the system immediately after the arrival of the former "N-tuple" customer. Consequently, each "N-tuple" customer waits less than in the ordinary system. We proceed with the formal definition of this system (an underbar characterizes random variables of the lower bound system):

$$\underline{w}_i^1 = 0, \qquad K_i^1 \geqq 1, \tag{3.6}$$

$$\underline{w}_i^n = \underline{w}_i^{K_i^\rho + 1} + \sum_{j=K_i^\rho+1}^{n-1} \nu_i^j, \qquad K_i^\rho + 1 < n \leqq K_i^{\rho+1}, \quad \rho \geqq 0, \tag{3.7}$$

$$\underline{z}_i^\rho = \begin{cases} [\underline{w}_i^{K_i^\rho} + \nu_i^{K_i^\rho} - t^\rho]^+, & K_i^\rho > K_i^{\rho-1}, \\ [\underline{W}^{\rho-1} + x^{\rho-1} - t^\rho]^+, & K_i^\rho = K_i^{\rho-1}, \end{cases} \tag{3.8}$$

$$\underline{W}^\rho = \max_{1 \leqq i \leqq N} \underline{z}_i^\rho \quad \text{and} \tag{3.9}$$

$$\underline{w}^{K_i^\rho + 1} = \underline{W}^\rho + x^\rho, \quad \rho \geqq 1, \quad K_i^{\rho+1} > K_i^\rho. \tag{3.10}$$

3.3. The bounding argument. The following proposition contains the bounding argument.

PROPOSITION 3.1. *For every sample path,* $\underline{W}^\rho \leqq W^\rho \leqq \bar{W}^\rho$ *for all* $\rho \geqq 1$.

Since Proposition 3.1 is obvious given the definition of the processes involved, we merely sketch the proof by stating three lemmas which contain the essence of the argument.

LEMMA 1. *Let $\underline{y}, y, \bar{y}$ be real numbers such that $\underline{y} \leqq y \leqq \bar{y}$, and let $\{a_i, i \geqq 1\}, \{b_i, i \geqq 1\}$ be sequences of nonnegative real numbers. Then*

$$\left[\underline{y} + \sum_1^n b_i - \sum_1^n a_i\right]^+ \leqq [\cdots [y + b_1 - a_1]^+ + \cdots + b_n - a_n]^+ \leqq \left[\bar{y} - \sum_1^n a_i\right]^+ + \sum_1^n b_i, \; n \geqq 1.$$

LEMMA 2. *Consider the following situation occurring in some service station of our system. There is some initial work A at time 0. Then n ordinary customers arrive with service times $b_1, \cdots, b_n$, followed by an "N-tuple" customer. Let $a_0, a_1, \cdots, a_n$ be the interarrival times of this system. Denote by z the waiting of the "N-tuple" customer (to reach the front of the queue with idle servers). Then*

$$\bar{z} = \left[A - \sum_0^n a_j\right]^+ + \sum_1^n b_j,$$

$$z = [\cdots [A - a_0]^+ + \cdots + b_n - a_n]^+,$$

$$\underline{z} = \left[A - \sum_0^n a_j + \sum_1^n b_j\right]^+,$$

each equation corresponding to the discipline followed in the Upper Bound System, Ordinary System and Lower Bound System, respectively.

Proof. Straightforward by using the definitions of these systems in (2.1)-(2.5) and (3.1)-(3.10). □

LEMMA 3. *Assume the situation of Lemma 2, the only difference being that the initial work of the server is different for the three systems. Then $\underline{A} \leqq A \leqq \bar{A}$ implies*

$$\underline{z} \leqq z \leqq \bar{z}.$$

Proof. Use Lemma 1 with $\underline{y} = \underline{A} - a_o$, $y = A - a_o$, $\bar{y} = \bar{A} - a_o$. □

We will now provide the stability conditions of the two bounding systems. The α, β, γ are as defined in (2.9). We consider first the Lower Bound System.

PROPOSITION 3.2. *If $\beta + \gamma < \alpha$, then $\{\underline{W}^\rho = 0\}$ is positive recurrent. If $\beta + \gamma = \alpha$, then $\{\underline{W}^\rho = 0\}$ is null recurrent. If $\beta + \gamma > \alpha$, then $\{\underline{W}^\rho = 0\}$ is transient.*

Proof. This system corresponds to a single server queue whose interarrival times are those of the "N-tuple" customers. The service time corresponding to the ρth arrival to this system is $x^\rho + V^{\rho+1}$. This service time clearly depends on $t^{\rho+1}$, and is independent of t^ρ. This system thus fits the assumptions of Lindley (1952) and the results follow directly from his. □

In the next proposition we state the stability condition for the Upper Bound System. This system corresponds to a queue whose interarrival times are again those of the "N-tuple" customers. The service time of the ρth arrival is $x^\rho + V^\rho$. This service time depends on t^ρ. A large t^ρ would probably imply that a large number of "ordinary" customers arrived during this interval, hence a large V^ρ. Although Lindley's results are thus inapplicable, Theorem 7 in Borovkov (1976, p. 14) covers this case.

PROPOSITION 3.3. *If $\beta + \gamma < \alpha$, then $\{\bar{W}^\rho = 0\}$ is positive recurrent.*

Proof. It is easy to check using (3.2)-(3.5) and (2.8) that

$$\bar{W}^{\rho+1} = [\bar{W}^\rho + x^\rho - t^{\rho+1}]^+ + V^{\rho+1}.$$

Alternately, defining $Y^\rho = \bar{W}^\rho - V^\rho$ we have

$$Y^{\rho+1} = [Y^\rho + x^\rho + V^\rho - t^{\rho+1}]^+.$$

Although the sequence of random variables $\{(x^\rho + V^\rho - t^{\rho+1}), \rho \geqq 1\}$ is not independent, it is identically distributed. In addition, it forms an irreducible Markov chain and is

hence ergodic. The process Y thus satisfies the requirements of Theorem 7 in Borovkov (1976, p. 14), and the proposition follows. □

We proceed now with the stability result concerning the original system.

THEOREM. *The event* $\{W^\rho = 0\}$ *is positive recurrent if and only if* $\beta + \gamma < \alpha$.

Proof. Assume $\beta + \gamma < \alpha$. By Proposition 3.3, $\{\bar{W}^\rho = 0\}$ is positive recurrent and since by Proposition 3.1, $W^\rho \leqq \bar{W}^\rho$ for $\rho \geqq 1$, it follows that $\{W^\rho = 0\}$ is also positive recurrent as well. If $\beta + \gamma \geqq \alpha$, since $\underline{W}^\rho \leqq W^\rho$ by Proposition 3.1, it follows from Proposition 3.2 that $\{W^\rho = 0\}$ cannot be positive recurrent. □

4. The stability condition. In this section we discuss the calculation of β in terms of the more basic data of the system. It is straightforward to obtain expressions that allow a numerical calculation of β; explicit expressions are obtained only in very special cases. Some asymptotic results can also be obtained.

Let $p_i(n, t)$ be the probability of having n ordinary customers arrive in station i between two "N-tuple" customers given that the time between their arrival is t, for $1 \leqq i \leqq N$ and $n \geqq 0$. Then

$$p_i(n, t) = e^{-\lambda_i t} \frac{(\lambda_i t)^n}{n!}. \tag{4.1}$$

Let B_i^{*n} denote the n-fold convolution of B_i, so that $B_i^{*n}(x) = P\{\sum_{j=1}^n \nu_i^j \leqq x\}$, $x \geqq 0$, $n \geqq 1$, $1 \leqq i \leqq N$. If we define

$$F_i(x, t) = P\{V_i^1 \leqq x | T = t\}, \qquad x \geqq 0, \quad 1 \leqq i \leqq N, \tag{4.2}$$

then

$$F_i(x, t) = p_i(0, t) + \sum_{n=1}^{\infty} p_i(n, t) B_i^{*n}(x). \tag{4.3}$$

Let

$$\begin{aligned} \hat{F}_N(x, t) &= P\{\max_{1 \leqq i \leqq N} \{V_i^1\} \leqq x | T = t\} \\ &= P\{V^1 \leqq x | T = t\}, \end{aligned} \tag{4.4}$$

so that

$$\hat{F}_N(x, t) = \prod_{i=1}^{N} F_i(x, t). \tag{4.5}$$

Finally, if we let

$$\hat{F}_N(x) = P\{V^1 \leqq x\},$$

then

$$\hat{F}_N(x) = \int_0^\infty \hat{F}_N(x, t)\, dG(t), \tag{4.6}$$

and in the symmetric case where $F_i = F$, $1 \leqq i \leqq N$,

$$\hat{F}_N(x, t) = [F(x, t)]^N,$$

and

$$\hat{F}_N(x) = \int_0^\infty [F(x, t)]^N\, dG(t). \tag{4.7}$$

Unfortunately, in general, an analytic expression for β using (4.6) or (4.7) is extremely hard to obtain. In what follows we will obtain an indication of the asymptotic growth rate of β (as a function of N). It is intuitively clear that β should grow with N since in (4.5) the maximum will be taken over a larger set.

To carry out the asymptotic analysis the following assumptions will be made. Arrival rates at all stations are equal i.e., $\lambda_i = \lambda$, $i = 1, \cdots, N$ and "ordinary" customers carry an exponential amount of work with parameter μ. Let $T = t^1$ denote the interarrival time of the first "N-tuple" customer, n be the number of "ordinary" customers that arrived during T at station i and P_n the probability of such an event. Then

$$
\begin{aligned}
P\{V_i^1 \leqq x | T\} &= \sum_{n=1}^{\infty} P\{V_i^1 \leqq x | T, n\} P_n + P_o \\
&= \sum_{n=1}^{\infty} \int_o^x \frac{\mu^n s^{n-1} e^{-\mu s}}{(n-1)!}\, ds \cdot e^{-\lambda T} \frac{(\lambda T)^n}{n!} + e^{-\lambda T} \\
&= \int_o^x \mu\lambda T \sum_{n=1}^{\infty} \frac{(\mu s \lambda T)^{n-1}}{(n-1)!\, n!}\, e^{-\lambda T} \cdot e^{-\mu s}\, ds + e^{-\lambda T}.
\end{aligned} \tag{4.8}
$$

Note that given T, the V_i^1, $i = 1, \cdots, N$, are independent. Instead of computing the mean of the maximum of N independent random variables distributed as in (4.8), we will derive their Characteristic Maximum $C_N(T)$. Then the Characteristic Maximum of N i.i.d. random variables with distribution function $F(x)$, is defined (see Gumbel (1958)) as the C_N such that

$$
\int_{C_N}^{\infty} dF(x) = \frac{1}{N},
$$

and it typically has the same asymptotic growth rate as the mean of the maximum of these random variables. Using (4.8), we have to find $C_N(T)$ that satisfies the equation

$$
\mu\lambda T e^{-\lambda T} \int_{C_N(T)}^{\infty} e^{-\mu s} \sum_{n=1}^{\infty} \frac{(\mu s \lambda T)^{n-1}}{(n-1)!\, n!}\, ds = \frac{1}{N}. \tag{4.9}
$$

By substituting $\mu s \lambda T = (y/2)^2$ and using the identity

$$
I_1(y) = \frac{y}{2} \sum_{n=0}^{\infty} \frac{(y/2)^{2n}}{n!(n+1)!}
$$

we obtain

$$
\frac{1}{N} = \lambda T e^{-\lambda T} \int_{\mu C_N(T)}^{\infty} e^{-s} \frac{I_1(2\sqrt{\lambda T s})}{\sqrt{\lambda T s}}\, ds. \tag{4.10}
$$

As $s \to \infty$,

$$
\frac{I_1(2\sqrt{\lambda T s})}{\sqrt{\lambda T s}} \simeq \frac{e^{2\sqrt{\lambda T s}}}{2\sqrt{\pi}(\lambda T s)^{3/4}}. \tag{4.11}
$$

Hence, since $\mu C_N(T)$ becomes large for large values of N, we substitute (4.11) into (4.10) and we find

$$
\frac{1}{N} \simeq \frac{\lambda T}{2\sqrt{\pi}} \int_{\mu C_N(T)}^{\infty} \frac{e^{-(\sqrt{s} - \sqrt{\lambda T})^2}}{(\lambda T s)^{3/4}}\, ds. \tag{4.12}
$$

By substituting $\sqrt{s} = \sqrt{w} + \sqrt{\lambda T}$ (4.12) becomes

$$
\frac{1}{N} \simeq \frac{(\lambda T)^{1/4}}{2\sqrt{\pi}} \int_{(\sqrt{\mu C_N(T)} - \sqrt{\lambda T})}^{\infty} e^{-w} \frac{dw}{\sqrt{w}\sqrt{\sqrt{w} + \sqrt{\lambda T}}}. \tag{4.13}
$$

By integrating (4.13) by parts and considering the dominant term of the resulting series (for $\mu C_N(T) \gg \lambda T$), we find

$$\frac{1}{N} \simeq \frac{(\lambda T)^{1/4}}{2\sqrt{\pi}} \frac{e^{-(\sqrt{\mu C_N(T)}-\sqrt{\lambda T})^2}}{(\sqrt{\mu C_N(T)}-\sqrt{\lambda T})((\mu C_N(T))^{1/4})}, \tag{4.14}$$

which is equivalent to

$$\sqrt{\mu C_N(T)} \simeq \sqrt{\lambda T} + \lambda \ln N + y \tag{4.15}$$

where y is of the order of $\ln(\sqrt{\ln N})/\sqrt{\ln N} \to 0$ for $N \to \infty$. Notice that for N large enough, the distribution function $G(x)$ of T assumes most of its mass for values of λT much smaller than $\mu C_N(T)$. Thus one can get the unconditional Characteristic Maximum $\bar{C}_N$ by averaging (4.15) over T. It is easy to see that

$$\bar{C}_N \simeq \frac{1+o(1)}{\mu} \ln N, \quad o(1) \to 0 \quad \text{as } N \to \infty, \tag{4.16}$$

which in fact indicates that β grows as $(1/\mu) \ln N$.

A last question we consider is how fast, as a function of N, does the input rate of "ordinary" work (λ) to the system need to decrease, so that C_N remains constant (independent of N). This roughly corresponds to keeping the waiting time of "N-tuple" customers constant. By equation (4.14) it follows that λ should decrease as $(1/N)^4$. This suggests that the capability of the system to handle work decreases rather rapidly with N.

Acknowledgments. We would like to thank Alan Weiss and Jim McKenna for many helpful discussions. We would also like to thank an anonymous referee for pointing out the applicability of Borovkov (1976) in Proposition 3.3.

REFERENCES

F. BACCELLI AND E. G. COFFMAN, JR. (1982), *A data base replication analysis using an M/M/m queue with service interruptions*, Performance Evaluation Review, 11, pp. 102-107.

A. A. BOROVKOV (1976), *Stochastic Processes in Queueing Theory*, Springer-Verlag, New York.

L. BREIMAN (1968), *Probability*, Addison-Wesley, New York.

E. G. COFFMAN, JR., H. O. POLLAK, E. GELENBE AND R. C. WOOD (1981), *An analysis of parallel-read sequential-write systems*, Performance Evaluation, 1, pp. 62-69.

L. GREEN (1980), *A queueing system in which customers require a random number of servers*, Oper. Res., 28, pp. 1335-1345.

E. J. GUMBEL (1958), *Statistics of Extremes*, Columbia Univ. Press, New York.

D. V. LINDLEY (1952), *The theory of queues with a single server*, Proc. Cambridge Philosophical Society, 48, 277-289.

M. YANNAKAKIS, C. H. PAPADIMITRIOU AND H. T. KUNG (1979), *Locking policies: Safety and freedom from deadlock*, Proc. of the 20th Annual Symp. on Foundations of Computer Science, pp. 286-297.

A. W. SHUM AND P. G. SPIRAKIS (1981), *Performance analysis of concurrency control methods in data base systems*, in Performance '81, F. J. Kylstra, ed., North-Holland, Amsterdam.

N. GOODMAN, R. SUIR AND T. C. TAY (1983), *A simple analytic model for performance of exclusive locking in database systems*, Proc. of ACM Symposium on Principles of Database Systems, Atlanta, pp. 203-215.

D. MITRA AND P. J. WEINBERGER (1984), *Some results on database locking: Solutions, computational algorithms and asymptotics*, in Mathematical Computer Performance and Reliability, G. Iazeolla, P. J. Courtois and A. Hordijk, eds., North-Holland, Amsterdam.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987

© 1987 Society for Industrial and Applied Mathematics
015

ON PROVING LIMITING COMPLETENESS*

PETER D. MOSSES† AND GORDON D. PLOTKIN‡

Abstract. We give two proofs of Wadsworth's classic approximation theorem for the pure λ-calculus. One of these illustrates a new method utilising a certain kind of intermediate semantics for proving correspondences between denotational and operational semantics. The other illustrates a direct technique of Milne, employing recursively-specified inclusive relations.

Key words. lambda calculus, semantics

1. Introduction. Suppose that we have both a (standard) denotational semantics and an operational semantics for some programming language. (For examples see [Sto1], [Mil2], [deB].) We would like to prove that they are *equivalent*, in that the output given by the operational semantics, for each program and input, corresponds exactly to the output specified by the denotational semantics. Thus not only is the former to be *consistent* with the latter, but also it is to be *complete*. For diverging computations, completeness is only required "in the limit": the (perhaps partial) outputs given by finite computations of the operational semantics are to converge to the full output specified by the denotational semantics. Following [Wad], we shall refer to this property as *limiting completeness*.

In [Wad], Wadsworth studied the λ-calculus, considering arbitrary λ-terms as programs (including their "input"). The "approximate normal forms" of terms, obtained by finite sequences of β-reductions (followed by the replacement of remaining β-redexes by a special constant symbol Ω that denoted ⊥), were regarded as outputs. The classic Approximation Theorem [Wad, Thm. 5.2] shows that the denotations of the approximate normal forms of a term do indeed converge to the denotation of the term itself. Thus the theory of β-conversion, although too weak for proving all equations between terms whose denotations are equal, is complete in this limiting sense, and adequate for the evaluation of λ-terms.

In general, it is quite easy to prove consistency, using structural induction. See [Sto1], [deB] for some examples. We shall not consider consistency proofs any further in this paper. Sometimes, it is also possible to prove limiting completeness quite directly using structural induction with subsidiary appeal to fixpoint induction for recursive constructs such as loops (where structural induction fails). However, when the programming languages concerned allow self-application—either explicitly as with Algol 60 procedures, or implicitly as with dynamic bindings in LISP—then the direct method seems to be precluded, for structural induction fails but there is no obvious recursive construct in the language where fixed-point methods could be applied. In such cases, the domains of denotations are defined reflexively (that is, recursively), and what one wants is to use induction on the level of the projective approximations of the domains.

Wadsworth solved this problem in his study of the λ-calculus [Wad] by labelling expressions M (and their subexpressions) with integers n, so that $M^{(n)}$ denoted the nth projection of the denotation of M. Having introduced some extra syntax to make

* Received by the editors December 26, 1984, and in revised form June 3, 1985. This work was supported in part by the Science and Engineering Research Council.

† Computer Science Department, Aarhus University, DK-8000 Aarhus C, Denmark.

‡ Department of Computer Science, University of Edinburgh, Edinburgh, Scotland, EH9 3JZ. This author benefited from a Visiting Research Fellowship.

the levels visible, he then studied the operational properties of the $M^{(n)}$ induced by this semantics and also their relation to the operational properties of the original M. Thus one parameter—the labelling—was used both for inductions relating to the denotational semantics (the projection levels) and for inductions relating to the operational semantics. It seems fair to say that as a result a somewhat heavy apparatus was obtained.

We present two other approaches to the proof of (the difficult part of) the Approximation Theorem. In one the two inductions are separated and the $M^{(n)}$ avoided by means of an intermediate semantics. In the other, using ideas of Milne [Mil1], [Mil2], [Sto1], [Sto2], the second induction is avoided entirely. In both approaches recursively-specified inclusive relations are used [Mil1]; these play only a technical role in the first one but lie at the heart of the second.

Our intermediate semantics is defined just like the standard denotational semantics, except that intermediate denotations take an argument k in $\mathbb{1}^{\infty}$ (the chain cpo of the extended natural numbers $0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq k \sqsubseteq \cdots \sqsubseteq \infty$). The operational idea is that at finite values of k, an intermediate denotation is to correspond to (perhaps partial) output produced by the operational semantics after at most k steps. In particular, at $k = 0$ it is the least element $\perp$. At $k = \infty$ it is by continuity the limit of the intermediate denotations.

The denotational idea is (roughly) that the intermediate denotation at ∞ of any phrase is just a function of the intermediate denotations at ∞ of its sub-phrases. This makes it possible to see the relation between the intermediate and standard denotations. To relate them formally, one defines a recursively-specified inclusive relation between the standard and the intermediate domains. The existence and properties of this relation are established by induction on the projective levels, and in fact the techniques are, for the most part, well known (see [Sto1], [Mul] for example). Then a simple structural induction establishes the relation between the standard and intermediate semantics.

The idea of the second approach is to strengthen the hypothesis so that structural induction succeeds. To do this, one defines (for every environment) a relation between semantic values and terms of the language which holds when the value is less than the limiting value of the term given via its operational semantics and moreover when in all suitable contexts the relation still holds. As explained by Stoy [Sto2] one can think of the relation as being that the value approximates to the term. The suitable contexts are given by other related pairs of values and terms and are determined by examining where the original proof by Structural Induction failed. As usual, the existence of such recursively specified relations is determined by induction on the projective levels and the desired result is shown by establishing by structural induction that the denotation of a term is related to the term itself.

Comparing the two approaches, we see that the second is the more direct while the first provides more information, via the intermediate semantics, on the details of the operational semantics. In that connection we should also remark that Wadsworth's labelled expressions have been very useful in the study of the λ-calculus [Bar]. It may be that the intermediate semantics with its feeling of explaining resource-bounded denotations will also be of use. Technically, as the reader will see, the proofs are less straightforward with the second approach, at least in the present case. The references demonstrate the wide applicability of the second approach; we expect this also for the intermediate semantics method although we have no precise general recipe available for defining such semantics.

We assume the general framework of denotational semantics [Ten], [Sto1], [Mil2], [Gor]. We take domains to be ω-complete partial orders (which are just partial orders

$\mathbf{D} = (D, \sqsubseteq)$ with a least element $\perp_D$, and lubs $\sqcup_n x_n$ of increasing sequences). The important functions between domains are those $f: \mathbf{D} \to \mathbf{E}$ which are *continuous*, meaning that if x_n is an increasing sequence in $\mathbf{D}$ then $f(x_n)$ is in $\mathbf{E}$ and then, too, $f(\sqcup_n x_n) = \sqcup_n f(x_n)$; we write $\mathbf{D} \to \mathbf{E}$ for the cpo of all continuous functions from $\mathbf{D}$ to $\mathbf{E}$, with the pointwise ordering $f \sqsubseteq g$ iff for all in $\mathbf{D}$, $f(x) \sqsubseteq g(x)$. See [SP] for further information on this topic, especially as regards constructions of domains including the solution of recursive domain equations. Our notation is essentially that of [Wad], to facilitate comparison of results, and that of [SP] to handle cpos.

2. The intermediate semantics approach. We shall now prove the completeness of an operational semantics for the λ-calculus relative to its usual denotational semantics, by introducing an appropriate intermediate semantics.

First let us recall the syntax of the λ-calculus. We assume a denumerable set **Var** of variables. Basically, the set **Exp** of *λ-terms* is taken to be the least such that:

- if x is in **Var** then x is in **Exp**;
- if x is in **Var** and M is in **Exp** then $(\lambda x.\, M)$ is in **Exp**;
- if M and N are in **Exp** then (MN) is in **Exp**.

However, for convenience, we extend **Exp** to include *partial terms* by means of a clause for the special symbol Ω:

- Ω is in **Exp**.

We may now regard **Exp** as a poset, taking the least partial order $\sqsubseteq$ on **Exp** such that:

- $\Omega \sqsubseteq M$, for all M in **Exp**;
- if $M \sqsubseteq M'$ then $(\lambda x.\, M) \sqsubseteq (\lambda x.\, M')$ for all M and M' in **Exp** and x in **Var**;
- if $M \sqsubseteq M'$ and $N \sqsubseteq N'$ then $(MN) \sqsubseteq (M'N')$ for all M, M', N, N' in **Exp**.

Note that $(\mathbf{Exp}, \sqsubseteq)$ is *not* a cpo. However, we shall ensure that all functions that we define on **Exp** are monotonic.

The standard denotational semantics of the λ-calculus is given by the (monotonic) function $\mathscr{V}: \mathbf{Exp} \to (\mathbf{Env} \to \mathbf{D})$ specified in Table 1. Here $\mathbf{D}$ is taken to be the (initial) solution of $\mathbf{D} \cong \mathbf{D} \to \mathbf{D}$ that includes some arbitrary nontrivial cpo $\mathbf{D}_0$ ([SP] (see also our Appendix)). (This is a slight generalisation of Scott's original $\mathbf{D}_\infty$-model [Sco], [Wad], where only complete lattices were considered.) We omit the isomorphism $\mathbf{D} \cong \mathbf{D} \to \mathbf{D}$ from formulae, when there is no danger of confusion.

TABLE 1
Standard semantics.

Domains	Denotations
$\mathbf{D} \cong \mathbf{D} \to \mathbf{D}$ (see text) $\mathbf{Env} = \mathbf{D}^{\mathbf{Var}}$	$\mathscr{V}: \mathbf{Exp} \to \mathbf{Env} \to \mathbf{D}$ $\mathscr{V}[\![x]\!]\rho = \rho(x)$ $\mathscr{V}[\![\lambda x.\, M]\!]\rho = \lambda d \in \mathbf{D}.\, \mathscr{V}[\![M]\!](\rho[d/x])$ $\mathscr{V}[\![MN]\!]\rho = (\mathscr{V}[\![M]\!]\rho)(\mathscr{V}[\![N]\!]\rho)$ $\mathscr{V}[\![\Omega]\!]\rho = \perp_{\mathbf{D}}$

Our operational semantics for the λ-calculus is given by the ω-indexed family of (monotonic) functions $\mathscr{R}^n: \mathbf{Exp} \to \mathbf{Exp}$ defined in Table 2 for any n and any M in **Exp**; $\mathscr{R}^n(M)$ may be regarded as the *partial* normal form of M determined by n steps of a (parallel) reduction algorithm. That is, $\mathscr{R}^n(M)$ can be obtained by making some β-reductions on M, followed by replacing all remaining β-redexes by Ω. Thus each $\mathscr{R}^n(M)$ is an approximate normal form of M in the terminology of [Wad]. Moreover,

TABLE 2
Operational semantics.

$\mathscr{R}^n$: **Exp** → **Exp** (each $n \in \omega$)

$\mathscr{R}^0(M) = \Omega$

$\mathscr{R}^{n+1}(x) = x$

$\mathscr{R}^{n+1}(\lambda x.\, M) = \lambda x.\, M'$ where $M' = \mathscr{R}^{n+1}(M)$

$$\mathscr{R}^{n+1}(MN) = \begin{cases} \Omega \text{ if } M' = \Omega, \\ \mathscr{R}^n([N'/x]M''), & \text{if } M' = \lambda x.\, M'', \\ (M'N') \quad \text{otherwise,} \end{cases}$$
where $M' = \mathscr{R}^{n+1}(M)$, $N' = \mathscr{R}^{n+1}(N)$

$\mathscr{R}^{n+1}(\Omega) = \Omega$

($[-/x]-$: **Exp** × **Exp** → **Exp** is the usual substitution operator)

$\mathscr{R}^n(M)$ is monotonic in n: $\mathscr{R}^n(M) \sqsubseteq \mathscr{R}^{n+1}(M)$, for all n, as can be shown by a simple induction on (n, M) (lexicographically ordered).

The completeness of this operational semantics (relative to the given denotation semantics) is just that for any term M, its denotation $\mathscr{V}[\![M]\!]$ is included in the limit of the denotations $\mathscr{V}[\![\mathscr{R}^n(M)]\!]$, as n goes to ∞—this limit exists since $\mathscr{V}$ is monotonic (and **Env** → **D** is a cpo).

THEOREM 1. *For M in* **Exp**

$$\mathscr{V}[\![M]\!] \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}[\![\mathscr{R}^n(M)]\!].$$

An immediate corollary of this theorem is that for all M,

$$\mathscr{V}[\![M]\!] \sqsubseteq \bigsqcup \{\mathscr{V}[\![A]\!] \mid A \text{ is an approximate normal form of } M\},$$

which is the hard part of Wadworth's classic Approximation Theorem for the λ-calculus [12]. (The reverse directions of both these theorems follow easily from the consistency of β-reduction and the minimality of Ω, with respect to $\mathscr{V}$. We shall not consider consistency any further in this paper.)

To prove our theorem, we introduce an intermediate semantics $\mathscr{V}'$: **Exp** → $\mathbb{1}^\infty$ → **Env'** → **D'**, as given in Table 3 ($\mathscr{V}'$: **Exp** → **Env'** → $\mathbb{1}^\infty$ → **D'** might be considered more natural, but leads to clumsier statements of theorems). Recall that $\mathbb{1}^\infty$ is the chain cpo of the extended natural numbers. There is an evident embedding of ω in $\mathbb{1}^\infty$, such that the usual operations of successor, predecessor and minimum on ω have unique continuous extensions to $\mathbb{1}^\infty$—we get $\infty = \infty + 1 = \infty - 1$ and $k \min \infty = k$. The domain

TABLE 3
Intermediate semantics.

Domains	Denotations
$\mathbf{D}' = (\mathbb{1}^\infty \to \mathbf{D}') \to \mathbf{D}'$ (see text) $\mathbf{Env}' = (\mathbb{1}^\infty \to \mathbf{D}')^{\mathbf{Var}}$	$\mathscr{V}'$: **Exp** → $\mathbb{1}^\infty$ → **Env'** → **D'** $\mathscr{V}'[\![x]\!]k\rho' = \begin{cases} \bot_{\mathbf{D}'} & \text{if } k = 0, \\ \rho'(x)k & \text{otherwise,} \end{cases}$ $\mathscr{V}'[\![\lambda x.\, M]\!] \in k\rho' = \begin{cases} \bot_{\mathbf{D}} & \text{if } k = 0, \\ \lambda c' \mathbb{1}^\infty \to \mathbf{D}'.\, \mathscr{V}'[\![M]\!]k(\rho'[c'/x]) & \text{otherwise} \end{cases}$ $\mathscr{V}'[\![MN]\!]k\rho' = \begin{cases} \bot_{\mathbf{D}'} & \text{if } k = 0, \\ (\mathscr{V}'[\![M]\!](k-1)\rho')(\lambda k' \in \mathbb{1}^\infty.\, \mathscr{V}'[\![N]\!](k' \min k)\rho') & \text{otherwise} \end{cases}$ $\mathscr{V}'[\![\Omega]\!]k\rho' = \bot_{\mathbf{D}'}$

$\mathbf{D}'$ is taken to be the (initial) solution of $\mathbf{D}' \simeq (\mathbb{1}^\infty \to \mathbf{D}') \to \mathbf{D}'$ that includes the same nontrivial domain $\mathbf{D}_0$ that we included earlier in $\mathbf{D}$. Again we often omit the isomorphism. As it happens, $\mathscr{V}'$ does not respect β-reduction: in general only $\mathscr{V}'[\![(\lambda^x M)N]\!] \sqsubseteq \mathscr{V}'[\![[N/x]M]\!]$ holds—but that is not important here.

Our first lemma relates $\mathscr{R}^n(M)$ to $\mathscr{V}'$ at n.

LEMMA 1. *For all $n < \infty$ and M in* **Exp**,

$$\mathscr{V}'[\![M]\!]n \sqsubseteq \mathscr{V}'[\![\mathscr{R}^n(M)]\!]n.$$

We shall prove this lemma by induction on (n, M), later in this section. By the continuity of $\mathscr{V}'[\![M]\!]n$ in n, we then get

$$\mathscr{V}'[\![M]\!]\infty \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}'[\![\mathscr{R}^n(M)]\!]\infty. \tag{1}$$

Now putting ∞ for k in Table 3, it seems quite obvious that $\mathscr{V}[\![M]\!]$ is equivalent to $\mathscr{V}'[\![M]\!]\infty$, and that we should be able to infer the required result, namely,

$$\mathscr{V}[\![M]\!] \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}[\![\mathscr{R}^n(M)]\!]. \tag{2}$$

However, to prove this "obvious" equivalence between $\mathscr{V}[\![M]\!]$ and $\mathscr{V}'[\![M]\!]\infty$, we need to relate values of $\mathbf{D}$ and $\mathbf{D}'$. As we have already remarked, the techniques for this are quite standard but they appeal to the construction of solutions to reflexive domain equations, and we relegate the details to Appendix I. It would be routine to relate any reflexively-defined domain to an analogous one involving $\mathbb{1}^\infty$, in essentially the same way. (Thus the proofs of the remaining lemmas except perhaps that of Lemma 4 should not really be considered when assessing the complexity of our approach.)

LEMMA 2. *There exists a relation $\sim \subseteq \mathbf{D} \times \mathbf{D}'$ such that*

(i) *for all d in $\mathbf{D}_0$, $d \sim d$ under the inclusion of $\mathbf{D}_0$ in $\mathbf{D}$ and $\mathbf{D}'$;*

(ii) *for all d in $\mathbf{D}$ and d' in $\mathbf{D}'$, $d \sim d'$ iff for all e in $\mathbf{D}$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'$, $e \sim c'(\infty)$ implies $d(e) \sim d'(c')$.*

The proof of Lemma 2 may be found in the Appendix.

The next lemma states that $\mathscr{V}[\![M]\!]$ and $\mathscr{V}'[\![M]\!]\infty$ are related by the relation provided by Lemma 2.

LEMMA 3. *For all M in* **Exp** *and ρ in* **Env** *and ρ' in* **Env**′, *if for all z in* **Var**, $\rho(z) \sim \rho'(z)\infty$ *then* $\mathscr{V}[\![M]\!]\rho \sim \mathscr{V}'[\![M]\!]\infty\rho'$.

We shall prove this lemma by induction on M, later in this section.

According to the next lemma, if $d \sim d'$ then d is (continuously) determined by d'. This will allow us to infer (2) from (1) above.

LEMMA 4. *There exist continuous functions $s : \mathbf{D} \to \mathbf{D}'$ and $r : \mathbf{D}' \to \mathbf{D}$ such that*

(i) *for all d in $\mathbf{D}$, $d \sim s(d)$;*

(ii) *for all d in $\mathbf{D}$ and d' in $\mathbf{D}'$, if $d \sim d'$ then $d = r(d')$.*

The proof of Lemma 4 may be found in the Appendix.

We now show how our theorem follows from Lemmas 1, 3 and 4.

Proof of Theorem 1. Let M in **Exp** and ρ in **Env** be arbitrary. The required result will follow if we show that

$$\mathscr{V}[\![M]\!]\rho \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}[\![\mathscr{R}^n(M)]\!]\rho.$$

Define ρ' in **Env**′ by

$$\rho'(x) = \lambda k \in \mathbb{1}^\infty . s(\rho(x))$$

where $s : \mathbf{D} \to \mathbf{D}'$ is as in Lemma 4(i), so that for all z in **Var**,

$$\rho(z) \sim s(\rho(z)) = \rho'(z)\infty.$$

Now by Lemma 1, for all n in $\omega(\subseteq \mathbb{1}^\infty)$ we have

$$\mathcal{V}'[\![M]\!]n\rho' \sqsubseteq \mathcal{V}'[\![\mathcal{R}^n(M)]\!]n\rho'$$

which by continuity in n and $\bigsqcup_{n\geq 0} n = \infty$ (in $\mathbb{1}^\infty$) gives

$$\mathcal{V}'[\![M]\!]\infty\rho' \sqsubseteq \bigsqcup_{n\geq 0} \mathcal{V}'[\![\mathcal{R}^n(M)]\!]\infty\rho'. \tag{3}$$

Lemma 3 gives

$$\mathcal{V}[\![M]\!]\rho \sim \mathcal{V}'[\![M]\!]\infty\rho' \tag{4}$$

and that for all n,

$$\mathcal{V}[\![\mathcal{R}^n(M)]\!]\rho \sim \mathcal{V}'[\![\mathcal{R}^n(M)]\!]\infty\rho'. \tag{5}$$

Let $r: \mathbf{D}' \to \mathbf{D}$ be as in Lemma 4(ii). Then we have

$$\begin{aligned}
\mathcal{V}[\![M]\!]\rho &= r(\mathcal{V}'[\![M]\!]\infty\rho') \quad \text{from (4)} \\
&\sqsubseteq r\Big(\bigsqcup_{n\geq 0} \mathcal{V}'[\![\mathcal{R}^n(M)]\!]\infty\rho'\Big) \quad \text{by (3)} \\
&= \bigsqcup_{n\geq 0} r(\mathcal{V}'[\![\mathcal{R}^n(M)]\!]\infty\rho') \quad \text{by continuity} \\
&= \bigsqcup_{n\geq 0} \mathcal{V}[\![\mathcal{R}^n(M)]\!]\rho \quad \text{from (5)},
\end{aligned}$$

as required.

We now return to our lemmata. The proof of Lemma 1 will make use of the following standard lemma about substitution, which confirms that the operator $[-/x]-$ has been defined correctly so as to respect the static determination of bindings.

SUBSTITUTION LEMMA. *For all M and N in* **Exp** *and ρ' in* **Env**′ *and k in $\mathbb{1}^\infty$,*

$$\mathcal{V}'[\![[N/x]M]\!]k\rho' = \mathcal{V}'[\![M]\!]k(\rho'[(\lambda k' \in \mathbb{1}^\infty .\ \mathcal{V}'[\![N]\!]k'\rho')/x]).$$

We shall omit the straightforward but tedious proof by induction (on M), which is entirely analogous to the usual substitution lemma for the standard semantics, $\mathcal{V}$.

Proof of Lemma 1. We are to show that for all $n < \infty$ and M in **Exp**,

$$\mathcal{V}'[\![M]\!]n \sqsubseteq \mathcal{V}'[\![\mathcal{R}^n(M)]\!]n.$$

We use induction on (n, M). For $n = 0$ and any M, we have $\mathcal{V}'[\![M]\!]0 = \bot \sqsubseteq \mathcal{V}'[\![\mathcal{R}^0(M)]\!]0$, as required.

For $n+1$ and any M_0, let v and w abbreviate $\mathcal{V}'[\![M_0]\!](n+1)$ and $\mathcal{V}'[\![\mathcal{R}^{n+1}(M_0)]\!](n+1)$, respectively. We shall show that $v \sqsubseteq w$, as required, by cases on M_0.

For Ω, we have $v = \bot$, hence $v \sqsubseteq w$.

For x, we have $\mathcal{R}^{n+1}(x) = x$, giving $v = w$.

For $(\lambda x.\ M)$, we have $\mathcal{R}^{n+1}(\lambda x.\ M) = (\lambda x.\ M')$, where $M' = \mathcal{R}^{n+1}(M)$. By the induction hypothesis for $(n+1, M)$ we have $\mathcal{V}'[\![M]\!](n+1) \sqsubseteq \mathcal{V}'[\![M']\!](n+1)$, hence $v \sqsubseteq w$ (by monotonicity).

Finally, for (MN), take ρ' in **Env**′. Then

$$v\rho' = (\mathcal{V}'[\![M]\!]n\rho')(\lambda k' \in \mathbb{1}^\infty .\ \mathcal{V}'[\![N]\!](k' \min n+1)\rho').$$

Let $M' = \mathcal{R}^{n+1}(M)$ and $N' = \mathcal{R}^{n+1}(N)$. By the induction hypothesis, we have

$$\begin{aligned}
\mathcal{V}'[\![M]\!]n &\sqsubseteq \mathcal{V}'[\![\mathcal{R}^n(M)]\!]n \\
&\sqsubseteq \mathcal{V}'[\![M']\!]n \quad \text{(by monotonicity)};
\end{aligned}$$

also for any $k' \in \mathbb{1}^\infty$, we have $k' \min n+1 \leqq n+1$ so that

$$\mathscr{V}'[\![N]\!](k' \min n+1) \sqsubseteq \mathscr{V}'[\![\mathscr{R}^{k' \min n+1}(N)]\!](k' \min n+1)$$
$$\sqsubseteq \mathscr{V}'[\![N']\!](k' \min n+1) \quad \text{(by monotonicity).}$$

Putting these calculations together, we get

$$v\rho' \sqsubseteq (\mathscr{V}'[\![M']\!]n\rho')(\lambda k' \in \mathbb{1}^\infty . \mathscr{V}'[\![N']\!](k' \min n+1)\rho'). \tag{6}$$

We shall now consider the cases for M', but note first that when $n=0$, we have $\mathscr{V}'[\![M']\!]0\rho' = \perp$, so that (6) gives $v\rho' \sqsubseteq \perp \sqsubseteq w\rho'$, and hence $v \sqsubseteq w$ as required. We may now suppose $n>0$.

For $M' = \Omega$, we have $\mathscr{V}'[\![\Omega]\!]n\rho' = \perp$, and then (6) again gives $v\rho' = \perp$ and $v \sqsubseteq w$ as required.

For $M' = \lambda x . M''$, we have

$$\begin{aligned} v\rho' &\sqsubseteq (\mathscr{V}'[\![M']\!]n\rho')(\lambda k' \in \mathbb{1}^\infty . \mathscr{V}'[\![N']\!]k'\rho') \quad \text{by } k' \min n+1 \sqsubseteq k' \\ &= \mathscr{V}'[\![M'']\!]n(\rho'[(\lambda k' \in \mathbb{1}^\infty . \mathscr{V}'[\![N']\!]k'\rho')/x]) \\ &= \mathscr{V}'[\![[N'/x]M'']\!]n\rho' \quad \text{by the substitution lemma} \\ &\sqsubseteq \mathscr{V}'[\![\mathscr{R}^n([N'/x]M'')]\!]n\rho' \quad \text{by the induction hypothesis} \\ &= \mathscr{V}'[\![\mathscr{R}^{n+1}(MN)]\!]n\rho' \\ &\sqsubseteq \mathscr{V}'[\![\mathscr{R}^{n+1}(MN)]\!](n+1)\rho' = w\rho' \end{aligned}$$

giving $v \sqsubseteq w$, as required.

In the remaining cases for M' (viz. x and $(M''N'')$) we have $\mathscr{R}^{n+1}(MN) = (M'N')$. Hence

$$w\rho' = (\mathscr{V}'[\![M']\!]n\rho')(\lambda k' \in \mathbb{1}^\infty . \mathscr{V}'[\![N']\!](k' \min n+1)\rho')$$

so $v\rho' \sqsubseteq w\rho'$ by (6), giving $v \sqsubseteq w$ as required.

This exhausts the cases for M', thus completing the final case (MN) for M_0. Having now completed the induction step to $n+1$, we may infer the required result. □

We finish this section with a proof of Lemma 3. Its simplicity justifies our earlier remarks about the obviousness of the equivalence of $\mathscr{V}[\![M]\!]$ and $\mathscr{V}'[\![M]\!]\infty$.

Proof of Lemma 3. We use (structural) induction on M. Our induction hypothesis is that for all components M of M_0 in **Exp**,

(7) for all ρ in **Env** and ρ' in **Env'**, if for all z in **Var**, $\rho(z) \sim \rho'(z)\infty$ then $\mathscr{V}[\![M]\!]\rho \sim \mathscr{V}'[\![M]\!]\infty\rho'$.

We shall show that (7) then holds also for M_0, by cases on M_0.

For Ω, we have $\mathscr{V}[\![\Omega]\!]\rho = \perp_{\mathbf{D}} = \perp_{\mathbf{D}_0} = \perp_{\mathbf{D}'} = \mathscr{V}'[\![\Omega]\!]\infty\rho'$, for any ρ and ρ' (considering the inclusion of $\mathbf{D}_0$ in $\mathbf{D}$ and $\mathbf{D}'$). But by Lemma 2(i), $\perp_{\mathbf{D}} \sim \perp_{\mathbf{D}'}$. Hence (7) holds for Ω.

For x, we have $\mathscr{V}[\![x]\!]\rho = \rho(x)$ and $\mathscr{V}'[\![x]\!]\infty\rho' = \rho'(x)\infty$, for any ρ and ρ'. So (7) holds for x.

For $(\lambda x . M)$, take ρ and ρ' such that for all z in **Var**, $\rho(z) \sim \rho'(z)\infty$. Also take e in $\mathbf{D}$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'$ such that $e \sim c'(\infty)$. We have

$$\begin{aligned} (\mathscr{V}[\![\lambda x . M]\!]\rho)(e) &= \mathscr{V}[\![M]\!](\rho[e/x]) \\ &\sim \mathscr{V}'[\![M]\!]\infty(\rho'[c'/x]) \quad \text{by the induction hypothesis} \\ &= (\mathscr{V}'[\![\lambda x . M]\!]\infty\rho')(c') \end{aligned}$$

so by Lemma 2(ii) we get $\mathscr{V}[\![\lambda x . M]\!]\rho \sim \mathscr{V}'[\![\lambda x . M]\!]\infty\rho'$, showing that (7) holds for $(\lambda x . M)$.

Finally, for (MN), take ρ and ρ' such that for all z, $\rho(z) \sim \rho'(z)\infty$. Now by the induction hypothesis,

$$\mathscr{V}[\![M]\!]\rho \sim \mathscr{V}'[\![M]\!]\infty\rho' \quad \text{and}$$
$$\mathscr{V}[\![N]\!]\rho \sim \mathscr{V}'[\![N]\!]\infty\rho'$$
$$= (\lambda k' \in \mathbb{I}^\infty . \mathscr{V}'[\![N]\!](k' \min \infty)\rho')(\infty)$$

so that

$$\mathscr{V}[\![MN]\!]\rho = (\mathscr{V}[\![M]\!]\rho)(\mathscr{V}[\![N]\!]\rho)$$
$$\sim (\mathscr{V}'[\![M]\!]\infty\rho')(\lambda k' \in \mathbb{I}^\infty . \mathscr{V}'[\![N]\!](k' \min \infty)\rho')$$
$$= \mathscr{V}'[\![MN]\!]\infty\rho'$$

so (7) holds in this case also.

There are no more cases, so we may infer the required result. □

3. The relational approach. Here we use a different operational semantics $\mathscr{S}^n : \mathbf{Exp} \to \mathbf{Exp}$, (given in Table 4) which is "outside-in" whereas the previous one was "inside-out" (and this seems to be needed to make the relational approach work). Again each $\mathscr{S}^n(M)$ is an approximate form of M, monotonic in n. And we will show:

THEOREM 2. *For all M in* **Exp**

$$\mathscr{V}[\![M]\!] \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}[\![\mathscr{S}^n(M)]\!]$$

thereby providing our second proof of Wadsworth's theorem.

If one tries to prove this directly by structural induction on M, the proof breaks down in the case where M is an application. Consequently, following the idea explained above for every ρ in **Env** we wish to define a relation $\sim_\rho$ between **D** and **Exp** so that:

$$d \sim_\rho M \quad \text{iff} \quad d \sqsubseteq \bigsqcup_{n \geq 0} \mathscr{V}[\![\mathscr{S}^n(M)]\!]\rho \quad \text{and}$$
$$\forall e \in \mathbf{D}, N \in \mathbf{Exp} . e \sim_\rho N \supset d(e) \sim_\rho (MN).$$

These relations are constructed in Appendix II where we also demonstrate the very useful:

LEMMA 5. *Suppose d in* **D** *and M in* **Exp** *are such that whenever* $e_i \sim_\rho N_i$ *for* $i = 1, n'$ *it is the case that*

$$d(e_1) \cdots (e_{n'}) \sqsubseteq \bigsqcup_n \mathscr{V}[\![\mathscr{S}^n(MN_1 \cdots N_{n'})]\!]\rho.$$

Then it follows that $d \sim_\rho M$.

Note that it follows that $\perp \sim_\rho M$ always holds.

Now to demonstrate Theorem 2, it is clearly enough to show that $\mathscr{V}[\![M]\!](\rho) \sim_\rho M$ always holds. This is now done by proving a stronger statement (to handle free variables) by structural induction on M.

TABLE 4
Another operational semantics.

$\mathscr{S}^n$: **Exp** → **Exp** (each $n \in \omega$)
$\mathscr{S}^0(M) = \Omega$
$\mathscr{S}^{n+1}(\Omega N_1 \cdots N_{n'}) = \Omega$
$\mathscr{S}^{n+1}(xN_1 \cdots N_{n'}) = x\mathscr{S}^n(N_1) \cdots \mathscr{S}^n(N_{n'})$
$$\mathscr{S}^{n+1}((\lambda x . M)N_1 \cdots N_{n'}) = \begin{cases} \lambda x \mathscr{S}^n(m) & \text{if } n' = 0, \\ \mathscr{S}^n(([N_1/x]M)N_2 \cdots N_{n'}) & \text{if } n' \neq 0 \end{cases}$$

LEMMA 6. *Let M be an expression and ρ be an environment. Suppose that $a_j \sim_\rho A_j$ for $j = 1, m$. Then*

$$\mathscr{V}[\![M]\!]\rho[a_1/x_1, \cdots, a_m/x_m] \sim_\rho [A_1/x_1, \cdots, A_m/x_m]M$$

holds, where the x_i are distinct variables.

Proof. By structural induction on M. We will write $\tilde{\rho}$ for $\rho[a_1/x_1, \cdots, a_m/x_m]$ and $\tilde{K}$ for $[A_1/x_1, \cdots, A_m/x_m]K$ (for any term K), where the a_j, A_j, x_j will be understood from the context.

Case I. M is an application, (M_1M_2).

By induction hypothesis we have $\mathscr{V}[\![M_i]\!](\tilde{\rho}) \sim_\rho \tilde{M}_i$ $(i = 1, 2)$.

So $\mathscr{V}[\![M_1]\!](\tilde{\rho})(\mathscr{V}[\![M_2]\!](\tilde{\rho})) \sim_\rho (\tilde{M}_1\tilde{M}_2)$. But as the left-hand side is $\mathscr{V}[\![M_1M_2]\!](\tilde{\rho})$ and the right-hand side $\widetilde{(M_1M_2)}$ so we are finished.

Case II. M is Ω. This is by the remark after Lemma 5.

Case III. M is a variable, x:

Subcase 1. x is some x_j.

Then $\mathscr{V}[\![x]\!]\tilde{\rho} = a_j \sim_\rho A_j = \tilde{x}$.

Subcase 2. x is no x_j. Then we must show that $\mathscr{V}[\![x]\!]\rho \sim_\rho x$ and we employ Lemma 5, taking $e_i \sim_\rho N_i$ $(i = 1, n')$ and calculating:

$$\begin{aligned}
&\mathscr{V}[\![x]\!](\rho)(e_1)\cdots(e_{n'}) \\
&\quad \sqsubseteq \mathscr{V}[\![x]\!](\rho)\Big(\bigsqcup_n \mathscr{V}[\![\mathscr{S}^n N_1]\!](\rho)\Big)\cdots\Big(\bigsqcup_n \mathscr{V}[\![\mathscr{S}^n N_{n'}]\!](\rho)\Big) \qquad \text{(since } e_i \sim_\rho N_i\text{)} \\
&\quad = \bigsqcup_n \mathscr{V}[\![x]\!](\rho)(\mathscr{V}[\![\mathscr{S}^n N_1]\!](\rho))\cdots(\mathscr{V}[\![\mathscr{S}^n N_{n'}]\!](\rho)) \\
&\quad = \bigsqcup_n \mathscr{V}[\![x(\mathscr{S}^n N_1)\cdots(\mathscr{S}^n N_{n'})]\!](\rho) \\
&\quad = \bigsqcup_n \mathscr{V}[\![\mathscr{S}^{n+1}(xN_1\cdots N_{n'})]\!](\rho)
\end{aligned}$$

as required.

Case IV. M is an abstraction, $\lambda x.\, M'$. We may assume without loss of generality that x is no x_j and x does not occur free in any of the A_j.

We apply Lemma 5 and take $e_i \sim_\rho N_i$ $(i = 1, n')$.

Subcase 1. $n' = 0$. We calculate

$$\begin{aligned}
\mathscr{V}[\![\lambda x.\, M']\!]\tilde{\rho} &= \lambda d \in \mathbf{D}.\, \mathscr{V}[\![M']\!]\widetilde{\rho[d/x]} \quad \text{(as } x \text{ is no } x_j\text{)} \\
&\sqsubseteq \lambda d \in \mathbf{D}.\, \bigsqcup_n \mathscr{V}[\![\mathscr{S}^n\tilde{M}']\!]\rho[d/x] \\
&\qquad \text{(by the induction hypothesis applied to } M'\text{)} \\
&= \bigsqcup_n \lambda d \in \mathbf{D}.\, \mathscr{V}[\![\mathscr{S}^n\tilde{M}']\!](\rho[d/x]) \\
&= \bigsqcup_n \mathscr{V}[\![\lambda x.\, \mathscr{S}^n\tilde{M}']\!](\rho) \\
&= \bigsqcup_n \mathscr{V}[\![\mathscr{S}^{n+1}(\lambda x.\, \tilde{M}')]\!](\rho) \\
&= \bigsqcup_n \mathscr{V}[\![\mathscr{S}^{n+1}(\widetilde{\lambda x.\, M'})]\!](\rho)
\end{aligned}$$

as required.

Subcase 2. $n'>0$. First calculate that:

$$\begin{aligned}
&\mathscr{V}[\![\lambda x.\, M']\!](\tilde{\rho})(e_1)\\
&\quad = \mathscr{V}[\![M']\!]\rho[e_1/x, a_1/x_1, \cdots, a_m/x_m] \quad (\text{as } x \text{ is no } x_j)\\
&\quad \sim_\rho [N_1/x, A_1/x_1, \cdots, A_m/x_m]M'\\
&\qquad\qquad (\text{as } e_1 \sim_\rho N_1 \text{ and by induction hypothesis applied to } M')\\
&\quad = [N_1/x]\tilde{M}' \text{ (since } x \text{ is no } x_j \text{ and does not occur free in any } A_j).
\end{aligned}$$

So as $e_2 \sim_\rho N_2, \cdots, e_{n'} \sim_\rho N_{n'}$ we have that

$$\mathscr{V}[\![\lambda x.\, M']\!](\tilde{\rho})e_1 \cdots e_{n'} \sim_\rho ([N_1/x]\tilde{M}')N_2 \cdots N_{n'}$$

and so we see that

$$\begin{aligned}
\mathscr{V}[\![\lambda x.\, M']\!](\tilde{\rho})e_1 \cdots e_{n'} &\sqsubseteq \bigsqcup_n \mathscr{V}[\![\mathscr{S}^n(([N_1/x]\tilde{M}')N_2 \cdots N_{n'})]\!](\rho)\\
&= \bigsqcup_n \mathscr{V}[\![\mathscr{S}^{n+1}((\lambda x.\, \tilde{M}')N_1N_2 \cdots N_{n'})]\!](\rho)\\
&= \bigsqcup_n \mathscr{V}[\![\mathscr{S}^{n+1}((\widetilde{\lambda x.\, M'})N_1 \cdots N_{n'})]\!](\rho)
\end{aligned}$$

as required, concluding the proof. □

Appendix I.

Construction of **D** *and* **D**′. This proceeds as described in [SP]. Let $\mathbf{D}_0$ be a fixed (nontrivial) cpo. For the construction of **D** satisfying

$$\mathbf{D} \cong (\mathbf{D} \to \mathbf{D})$$

first define an ω-chain $\Delta = \langle \mathbf{D}_n, f_n \rangle$ of cpos $\mathbf{D}_n$ and embeddings $f_n : \mathbf{D}_n \to \mathbf{D}_{n+1}$ by

$$\mathbf{D}_{n+1} = (\mathbf{D}_n \to \mathbf{D}_n)$$

and

$$\begin{aligned}
f_0(d) &= \lambda e \in \mathbf{D}_0 .\, d, \qquad f_0^R(g) = g(\perp_{\mathbf{D}_0}),\\
f_{n+1}(g) &= f_n \circ g \circ f_n^R, \qquad f_{n+1}^R(h) = f_n^R \circ h \circ f_n.
\end{aligned}$$

Next **D** is the colimit of Δ, being the sub-cpo of $\prod_n \mathbf{D}_n$ of all sequences d with, for every n, $d_n = f_n^R(d_{n+1})$. We have a cone $\mu : \Delta \to D$ of embeddings where

$$(\mu_n(d_n))_m = \begin{cases} f_{nm}(d_n) & (m \geqq n),\\ f_{mn}^R(d_n) & (m < n), \end{cases}$$

and $\mu_n^R(d) = d_n$, for all n (and where $f_{nm} = f_{m-1} \circ \cdots \circ f_n$, $f_{nm}^R = f_n^R \circ \cdots \circ f_{m-1}^R$). Note that μ_0 is the inclusion of $\mathbf{D}_0$ in **D**. Now we have a cone $\nu : \Delta^- \to (\mathbf{D} \to \mathbf{D})$ where $\Delta^- = \langle \mathbf{D}_{n+1}, f_{n+1} \rangle$ is Δ less its first element and $\nu_n(g) = \mu_n \circ g \circ \mu_n^R$ (and $\nu_n^R(h) = \mu_n^R \circ h \circ \mu_n$).

Then the isomorphism pair

$$\mathbf{D} \underset{\Psi}{\overset{\Phi}{\rightleftharpoons}} (\mathbf{D} \to \mathbf{D})$$

is given by the formulae:

$$\Phi = \bigsqcup_{n \geqq 0} \nu_n \circ \mu_{n+1}^R, \qquad \Psi = \bigsqcup_{n \geqq 0} \mu_{n+1} \circ \nu_n^R.$$

We need to examine the first formula in more detail.

For Φ we calculate that for any d, e in $\mathbf{D}$

$$(\Phi(d)(e)) = \left(\bigsqcup_m \mu_m(d_{m+1}(\mu_m^R(e)))\right) = \bigsqcup_m \mu_m(d_{m+1}(e_m))$$

and then note that for any n, m

$$\begin{aligned}(\Phi(d)(e))_n &= \bigsqcup_{m \geqq n} \mu_n^R(\mu_m(d_{m+1}(e_m))) \\ &= \bigsqcup_{m \geqq n} f_{nm}^R(d_{m+1}(e_m))\end{aligned}$$

(making use of the calculation, $\mu_n^R \circ \mu_m = (\mu_m \circ f_{nm})^R \circ \mu_m = f_{nm}^R \circ \mu_m^R \circ \mu_m = f_{nm}^R$). In the case where $e = \mu_n(e_n)$ with e_n in $\mathbf{D}_n$ we have

$$\begin{aligned}(\Phi(d)(\mu_n e_n))_n &= \bigsqcup_{m \geqq n} f_{nm}^R(d_{m+1}((\mu_n e_n)_m)) \\ &= \bigsqcup_{m \geqq n} f_{nm}^R(d_{m+1}(f_{nm}(e_n))) \\ &= \bigsqcup_{m \geqq n} f_{(n+1)(m+1)}^R(d_{m+1})(e_n) \\ &= d_{n+1}(e_n).\end{aligned}$$

Turning to $\mathbf{D}'$, which is to satisfy the isomorphism

$$\mathbf{D}' \cong (\mathbb{1}^\infty \to \mathbf{D}') \to \mathbf{D}',$$

we take the ω-chain $\Delta' = \langle \mathbf{D}'_n, f'_n \rangle$ of cpos and embeddings where

$$\mathbf{D}'_0 = \mathbf{D}_0, \qquad \mathbf{D}'_{n+1} = (\mathbb{1}^\infty \to \mathbf{D}'_n) \to \mathbf{D}'_n$$

and

$$\begin{aligned}&f'_0(d') = \lambda c' \in (\mathbb{1}^\infty \to \mathbf{D}'_0) \,.\, d', \qquad f_0'^R(g') = g'(\bot_{\mathbb{1}^\infty \to \mathbf{D}'}), \\ &f'_{n+1}(g') = f'_n \circ g' \circ (\lambda c' \in \mathbb{1}^\infty \to \mathbf{D}'_{n+1} \,.\, f_n'^R \circ c'), \\ &f_{n+1}'^R(h') = f_n'^R \circ h' \circ (\lambda c' \in \mathbb{1}^\infty \to \mathbf{D}'_n \,.\, f'_n \circ c').\end{aligned}$$

Now $\mathbf{D}'$ is the colimit of Δ' being the sub-cpo of $\prod_n \mathbf{D}'_n$ of all sequences d' with, for every n, $d'_n = f_n'^R(d'_{n+1})$. We have a cone $\mu' : \Delta' \to \mathbf{D}'$ of embeddings defined as before (and with $\mu_n'^R(d') = d'_n$, as before). Now we have a cone ν' from Δ'^- (defined as before) to $(\mathbb{1}^\infty \to \mathbf{D}') \to \mathbf{D}'$, where $\nu'_n(g') = \mu'_n \circ g' \circ (\lambda c \in \mathbb{1}^\infty \to \mathbf{D}' \,.\, \mu_n'^R \circ c)$ (and $\nu_n'^R(h') = \mu_n'^R \circ h' \circ (\lambda c \in \mathbb{1}^\infty \to \mathbf{D}'_n \,.\, \mu'_n \circ c)$).

Then the isomorphism pair

$$\mathbf{D}' \underset{\Psi'}{\overset{\Phi'}{\rightleftharpoons}} (\mathbb{1}^\infty \to \mathbf{D}') \to \mathbf{D}'$$

is given by the analogous formulae to those for Φ and Ψ. Upon detailed examination these yield for any d' in $\mathbf{D}'$ and $c' \in (\mathbb{1}^\infty \to \mathbf{D}')$ that

$$(\Phi'(d')(c'))_n = \bigsqcup_{m \geqq n} f_{nm}'^R(d'_{m+1}(\mu_m'^R \circ c')).$$

Also for any $d' \in \mathbf{D}'$ and $c'_n \in (\mathbb{1}^\infty \to \mathbf{D}'_n)$ we have that

$$(\Phi'(d')(\mu'_n \circ c'_n))_n = d'_{n+1}(c'_n)$$

The relation between $\mathbf{D}$ *and* $\mathbf{D}'$. First we inductively construct relations $\sim_n$ between $\mathbf{D}_n$ and $\mathbf{D}'_n$ by

$$d \sim_0 d' \quad \text{iff } d = d';$$

$$d \sim_{n+1} d' \quad \text{iff } \forall e \in \mathbf{D}_n,\ c' \in (\mathbb{1}^\infty \to \mathbf{D}'_n)\ .\ e \sim_n c'(\infty) \supset d(e) \sim_n d'(c').$$

We recall that for cpos $\mathbf{D}$ and $\mathbf{E}$ a relation $R \subseteq \mathbf{D} \times \mathbf{E}$ is *inclusive* (termed ω-complete in [SP]) iff it is closed under lubs of increasing sequences, which is to say that if d_n, e_n are increasing sequences in $\mathbf{D}$ and $\mathbf{E}$ respectively, then if $R(d_n, e_n)$ holds for every n so does $R(\sqcup d_n, \sqcup e_n)$.

LEMMA 7. *Each* $\sim_n$ *is inclusive.*

Proof. By induction on n. For $n = 0$ this is obvious. For $n+1$ assume for all $m \geqq 0$ that $d_m \sim_{n+1} d'_m$ where d, d' are increasing sequences. Take e in $\mathbf{D}_n$, c' in $\mathbb{1}^\infty \to \mathbf{D}'_n$ with $e \sim c'(\infty)$. Then $d_m(e) \sim_n d'_m(c')$ and so $\sqcup_m (d_m(e)) \sim_n \sqcup_m (d'_m(c'))$, by induction hypothesis and so we have $(\sqcup_m d)(e) = \sqcup_m (d_m(e)) \sim_n \sqcup_m (d'_m(c')) = (\sqcup_m d')(c')$, concluding the proof.

LEMMA 8. (i) *For all* d *in* $\mathbf{D}_n$ *and* d' *in* $\mathbf{D}'_n$, $d \sim_n d' \supset f_n(d) \sim_{n+1} f'_n(d')$.

(ii) *For all* d *in* $\mathbf{D}_{n+1}$ *and* d' *in* $\mathbf{D}'_{n+1}$, $d \sim_{n+1} d' \supset f_n^R(d) \sim_n f_n'^R(d')$.

Proof. By simultaneous induction on n. For $n = 0$ suppose first that $d \sim_0 d'$. To show $f_0(d) \sim_1 f'_0(d')$ take e in $\mathbf{D}_0$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'_0$ with $e \sim_0 c'(\infty)$. Then $f_0(d)(e) = d$ and $f'_0(d')(c') = d'$, so $f_0(d)(e) \sim_0 f'_0(d')(c')$ as required. Next suppose $d \sim_1 d'$. As $\perp \sim_0 (\lambda k \in \mathbb{1}^\infty\ .\ \perp)(\infty)$ we have $f_0^R(d) = d(\perp) \sim_0 d'(\lambda k \in \mathbb{1}^\infty\ .\ \perp) = f_0'^R(d')$ as required.

For $n+1$, suppose first that $d \sim_{n+1} d'$ and take e in $\mathbf{D}_{n+1}$ and $c' \in (\mathbb{1}^\infty \to \mathbf{D}'_{n+1})$ such that $e \sim_{n+1} c'(\infty)$. Then $f_n^R(e) \sim_n f_n'^R(c'(\infty))$ (by induction hypothesis); so $d(f_n^R(e)) \sim_n d'(f_n'^R \circ c')$ (by assumption); so $f_n(d(f_n^R(e))) \sim_{n+1} f'_n(d(f_n'^R \circ c'))$ (by induction hypothesis). But this is just $f_{n+1}(d)(e) \sim_{n+1} f'_{n+1}(d')(c')$ as required.

Suppose next that $d \sim_{n+2} d'$ and take e in $\mathbf{D}_n$ and $c' \in (\mathbb{1}^\infty \to \mathbf{D}'_n)$ such that $e \sim_n c'(\infty)$. Then $f_n(e) \sim_{n+1} f'_n(c'(\infty))$ (by induction hypothesis); so $d(f_n(e)) \sim_{n+1} d'(f'_n \circ c')$ (by assumption); so $f_n^R(d(f_n(e))) \sim_n f_n'^R(d'(f'_n \circ c'))$ (by induction hypothesis). But this is just $f_{n+1}^R(d)(e) \sim f_{n+1}'^R(d')(c')$ as required, concluding the proof. □

Now we can define our relation between $\mathbf{D}$ and $\mathbf{D}'$ by

$$d \sim d' \quad \text{if } \forall n\ .\ d_n \sim_n d'_n.$$

By Lemma 7 and the componentwise calculation of lubs of ω-sequences in $\mathbf{D}$ and $\mathbf{D}'$ we get that $\sim$ is inclusive. By Lemma 8 we get that for any d in $\mathbf{D}_n$ and d' in $\mathbf{D}'_n$ if $d \sim_n d'$ then $\mu_n(d) \sim \mu'_n(d')$.

We are now in a position to supply the proof of Lemma 2.

Proof of Lemma 2. (i) What this claims is that for any d_0 in $\mathbf{D}_0$ it holds that $\mu_0(d_0) \sim \mu'_0(d_0)$, which follows at once by the above remark and the fact that $d_0 \sim_0 d_0$.

(ii) $\Rightarrow$: First suppose $d \sim d'$ (so that $d_{n+1} \sim d'_{n+1}$ for any n). Take e in $\mathbf{D}$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'$ such that $e \sim c'(\infty)$ (to show that $d(e) \sim d'(c')$, meaning that $\Phi(d)(e) \sim \Phi'(d')(c')$). Now since $e \sim c'(\infty)$ we have for any m that $e_m \sim_m (\mu_m'^R \circ c')(\infty)$ and so by assumption, we have $d_{m+1}(e_m) \sim_m d'_{m+1}(\mu_m'^R \circ c')$. Therefore by Lemmas 7 and 8(ii) we have for any n that

$$\bigsqcup_{m \geqq n} f_{nm}^R(d_{m+1}(e_m)) \sim_n \bigsqcup_{m \geqq n} f_{nm}'^R(d'_{m+1}(\mu_m'^R \circ c'))$$

and by the above remarks on Φ and Φ' this is just

$$(\Phi(d)(e))_n \sim_n (\Phi'(d')(c'))_n$$

showing $\Phi(d)(e) \sim \Phi'(d')(c')$ as required.

(iii) $\Leftarrow$: Suppose that whenever e in $\mathbf{D}$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'$ satisfy $e \sim c'(\infty)$ then $\Phi(d)(e) \sim \Phi'(d')(c')$. We wish to show that $d \sim d'$ and by Lemma 8(ii) it is enough to show $d_{n+1} \sim_{n+1} d'_{n+1}$ for all n. So take e in $\mathbf{D}_n$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'_n$ such that $e \sim_n c'(\infty)$. Then by a previous remark we have $\mu_n(e) \sim (\mu'_n \circ c')\infty$ and so, by our supposition, that $\Phi(d)(\mu_n(e)) \sim \Phi'(d')(\mu'_n \circ c')$.

So we have

$$\begin{aligned} d_{n+1}(e) &= (\Phi(d)(\mu_n(e)))_n \quad \text{(by a calculation given above)} \\ &\sim_n (\Phi'(d')(\mu'_n \circ c'))_n \quad \text{(as just shown)} \\ &= d'_{n+1}(c') \quad \text{(by a calculation given above)} \end{aligned}$$

showing that $d_{n+1} \sim_{n+1} d'_{n+1}$ and thereby concluding the proof. □

The functions s and r. First we inductively define continuous $s_n : \mathbf{D}_n \to \mathbf{D}'_n$ and $r_n : \mathbf{D}'_n \to \mathbf{D}_n$ by

$$s_0 = r_0 = id_{\mathbf{D}_0},$$

$$s_{n+1}(d) = \lambda c' \in \mathbb{1}^\infty \to \mathbf{D}'_n \,.\, s_n(d(r_n(c'(\infty)))),$$

$$r_{n+1}(d') = \lambda e \in \mathbf{D}_n \,.\, r_n(d'(\lambda k \in \mathbb{1}^\infty \,.\, s_n(e))).$$

LEMMA 9. (i) *For all n,* $f'^R_n \circ s_{n+1} \sqsupseteq s_n \circ f^R_n$.

(ii) *For all n,* $f^R_n \circ r_{n+1} \sqsupseteq r_n \circ f'^R_n$.

Proof. By simultaneous induction on n. For $n = 0$, first we have $f'^R_0(s_1(d)) = s_1(d)(\perp) = s_0(d(r_0(\perp(\infty)))) = d(\perp) = s_0(f^R_0(d))$; and second $f^R_0(r_1(d')) = r_1(d')(\perp) = r_0(d'(\lambda k \in \mathbb{1}^\infty \,.\, s_0(\perp))) = d'(\perp) = r_0(f'^R_0(d'))$.

For $n+1$ and part (i) we first calculate that for any d in $\mathbf{D}_{n+2}$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'_n$,

$$\begin{aligned} f'^R_{n+1}(s_{n+2}(d))(c') &= f'^R_n(s_{n+2}(d)(f'_n \circ c')) \\ &= f'^R_n(s_{n+1}(d(r_{n+1}(f'_n(c'(\infty)))))) \end{aligned}$$

and then calculate that

$$s_{n+1}(f^R_{n+1}(d))(c') = s_n(f^R_n(d(f_n(r_n(c'(\infty)))))).$$

But $(f'^R_n \circ s_{n+1}) \sqsupseteq (s_n \circ f^R_n)$ (by induction hypothesis, part (i)) and also $(r_{n+1} \circ f'_n) \sqsupseteq f_n \circ f^R_n \circ r_{n+1} \circ f'_n \sqsupseteq f_n \circ r_n \circ f'^R_n \circ f'_n$ (by induction hypothesis, part (ii)) $= (f_n \circ r_n)$ and applying these two facts enables us to complete the above calculation, showing that

$$f'^R_{n+1}(s_{n+2}(d))(c') \sqsupseteq s_{n+1}(f^R_{n+1}(d))(c')$$

as required. For part (ii) the proof is similar. □

Because of part (i) of this lemma, for any d in $\mathbf{D}$ and any n, the sequence $\langle f'^R_{nm}(s_m(d_m)) \rangle_{m \geqq n}$ is increasing and so we can define $s : \mathbf{D} \to \mathbf{D}'$ by:

$$(s(d))_n = \bigsqcup_{m \geqq n} f'^R_{nm}(s_m(d_m))$$

it being simple to verify that $s(d)$ is in $\mathbf{D}'$. Similarly using part (ii), we can define $r : \mathbf{D}' \to \mathbf{D}$ by

$$(r(d'))_n = \bigsqcup_{m \geqq n} f^R_{nm}(r_m(d'_m)).$$

LEMMA 10. (i) *For all d in* $\mathbf{D}_n$, $d \sim_n s_n(d)$.

(ii) *For all* d *in* $\mathbf{D}_n$ *and* d' *in* $\mathbf{D}'_n$ *if* $d \sim_n d'_n$ *then* $d = r_n(d')$.

Proof. Simultaneous induction on n, the case $n = 0$ being evident. For $n+1$ and part (i) suppose d is in $\mathbf{D}_{n+1}$. Take e in $\mathbf{D}_n$ and c' in $\mathbb{1}^\infty \to \mathbf{D}'_n$ such that $e \sim_n c'(\infty)$ (so that $e = r_n(c'(\infty))$, by induction hypothesis part (ii)). Now we just calculate that

$$\begin{aligned} d(e) &\sim_n s_n(d(e)) \quad \text{(by induction hypothesis part (i))} \\ &= s_n(d(r_n(c'(\infty)))) \\ &= s_{n+1}(d)(c') \end{aligned}$$

as required.

For part (ii) suppose d is in $\mathbf{D}_{n+1}$ and d' is in $\mathbf{D}'_{n+1}$ and $d \sim_{n+1} d'$. For any e in $\mathbf{D}_n$ we have

$$\begin{aligned} e &\sim_n s_n(e) \quad \text{(by induction hypothesis part (i))} \\ &= (\lambda k \in \mathbb{1}^\infty \,.\, s_n(e))(\infty) \end{aligned}$$

and so $d(e) \sim_n d'(\lambda k \in \mathbb{1}^\infty \,.\, s_n(e))$, by assumption and so $d(e) = r_n(d'(\lambda k \in \mathbb{1}^\infty \,.\, s_n(e)))$, by induction hypothesis, part (ii). But this just says, since e was chosen arbitrarily, that $d = r_{n+1}(d')$.

We can now prove Lemma 4.

Proof of Lemma 4. (i) Take d in $\mathbf{D}$. Then for any m we have $d_m \sim_m s_m(d_m)$, by Lemma 10(i), and so for any n and any $m \geqq n$ we have by Lemma 8(ii), $d_n \sim_n f'^R_{nm}(s_m(d_m))$ and so, by Lemma 7, $d_n \sim_n (s(d))_n$, showing $d \sim s(d)$ as required.

(ii) Suppose $d \sim d'$. Then for any m we have $d_m \sim d'_m$ and so $d_m = r_m(d'_m)$ by Lemma 10(ii). So for any n and $m \geqq n$, by Lemma 8(ii) we have $d_n = f^R_{nm}(d_m) = f^R_{nm}(r_m(d'_m))$ and so $d_n = (r(d'))_n$ by the definition of $r(d')$ showing $d = r(d')$, as required. □

Appendix II. We show that a relation, $\sim_\rho$, with the required properties exists. First define relations $\sim^n_\rho$ between $\mathbf{D}_n$ and **Exp** for every $n \geqq 0$ by:

$$d \sim^0_\rho M \quad \text{iff } \forall n' \geqq 0 \forall N_1, \cdots, N_{n'} \,.\, \mu_0(d) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k(MN_1 \cdots N_{n'})]\!](\rho);$$

$$d \sim^{n+1}_\rho M \quad \text{iff } \mu_{n+1}(d) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho) \quad \text{and}$$

$$(\forall e \in \mathbf{D}_n, N \in \mathbf{Exp} \,.\, e \sim^n_\rho N \supset d(e) \sim^n_\rho (MN)).$$

LEMMA 11. (i) $d \sim^0_\rho M$ *iff* $\mu_0(d) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$ *and for all* N, $d \sim^0_\rho (MN)$.

(ii) $\perp \sim^0_\rho M$ *always holds.*

Proof. Obvious.

LEMMA 12. *Each* $\sim^n_\rho$ *is inclusive* (*in its first argument*).

Proof. Easy induction on n.

LEMMA 13. (i) *If* $d \sim^n_\rho M$ *then* $f_n(d) \sim^{n+1}_\rho M$.

(ii) *If* $d \sim^{n+1}_\rho M$ *then* $f^R_n(d) \sim^n_\rho M$.

Proof. By simultaneous induction on n. For $n = 0$, assume for i that $d \sim^0_\rho M$. Then first we calculate that:

$$\mu_1(f_0(d)) = \mu_0(d) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$$

and second, supposing that $e \sim^0_\rho N$ we see that $f_0(d)(e) = d \sim^0_\rho (MN)$, by Lemma 11(i). So $f_0(d) \sim^1_\rho M$ as required.

Next, for (ii) suppose that $d \sim_\rho^1 M$. We show, using Lemma 11 that $f_0^R(d) \sim_\rho^0 M$. First,

$$\mu_0(f_0^R(d)) = \mu_1(f_0 \circ f_0^R(d)) \sqsubseteq \mu_1(d) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho),$$

since $d \sim_\rho^1 M$. Next for any N, since $\bot \sim_\rho^0 N$, by Lemma 11(ii) we have $f_0^R(d) = d(\bot) \sim_\rho^0 (MN)$, as required.

For $n+1$, part (i) suppose $d \sim_\rho^{n+1} M$. Then first we have

$$\begin{aligned}\mu_{n+2}(f_{n+1}(d)) &= \mu_{n+1}(d) \\ &\sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho) \quad (\text{as } d \sim_\rho^{n+1} M)\end{aligned}$$

and second, taking e in $\mathbf{D}_{n+1}$ and N in **Exp** with $e \sim_\rho^{n+1} N$, we see that $f_n^R(e) \sim_\rho^n N$ (by the induction hypothesis, part (i)) and so that $d(f_n^R(e)) \sim_\rho^n (MN)$ (as $d \sim_\rho^{n+1} M$) and so that $f_n(d(f_n^R(e))) \sim_\rho^{n+1} (MN)$ (by induction hypothesis, part (ii)). But this last is just $f_{n+1}(d)(e) \sim_\rho^{n+1} (MN)$ and so we have $f_{n+1}(d) \sim_\rho^{n+2} M$ as required. The verification of part (ii) is very similar and we omit it. □

Note that $\bot \sim_\rho^n M$ always holds and that if $d' \sqsubseteq d \sim_\rho^n M$ then $d' \sim_\rho^n M$, that is $\sim_\rho^n$ is downwards closed in its left argument. All in all, $\{d \mid d \sim_\rho^n M\}$ is always a nonempty Scott closed set.

Now we can define the relation $\sim_\rho$ by:

$$d \sim_\rho M \quad \text{iff } \forall n \,.\, d_n \sim_\rho^n M.$$

This is clearly inclusive using Lemma 12, and indeed $\bot \sim_\rho M$ always holds (and actually $\{d \mid d \sim_\rho M\}$ is always nonempty Scott closed). Note that, by Lemma 13, if $d \sim_\rho^n M$ then $\mu_n(d) \sim_\rho M$.

Now we show that the recursive specification for $\sim_\rho$ is satisfied.

LEMMA 14. *$d \sim_\rho M$ iff $d \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!]\rho$ and*

$$(\forall e \sim_\rho N \,.\, d(e) \sim_\rho (MN)).$$

Proof. This is routine, but perhaps worth writing down.

$\Rightarrow$ Suppose $d \sim_\rho M$. Then $d_n \sim_\rho^n M$, so $\mu_n(d_n) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$ holds for every n and so holds for $d = \bigsqcup_n \mu_n(d_n)$ too. Next, suppose $e \sim_\rho N$. Then $e_n \sim_\rho^n N$ and so, as $d_{n+1} \sim_\rho^{n+1} M$, we have $d_{n+1}(e_n) \sim_\rho^n (MN)$ and so $\mu_n(d_{n+1}(e_n)) \sim_\rho (MN)$ and so, as $d(e) = \bigsqcup \mu_n(d_{n+1}(e_n))$, $d(e) \sim_\rho (MN)$, as required.

$\Leftarrow$ Suppose $d \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$ and whenever $e \sim_\rho N$ then $d(e) \sim_\rho (MN)$. We will show that $d_{n+1} \sim_\rho^{n+1} M$, for every n. Clearly $\mu_{n+1}(d_{n+1}) \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$, from the supposition. Suppose that $e \sim_\rho^n N$. Then $\mu_n(e) \sim_\rho N$ and so $d(\mu_n e) \sim_\rho N$ and so $d_{n+1}(e) = d(\mu_n e)_n \sim_\rho^n N$, as required. □

Proof of Lemma 5. We show by induction on n that for any d, M if for any $n' \geqq 0$ whenever $e_i \sim_\rho N_i$ (for $i = 1, n'$) then

$$d e_1 \cdots e_{n'} \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M N_1 \cdots N_{n'}]\!](\rho)$$

then it follows that $d_n \sim_\rho^n M$.

First suppose $n = 0$. Take $N_1, \cdots, N_{n'}$. Then $\bot \sim_\rho N_i$ and so

$$\mu_0(d_0) = \mu_0(d_0)\bot \cdots \bot \sqsubseteq d \bot \cdots \bot \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M N_1 \cdots N_{n'}]\!](\rho)$$

as required.

For $n+1$, first

$$\mu_{n+1}(d_{n+1}) \sqsubseteq d \sqsubseteq \bigsqcup_k \mathcal{V}[\![\mathcal{S}^k M]\!](\rho)$$

and second, suppose that $e \sim_\rho^n N$. Then $\mu_n(e) \sim_\rho N$. Now suppose that, in order to apply the induction hypothesis to $d(\mu_n e)$, (MN), that $e_i \sim_\rho N_i$ for $i = 1, n'$. Then

$$d(\mu_n e)e_1 \cdots e_{n'} \sqsubseteq \bigsqcup_k \mathscr{V}[\![\mathscr{S}^k(MN)N_1 \cdots N_{n'}]\!](\rho)$$

by assumption on d, M and so $(d(\mu_n e))_n \sim_\rho^n (MN)$. But as $d(\mu_n e)_n = d_{n+1}(e)$, it follows that $d_{n+1}(e) \sim_\rho^n (MN)$ and so, finally, that $d_{n+1} \sim_\rho^{n+1} M$, as required. □

Acknowledgments. We wish to acknowledge the helpful suggestions of the referees.

REFERENCES

[Bar] H. BARENDREGT, *The Lambda Calculus*, North-Holland, Amsterdam, 1981.

[deB] J. DE BAKKER, *Mathematical Theory of Program Correctness*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[Gor] M. J. C. GORDON, *The Denotational Description of Programming Languages*, Springer-Verlag, Berlin, 1979.

[Mil1] R. E. MILNE, *The formal semantics of computer languages and their implementations*, Ph.D. thesis, Univ. Cambridge, 1974.

[Mil2] R. E. MILNE AND C. STRACHEY, *A Theory of Programming Language Semantics*, Chapman and Hall, London, 1976.

[Mul] K. MULMULEY, *A mechanizable theory for existence proofs of inclusive predicates*, to appear.

[Sco] D. S. SCOTT, *Continuous lattices*, in Toposes, Algebraic Geometry and Logic, Proc. Dalhousie Conference 1971, F. W. Lawvere, ed., Lecture Notes in Mathematics 274, Springer-Verlag, Berlin, 1972.

[SP] M. B. SMYTH AND G. D. PLOTKIN, *The category-theoretic solution of recursive domain equations*, this Journal, 11, (1982), pp. 761-783.

[Sto1] J. E. STOY, *The Scott-Strachey Approach to Programming Language Theory*, 1977, MIT Press, Cambridge, MA.

[Sto2] ———, *The congruence of two programming language definitions*, TCS, 13 (1981), pp. 151-174.

[Ten] R. D. TENNENT, *Principles of Programming Languages*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Wad] C. P. WADSWORTH, *The relation between computational and denotational properties for Scott's D_∞-models of the lambda calculus*, this Journal, 5 (1976), pp. 488-521.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987
© 1987 Society for Industrial and Applied Mathematics
016

SPEED-UP OF TURING MACHINES WITH ONE WORK TAPE AND A TWO-WAY INPUT TAPE*

WOLFGANG MAASS† AND AMIR SCHORR‡

Abstract. In this paper we consider the next more powerful restricted type of Turing machine, which cannot be handled by the existing lower bound arguments: Turing machines with one work tape and a two-way input tape. We show that one can simulate a deterministic Turing machine of this type with time bound $O(n^3)$ by a Σ_2-Turing machine *of the same type* with time bound $O(n^2 \cdot \log^2 n)$. This implies the new separation result $\Sigma_2\,\mathrm{TIME}_1(n) \not\subseteq \mathrm{DTIME}_1(n^{3/2}/\log^6 n)$. Further, we improve Kannan's separation result $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.104})$ to $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.22})$. Finally we show that with Turing machines of the considered type that use $2k$ alternations, the achieved speed-up increases and for large k approximates $t^{1/2}(n)$. In view of the close relationship between spacebounded Turing machines and time-bounded Turing machines with unboundedly many alternations, this refinement provides a link between our speed-up result and the well-known space-compression result for offline 1-tape Turing machines.

Key words. Turing machines, complexity classes, lower bound arguments

AMS(MOS) subject classifications. 3D10, 3D15, 68C40

1. Introduction. A major goal of machine based complexity theory is the separation of complexity classes that arise from Turing machines with different control structures: deterministic, nondeterministic, and alternating. Since an alternating Turing machine may be viewed as a model for a highly parallel computer, such separation results are relevant for the comparison between sequential and parallel computational models (in addition to their interest for the P versus NP problem). With regard to such separation results one is currently in the following situation. On one hand one can separate deterministic linear time and nondeterministic linear time for the most powerful types of Turing machines: multi-tape Turing machines. But the demonstrated difference between deterministic and nondeterministic complexity classes is "infinitesimal": $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}(n \cdot (\log^* n)^{1/4})$ (Paul, Pippenger, Szemeredi and Trotter [11]). On the other hand more significant time differences between determinism and nondeterminism have been demonstrated for restricted types of Turing machines: a quadratic difference for 1-tape Turing machines without input tape by Hennie [3], and for 1-tape Turing machines with an additional 1-way input tape by Maass [7].

Since the more abstract methods that yield the separation result for multi-tape Turing machines can probably not be used to prove more significant time differences (see the discussion in [11]), it appears reasonable to look for more concrete arguments that yield larger time differences for increasingly more powerful *restricted* types of Turing machines (the goal is to approximate the power of 2-tape Turing machines from below). In this paper we attack the next more powerful restricted type of Turing machine that cannot be handled by previously developed concrete lower bound arguments: Turing machines with one work tape and a *two*-way input tape.

In § 2 we show that one can speed up deterministic Turing machines of this type by Turing machines of the same type (i.e. one work tape and a 2-way input tape) that use a bounded number of alternations. In particular we show that

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_2\,\mathrm{TIME}_1(t^{2/3}(n) \cdot \log^2 t(n)) \quad \text{for } t(n) \geqq n^3,$$

* Received by the editors October 16, 1985; accepted for publication (in revised form) April 28, 1986.

† Department of Mathematics, Statistics, and Computer Science, University of Illinois, Chicago, Illinois 60680. This author's work was supported in part by the National Science Foundation under grant DCR-8504247.

‡ Department of Computer Science, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel.

and more generally, that

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_{2k}\,\mathrm{TIME}_1(t^{(k+1)/(2k+1)}(n)\cdot \log^2 t(n))$$

$$\text{for } k \geqq 1 \text{ and } t(n) \geqq n^{2+1/k}$$

(see below for definitions).

In § 3 we derive from this speed-up result two new separation results for the considered type of Turing machine:

$$\Sigma_{2k}\,\mathrm{TIME}_1(n) \not\subseteq \mathrm{DTIME}_1(n^{(2k+1)/(k+1)}/\log^6 n) \quad \text{for all } k \geqq 1,$$

and

$$\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.22}).$$

The following definitions are used in this paper. A k-tape Turing machine (TM) is a TM with k work tapes (each with one head) and an additional 2-way input tape. This 2-way input tape is an extra read-only tape that is occupied by the input together with a left and a right endmarker. The associated (read-only) input head can move in both directions ("2-way").

Alternating TM's were introduced by Chandra, Kozen and Stockmeyer [2]. Each state of an alternating TM is either existential or universal. One calls a node v in the computation tree of an alternating TM M for some input w *accepting* if

—either M is at v in an existential state and there exists a son of v that is accepting,

—or M is at v in a universal state and all the sons of v are accepting,

—or M is at v in an accepting final state.

M accepts w if the root of this computation tree is accepting. M is called a Σ_m-TM if for all inputs w the computation tree of M starts with an existential state and alternates along each computation path at most $m-1$ times between existential and universal states. Thus Σ_1-TM's are just nondeterministic TM's. Deterministic TM's ($=\Sigma_0$-TM's) may be viewed as special cases of nondeterministic TM's, where the transition function is single-valued.

For any such TM, we say that M *is of time complexity* $t(n)$ if for every accepted input w of length n the computation tree for input w stays accepting if it is pruned at depth $t(n)$ (see Paul, Prauss and Reischuk [9]). This definition is convenient insofar as one does not have to worry too much about the time-constructibility of $t(n)$ (otherwise one might demand instead that the depth the depth of the computation tree of M on input w is bounded by $t(n)$).

We define for arbitrary functions $t(n) \geqq n$ $\Sigma_m\,\mathrm{TIME}_k(t(n)) = \{L \subseteq \{0,1\}^* \mid L$ is accepted by a k-tape Σ_m-TM of time complexity $O(t(n))\}$.

Among related results, we would like to mention (besides the already discussed paper [11]) those by Paul, Prauss and Reischuk [9], [10] and Kannan [5]. It is shown in [9] that every nondeterministic 1-tape TM *without* input tape, of time complexity $t(n)$, can be simulated by an alternating 1-tape TM of time complexity $O(n+t^{1/2}(n))$, which uses an *unbounded* number of alternations. Kannan [5] shows that any deterministic 1-tape TM with two-way input tape of time complexity n^{2l} ($l \geqq 1$ integer) can be simulated by a *multitape* Σ_4-TM of time complexity $O(n^{1+(4/3)\cdot l}\log n)$. This implies the separation result $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.104})$, which we improve in this paper to $\mathrm{NTIME} \not\subseteq \mathrm{DTIME}_1(n^{1.22})$.

Compared with the preceding results on the speed-up of deterministic TM's by alternating TM's, we use in this paper a somewhat different strategy in order to speed

up the considered restricted type of deterministic TM by a TM of the *same* restricted type, which uses only a *constant* number of alternations. This allows us to separate complexity classes that arise from restricted TM's, which differ *only* in their control structure.

Our speed-up from $t(n)$ to $t^{(k+1)/(2k+1)}(n) \cdot \log^2 t(n)$ with $2k-1$ alternations approximates $t^{1/2}(n)$ (for $(2k-1 \to \infty)$. This fits nicely together with the known time-space tradeoff for the considered restricted TM's. Space-bounded TM's are closely related to TM's which use an unbounded number of alternations (it follows from [2] and [9] that $\mathrm{ATIME}_1(t(n)) \subseteq \mathrm{DSPACE}_1(t(n)) \subseteq \mathrm{ATIME}_1(t^2(n))$). Thus the well-known result $\mathrm{DTIME}_1(t(n)) \subseteq \mathrm{DSPACE}_1(t^{1/2}(n))$ of Paterson [8] (see Ibarra and Moran [4] and Kurtz and Maass [6] for some recent refinements) as well as the mentioned result from [9] may be viewed as limit cases of the speed-up result in this paper.

2. Speed-up by bounded alternation.

THEOREM 2.1. *For every function* $t(n) \geqq n$

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_2 \mathrm{TIME}_1(t^{2/3}(n) \cdot \log^2 t(n) + t^{1/3}(n) \cdot n).$$

In particular for every function $t(n) \geqq n^3$

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_2 \mathrm{TIME}_1(t^{2/3}(n) \cdot \log^2 t(n)).$$

Proof. Let M be a deterministic TM of time-complexity $t(n)$ with one work tape and a two-way input tape. We construct a Σ_2-TM $\tilde{M}$ with one work tape and a two-way input tape, that accepts the same inputs as M and is of the desired time complexity.

For any fixed input w of length n that is accepted by M, the simulating TM $\tilde{M}$ guesses a suitable separator for the computation graph of M. Subsequently $\tilde{M}$ verifies in parallel that for each component of the remaining graph the associated guesses are consistent.

We first describe the design of the separator. One partitions the work tape of M into blocks B of length $t^{1/3}(n)$ in such a way that the work head of M crosses at most $t^{2/3}(n)$ often a block boundary. Such partition exists, since there are $t^{1/3}(n)$ different partitions into blocks of length $t^{1/3}(n)$. Further, the *sum* of the number of crossings of block boundaries for all $t^{1/3}(n)$ partitions is bounded by $t(n)$ (since at every step the work head of M crosses for at most one partition a block boundary).

For each of those blocks B of the fixed partition of M's work tape such that M's head spends altogether at least $t^{2/3}(n)$ steps in B or leaves and re-enters B at least $t^{1/3}(n)$ often, one partitions the sequence of steps where M's work head is inside B into time blocks T_B. Each time block T_B is chosen as large as possible, subject to these two constraints:

1) T_B consists of at most $t^{2/3}(n)$ steps.

2) T_B contains at most $t^{1/3}(n)$ steps where the work head of M leaves B at the next step (note that in such a case the next element of T_B is the first subsequent step where the head re-enters B—unless we have reached the end of T_B).

For the previously described partition of space and time $\tilde{M}$ writes (during the $\exists$-space of the accepting subtree of its computation on input w) the following "guessed" data on its work tape. These data are arranged in a particular way, to support the subsequent "verification" during $\tilde{M}$'s $\forall$-phase. For the fixed $n = |w|$ we write from now on t instead of $t(n)$.

Starting with the leftmost tape block B that is visited by M's work head, $\tilde{M}$ writes from left to right for each block B the following guessed data associated with B on its work tape.

If those steps, where M's work head is inside B, have been partitioned into time blocks T_B, $\tilde{M}$ writes from left to right for each time block T_B the inscription of B at the beginning and at the end of T_B on two tracks of a tape segment S of length $t^{1/3}$. On two additional tracks $\tilde{M}$ marks on tape segment S the position of M's work head at the beginning and end of T_B together with the state of M and the position of M's input head (represented by a number from $\{1, \cdots, n\}$ in binary code) at these two steps. To the left and right of the considered segment S of $\tilde{M}$'s work tape $\tilde{M}$ writes those entries of the crossing sequence of M at the left and right ends of block B, that are associated with steps in time block T_B. By our choice of the length of T_B there are at most $t^{1/3}$ such entries.

Each entry in a crossing sequence for a boundary b on M's work tape consists in the present construction of the current state and input head position of M, together with the number of preceding crossings of this boundary b (represented by a number between 0 and $t-1$ in binary code). The latter number will be useful in the subsequent verification phase, where we have to verify in particular that the guessed crossing sequence for the right end of a tape block B agrees with the guessed crossing sequence for the left end of the next tape block B' to the right of B. The bit-length of a segment of a crossing sequence with up to $t^{1/3}$ entries is of the order $O(t^{1/3} \cdot \log t)$.

After $\tilde{M}$ has written on its work tape the described guesses associated with time block T_B, it writes to the right of it the analogous guesses for the next time block T'_B for the same tape block B. After all time blocks for tape block B are handled, $\tilde{M}$ writes to the right of these guesses those guesses that are associated with the next block B' to the right of B.

In the case that the steps where M's work head is inside B have *not* been partitioned into time blocks T_B, $\tilde{M}$ guesses, instead of the preceding, just the crossing sequences for the left and right ends of B. $\tilde{M}$ also places markers in distances of $t^{1/3}$ along the guessed string (to provide a tape segment that has exactly the same length as B for the subsequent simulation of the computation inside B). If the work head of M happens to be inside B during the first or last step of M's computation, $\tilde{M}$ guesses in addition the state and head positions of M for these two steps. Note that for the blocks B considered here, $\tilde{M}$ does not guess the initial or final tape content of B (in order to keep the total length of its guesses sufficiently short).

For the chosen partition of M's work tape there are at most $t^{2/3}$ crossings of block boundaries. Thus all guesses of segments of crossing sequences together require $O(t^{2/3} \cdot \log t)$ bits on the work tape of $\tilde{M}$.

It is also easy to see that there are altogether at most $O(t^{1/3})$ many time blocks T_B, since every time block either consists of $t^{2/3}$ steps, or involves $t^{1/3}$ crossings of a boundary of B, or is the last one of at least two time blocks for the same tape block B. To verify this estimate, we observe that there are altogether at most $t^{1/3}$ time blocks T_B that consist of $t^{2/3}$ steps (because M uses only t steps), and altogether at most $t^{1/3}$ time blocks T_B that contain $t^{1/3}$ steps where M's head leaves B at the next step (because at most $t^{2/3}$ crossings of block boundaries occur).

The preceding implies that the guessed initial and final inscriptions of blocks B for time blocks T_B occupy together at most $O(t^{2/3}(n))$ cells on the work tape of $\tilde{M}$. Thus the total guessing phase of $\tilde{M}$ requires no more than $O(t^{2/3} \cdot \log t)$ bits.

We would like to point out that M's work head may visit during its computation up to $t^{2/3}(n)+2$ many different tape blocks B. Therefore $\tilde{M}$ could not afford to guess

initial and final tape inscriptions for all these blocks B. This explains why blocks B with and without time blocks T_B were handled differently in the preceding.

In the following $\forall$-phase the Σ_2-TM $\tilde{M}$ verifies in parallel the following facts about the guessed data:

FACT 1. *For each time block T_B the following guessed data are consistent: the initial and final inscriptions of block B, the initial and final state and head positions of M, and the associated segments of the crossing sequences for the left and right boundary of B.*

To verify this consistency, $\tilde{M}$ simulates the corresponding computation steps of M, sweep by sweep. Each time when $\tilde{M}$ simulates another sweep over B, its work head moves first to the next entry of the corresponding crossing sequence. The length of this move can be estimated by the bit length of the segments of the two crossing sequences associated with T_B: $O(t^{1/3} \cdot \log t)$. Once $\tilde{M}$'s work head has arrived at the next entry of the crossing sequence, $\tilde{M}$ brings its input head into the position that is prescribed by the corresponding binary number between 1 and n in this entry of the crossing sequence.

This procedure takes $O(n)$ steps. We assume that the subsequent simulation of the next sweep of M's work head over B is simulated on the same tape segment of $\tilde{M}$'s work tape, where it has recorded its guess of the initial and final block content for T_B (we may use an extra track for this simulation). After the simulation of each sweep over B, $\tilde{M}$ moves its work head to the corresponding next entry of the crossing sequence, in order to compare its current state and input head position with the corresponding guessed data.

Of course $\tilde{M}$ starts the simulation of the first sweep over B that belongs to T_B with the guessed initial content of B and with the guessed state and head position. Analogously, after $\tilde{M}$ has simulated the last sweep of T_B over B, it compares the current content of the track that simulates B and the current state and head positions with the guessed data for T_B.

It is obvious that the described verification of the consistency of the guessed data for a time block T_B requires at most

$$O(t^{2/3} \cdot \log t + t^{1/3} \cdot n)$$

steps.

FACT 2. *For each block B without time blocks the guessed data are consistent.*

This is verified by simulating all sweeps of M's work head over B, in the same way as above (initially B is blank). The fact that B has no time blocks implies that the same time bound as in Fact 1 holds.

FACT 3. *For any two consecutive time blocks T_B and T'_B for the same tape block B the guessed final block inscription, state and head positions of T_B agree with the initial block inscription, state and head positions of T'_B.*

Here $\tilde{M}$ has to compare two strings of length $O(t^{1/3})$, that are written at most $O(t^{1/3} \cdot \log t)$ cells apart on $\tilde{M}$'s work tape (only the associated segments of crossing sequences are written in between). This procedure takes $O(t^{2/3} \cdot \log t)$ steps.

FACT 4. *For any two consecutive tape blocks B and B', each entry in the crossing sequence for the right end of B agrees with the corresponding entry in the crossing sequence for the left end of B'.*

More precisely $\tilde{M}$ checks here the following. If $\tilde{M}$ places markers above any entry e_1 in the crossing sequence at the right end of B and any entry e_2 in the crossing sequence at the left end of B', and if $\tilde{M}$ verifies that both entries have assigned the same running number (in their respective crossing sequence), then e_1 and e_2 agree.

The considered entries e_1, e_2 are located on $\tilde{M}$'s work tape at most $O(t^{2/3} \cdot \log t)$ cells apart. The work head of $\tilde{M}$ has to cover the distance in between at most $O(\log t)$ often in the described procedure. Therefore this procedure takes $O(t^{2/3} \cdot \log^2 t)$ steps.

FACT 5. *Consecutive entries in the same crossing sequence have been assigned consecutive running numbers.*

Two consecutive entries may be assigned to two different time blocks. However they are at most $O(t^{1/3} \cdot \log t)$ cells apart. Thus for any two consecutive entries $\tilde{M}$ can verify this in $O(t^{1/3} \cdot \log^2 t)$ steps.

FACT 6. *For exactly one block B the computation starts in B (with correct initial state and initial head positions) and for exactly one block B' the computation ends in B' in an accepting final state.*

$\tilde{M}$ can verify this with $O(1)$ sweeps over the guessed data.

The description of $\tilde{M}$'s $\forall$-phase is now complete.

The preceding arguments imply that if input w of length n is accepted by M with $t(n)$ steps, $\tilde{M}$ accepts w via a subtree of the computation tree that has depth $O(t^{2/3}(n) \cdot \log^2 t(n) + t^{1/3}(n) \cdot n)$. On the other hand one constructs $\tilde{M}$ so that whenever $\tilde{M}$ accepts some input w (with a subtree of arbitrary depth), then M also accepts w. Therefore $\tilde{M}$ accepts the same language as M, and $\tilde{M}$ is of time complexity $O(t^{2/3}(n) \cdot \log^2 t(n) + t^{1/3}(n) \cdot n)$.

THEOREM 2.2. *For all natural numbers $k \geqq 1$ and every function $t(n) \geqq n$*

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_{2k}\,\mathrm{TIME}_1(t^{(k+1)/(2k+1)}(n) \cdot \log^2 t(n) + t^{1/(2k+1)}(n) \cdot n).$$

In particular for $t(n) \geqq n^{2+1/k}$

$$\mathrm{DTIME}_1(t(n)) \subseteq \Sigma_{2k}\,\mathrm{TIME}_1(t^{(k+1)/(2k+1)}(n) \cdot \log^2 t(n)).$$

Proof. The proof is analogous to that of Theorem 2.1. We use the additional alternations of the Σ_{2k}-TM $\tilde{M}$ to refine the partition of the given computation of the deterministic TM M. For a given input w of length n we cut the work tape of M in such a way into blocks B of length $t^{k/(2k+1)}$, that M's work head crosses in the computation on input w at most $t^{(k+1)/(2k+1)}$ often a block boundary (again we write t instead of $t(n)$). Further for those blocks B in which M spends $t^{2k/(2k+1)}$ steps, or whose boundary it crosses $t^{k/(2k+1)}$ often, one cuts the sequence of steps where M's work head is inside B into time blocks T_B with $\leqq t^{2k/(2k+1)}$ steps and $\leqq t^{k/(2k+1)}$ crossings. This can be done in such a way that only $O(t^{1/(2k+1)})$ time blocks arise. Thus $\tilde{M}$ can afford to guess for all time blocks T_B the intermediate contents of block B. In the subsequent $\forall$-phase $\tilde{M}$ starts to check in parallel that for each time block T_B the guessed data are consistent. Since a direct simulation of T_B would take too long, $\tilde{M}$ uses the second $\exists$-phase to cut T_B into smaller time blocks $T_B^{(2)}$ that consist of $\leqq t^{(2k-1)/(2k+1)}$ steps and have $\leqq t^{(k-1)/(2k+1)}$ crossings. Again this can be done so that each time block T_B consists of at most $O(t^{1/(2k+1)})$ smaller time blocks $T_B^{(2)}$. If necessary, one has to cut also for those blocks B that have no time blocks the steps that M spends inside B into smaller time blocks $T_B^{(2)}$ (of the same maximal size as above).

By continuing this process of guessing in parallel, one finally arrives in the kth $\exists$-phase at very small time blocks $T_B^{(k)}$ with $\leqq t^{(k+1)/(2k+1)}$ steps and $\leqq t^{1/(2k+1)}$ crossings. The consistency of the guessed data for each $T_B^{(k)}$ can be checked in parallel by a simulation of the corresponding segments of M's computation.

Similarly as in the proof of Theorem 2.1 several additional facts about the guessed data have to be verified to make sure that whenever $\tilde{M}$ accepts w (with an arbitrary number of steps) then M also accepts w. In particular one has to check that for any two subsequent smaller time blocks $T_B^{(i+1)}$ the guessed content of B at the end of the

first one agrees with the guessed content of B at the beginning of the second one. A straightforward verification of this fact would take too long. Therefore $\tilde{M}$ guesses in addition a partition of B into mini-blocks of length $t^{1/(2k+1)}$. Then $\tilde{M}$ can verify the desired agreement of guessed inscriptions of B in parallel, separately for each mini-block of B.

3. Application to separation problems.

THEOREM 3.1. *For every natural number* $k \geqq 1$

$$\Sigma_{2k}\,\mathrm{TIME}_1(n) \not\subseteq \mathrm{DTIME}_1\left(\frac{n^{(2k+1)/(k+1)}}{\log^6 n}\right).$$

Remark 3.2. Note that the TM's that are considered in Theorem 3.1 differ *only* in their control structure, not in their memory or input device.

Proof. This separation result follows from the preceding speed-up result (Theorem 2.2) in combination with an adaption of standard diagonalization and padding techniques to the considered type of TM.

Via diagonalization, one shows that

$$\mathrm{DTIME}_1(n^{2+1/k}) \not\subseteq \mathrm{DTIME}_1\left(\frac{n^{2+1/k}}{\log n \cdot \log^* n}\right)$$

(besides the program of the simulated machine the simulating machine also moves a step counter along with work head; this causes the factor $1/\log n$ on the right-hand side). Together with Theorem 2.2 this implies that

$$\Sigma_{2k}\,\mathrm{TIME}_1(n^{(k+1)/k} \cdot \log^2 n) \not\subseteq \mathrm{DTIME}_1\left(\frac{n^{2+1/k}}{\log n \cdot \log^* n}\right).$$

Via padding, one can derive from this

$$\Sigma_{2k}\,\mathrm{TIME}_1(n) \not\subseteq \mathrm{DTIME}_1\left(\frac{n^{(2k+1)/(k+1)}}{\log^{(6k+4)/(k+1)} n \cdot \log^* n}\right)$$

(this is straightforward but not completely trivial: in order to simulate a TM $\tilde{M}$ that processes the padded input by a TM M that processes the unpadded input, M has to record the current position of $\tilde{M}$'s input head via a binary counter). This implies the claim of the theorem.

The following theorem improves a separation result by Kannan [4], who had shown that $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.104})$. It is well known that $\mathrm{NTIME}(n) = \mathrm{NTIME}_2(n)$ [1].

THEOREM 3.3. $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTIME}_1(n^{1.22})$.

Proof. We use a method of Paul and Reischuk [10] to convert a computation with alternations into a deterministic computation. Assume for a contradiction that $\mathrm{NTIME}(n) \subseteq \mathrm{DTIME}_1(n^{1.22})$. With the help of padding one can show that this implies $\mathrm{NTIME}(n^{1.22}) \subseteq \mathrm{DTIME}_1((n^{1.22})^2 \cdot \log n)$. We apply both inclusions to a linear time Σ_2-TM M with one work tape and a two-way input tape such that

$$L(M) \in \Sigma_2\,\mathrm{TIME}_1(n) - \mathrm{DTIME}_1\left(\frac{n^{3/2}}{\log^6 n}\right)$$

(such TM M exists by Theorem 3.1). For an arbitrary fixed input w of length n one replaces in the computation tree of M on input w each maximal $\forall$-subtree by a deterministic computation of length $O(n^{1.22})$ (we use here the first inclusion). The resulting computation tree can be construed as a nondeterministic computation of

length $O(n^{1.22})$ on a two-tape TM $\tilde{M}$ (it would be nice if we could view $\tilde{M}$ as a TM with *one* work tape and a two-way input tape, because then we could replace $\mathrm{NTIME}(n)$ by $\mathrm{NTIME}_1(n)$ in the claim of the theorem; however the implanted deterministic TM of time complexity $O(n^{1.22})$ uses both the content of M's input tape and the content of M's work tape at the beginning of the $\forall$-phase as its input). Thus we can apply the second inclusion. In this way one derives

$$L(M) \in \mathrm{DTIME}_1(n^{(1.22)^2} \cdot \log n) \subseteq \mathrm{DTIME}_1(n^{3/2}/\log^6 n),$$

a contradiction to the choice of M.

Note that the constant 1.22 that arises in this separation result can be replaced by any number less than $\sqrt{3/2}$.

Acknowledgment. We would like to thank the referees for their helpful comments.

REFERENCES

[1] R. V. Book, S. A. Greibach and B. Wegbreit, *Time and tape bounded Turing acceptors and* AFL's, J. Comput. System Sci., 4 (1970), pp. 606–621.

[2] A. K. Chandra, D. Kozen and L. J. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[3] F. C. Hennie, *One-tape, off-line Turing machine computations*, Inform. and Control, 8 (1965), pp. 553–578.

[4] O. H. Ibarra and S. Moran, *Some time-space trade-off results concerning single tape and offline Turing machines*, this Journal, 12 (1983), pp. 388–394.

[5] R. Kannan, *Alternation and the power of nondeterminism*, Proc. of the 15th ACM Symposium on the Theory of Computing, 1983, pp. 344–346.

[6] S. A. Kurtz and W. Maass, *Some time, space and reversal tradeoffs*, Inform. and Control, to appear.

[7] W. Maass, *Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines*, Trans. Amer. Math. Soc., 292 (1985), pp. 675–693.

[8] M. Paterson, *Tape bounds for time-bounded Turing machines*, J. Comput. System Sci., 6 (1972), pp. 116–124.

[9] W. J. Paul, E. J. Prauss and E. J. Reischuk, *On alternation*, Acta Informatica, 14 (1980), pp. 243–255.

[10] W. J. Paul and E. J. Reischuk, *On alternation* II, Acta Inform., 14 (1980), pp. 391–403.

[11] W. J. Paul, N. Pippenger, E. Szemeredi and W. Trotter, *On determinism versus nondeterminism and related problems*, Proc. 24th IEEE Symposium on the Foundation of Computer Science, 1983, pp. 429–438.

SIAM J. COMPUT.
Vol. 16, No. 1, February 1987

© 1987 Society for Industrial and Applied Mathematics

RANDOM ITERATIONS OF THRESHOLD NETWORKS AND ASSOCIATIVE MEMORY*

FRANÇOISE FOGELMAN SOULIE† AND GÉRARD WEISBUCH‡

Abstract. A threshold network that can memorize and retrieve a given set of patterns X_S is presented. A theoretical framework is given to allow expression of this property in terms of the dynamics of the network: the patterns in X_S must be attractive fixed points. We give sufficient conditions for this, using a concept of energy. Scaling laws for the storing capacity of random patterns are provided and tested in the case of random as well as nonrandom patterns, such as the letters of the alphabet.

Key words. discrete iterations, fixed points, attractivity, associative memory, distributed computation

AMS(MOS) subject classifications. 68Q80, 68T10

1. Introduction. Rapid evolution of the VLSI technology and introduction of parallel computers have recently evoked new interest in the computational properties of large ensembles of interconnected devices. It has been argued that such networks could be adequately used in many problems, such as learning, pattern and speech recognition, constraint satisfaction searches, associative memory, etc.

In many instances, the problem at hand is sufficiently well delineated so as to allow for the design of a dedicated efficient hardware architecture [10], [12]. But, when the class of problems is broader, it would be more adequate to have the network structure adapt itself to the incoming problem.

Recent work has been devoted to this question and various solutions have been proposed: in most cases, the network is designed so as to have a dynamical behavior whose attractors, usually fixed points, encode the useful information, for example the patterns in pattern recognition, or the addresses in content addressable memory.

Huberman and Hogg [7], [8] have designed a network of locally interconnected units capable of learning any given patterns and retrieving them from partial information. The array is self-repairing and may easily be implemented on a VLSI chip, thus providing a very fast and reliable computing device.

Following the lines of Minsky and Papert's perceptron [13], researchers have recently proposed to build networks of neuron like elements, namely threshold devices. Physicists Little [11], Hopfield [6] and Peretto [14] have shown the formal analogy between a threshold network and a spin glass which allowed them to use statistical mechanics in the study of the dynamics of such networks.

In his work on associative memory, Kohonen [10] has proposed to use linear models to store and retrieve information: after the learning process, during which patterns are successively presented to the network, it will be possible to retrieve a memorized item by presenting to the network a partial or noised version. Thus memories are stored into the system in a way similar to so-called *content-addressable memories*: each item can be accessed by a code (its "address") which is related to the item itself.

Kohonen's technique [10] is based on the use of the generalized inverse matrix theory: if $x^1, \cdots, x^m \in \mathbb{R}^n$ are the patterns to be memorized, then by recursively computing matrix $M = XX^+$ (X is the matrix with columns $x^1, \cdots, x^m$ and X^+ is its

* Received by the editors January 23, 1985; accepted for publication (in revised form) April 1, 1986. This work was partially supported by Xerox PARC, which also provided research facilities during a visit by the authors. Partial support was also provided by INSERM grant 83 80 004.

† Laboratoire de Dynamique des Réseaux, c/o CESTA, 1 rue Descartes, 75 005 Paris, and University of Paris V, Paris, France.

‡ Laboratoire de Physique des Solides, Ecole Normale Supérieure, 24 rue Lhomond, 75 231 Paris, France.

generalized inverse), the network with connection matrix M will "retrieve" the x^k's when shown some close pattern x. If x is close to x^k ($x = x^k + v$), then it is shown that Mx is even closer to x^k.

Note that the method presented by Kohonen to implement associative memories is different—in its very purpose—from the classical techniques: in hash coding, for example, patterns $x^1, \cdots, x^m$ are accessed by their addresses $f(x^1), \cdots, f(x^m)$ and function f is designed so as to "spread over" the items $x^1, \cdots, x^m$ in order to minimize the number of collisions. The system has to provide an *exact* answer to any query x, or stated in different words, an exact matching between the key x and the content of $f(x)$. To achieve that, it is necessary to devise a good process to handle overflows.

In Kohonen's system, on the contrary, the goal is to achieve the best possible matching between a given key and stored items. Different keys may be classified (i.e. recognized) in the same class: if they look "similar" to some x^k, they will retrieve some pattern *close to* (but not necessarily identical to) x^k. Hence collisions are not avoided, and no overflow handling process is needed: collisions are used just the opposite to cope with interesting "similarities" among patterns.

Performances of this model are quite good, but it must be stressed that such a model cannot learn *any* association: only first order predicates (see [13]) can be taught such networks and higher order predicates need more sophisticated architectures. This is also a limitation to the models based on threshold units discussed in this paper. Recent work by Hinton and colleagues [5] on "Boltzmann machines" has proved that this limitation could be overcome: pushing the analogy with spin glasses, they propose a learning algorithm by which the connection strengths in the network (i.e. the weights of the threshold elements) can be modified so as to make the desired patterns fixed points of the dynamics. This approach proves that multi-layers networks or networks with "hidden units," such as the Boltzmann machine [5], can solve higher order problems. Other work on multi-layered networks: Fukushima and Miyake [2], and d'Humières and Huberman [9] have shown that such structures exhibit interesting learning and associative memory abilities; but these architectures, which embody some features of brain functioning, are very hard to handle.

These different contributions show that the possibility of building efficient adaptive networks is now at hand: locally connected networks may be implemented on VLSI chips, thus eventually providing very powerful machines for pattern or speech recognition, associative memory, etc. Nevertheless, in order to design the most efficient architectures, much effort is still needed to understand and predict the behavior of such networks.

In this paper, we present a general theoretical framework that allows us to study the building of an associative memory by means of a threshold network. Using techniques introduced in discrete iteration theory [1], [3], [16], we give theoretical conditions to make a network memorize a given set of patterns (Lemma 1) and retrieve them from only partial information (Theorem 3). We apply these theoretical results to the particular case of a rule borrowed from Little's and Hopfield's work—which is shown to be closely correlated with Kohonen's—and compare the performances in the cases of global and local connections. Simulations are shown to support the results.

In § 2, we give definitions. In § 3, we provide a necessary and sufficient condition for memorizing a given set of patterns X_S by a threshold network and compute the bound to the storing capacity of the network, in the case of the Little-Hopfield conditions. In § 4, we study the attractivity of a fixed point in its first neighborhood for a random iteration, which allows us to find a sufficient condition for attractivity and thus for associative memory.

2. Definitions. Let $F: \{0,1\}^n \to \{0,1\}^n$ be a mapping whose components $f_1, f_2, \cdots, f_n$ are *threshold functions*:

$$\forall i(1,\cdots,n), \quad \forall x \in \{0,1\}^n, \qquad f_i(x) = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} a_{ij}x_j - b_i \geqq 0, \\ 0 & \text{otherwise.} \end{cases}$$

a_{ij} is called the *weight* or the interation strength or the synaptic weight of the connection from j to i.

$A = (a_{ij})_{i,j=1,\cdots,n}$ is the interaction matrix.

$b = (b_i)_{i=1,\cdots,n}$ is the *threshold* vector.

$F = (f_1, \cdots, f_n)$ is called a *deterministic threshold network.*

A *sequential iteration* on F associated to the permutation π of $(1, \cdots, n\}$ is defined by an initial state $x(0) \in \{0,1\}^n$ and the following re-evaluation rule:

$\forall t \geqq 0$, $x(t+1) = F_\pi[x(t)]$ with F_π defined by:

$x_{i(1)}(t+1) = f_{i(1)}[x(t)]$,

$x_{i(2)}(t+1) = f_{i(2)}[y^2]$ where $y_j^2 = x_j(t)$ $\quad \forall j \neq i(1)$,

$y_{i(1)}^2 = x_{i(1)}(t+1)$,

$\cdots$

$x_{i(k)}(t+1) = f_{i(k)}[y^k]$ where $y_j^k = x_j(t)$ $\quad \forall j \notin \{i(1), \cdots, i(k-1)\}$,

$y_{i(j)}^k = x_{i(j)}(t+1)$ $\quad \forall j(1, \cdots, k-1)$,

$\cdots$

$x_{i(n)}(t+1) = f_{i(n)}[y^n]$ where $y_j^n = x_j(t+1)$ $\quad \forall j \neq i(n)$,

$y_{i(n)}^n = x_{i(n)}(t)$

where $(i(1), \cdots, i(n)) = (\pi^{-1}(1), \cdots, \pi^{-1}(n))$ and $x(t)$ is the state of the network at time t.

In a sequential iteration, elements change state one at a time in the order prescribed by the permutation π.

Let $(i_t)_{t \geqq 0}$ be a random process with values in $\{1, \cdots, n\}$: we call it a *visiting strategy* of the set $\{1, \cdots, n\}$.

A *random iteration* on F with strategy $(i_t)_{t \geqq 0}$ is defined by an initial state $x(0)$ and the following re-evaluation rule: for all $t \geqq 0$, $x(t+1) = F_{(i_t)}[x(t)]$ with $F_{(i_t)}$ defined by:

$$\forall t \geqq 0, \quad \forall j \neq i_t, \quad x_j(t+1) = x_j(t),$$
$$x_{i_t}(t+1) = f_{i_t}[x(t)].$$

In a random iteration, elements change state one at a time in the order prescribed by the random process $(i_t)_{t \geqq 0}$. Of course, any sequential iteration can be viewed as a *trivial* random iteration: i.e. one with a periodic, of period n, visiting strategy $(i_t)_{t \geqq 0}$. Hence, on a formal ground, any result standing for random iterations will be true for sequential iterations as well.

Random iterations correspond to a particular choice of the visiting strategy of chaotic iterations [15]; they have been extensively used for the spin glass problem in the Monte Carlo updating rules [6].

The *energy* of a given state x is defined by:

$$E(x) = -\frac{1}{2}\sum_i x_i \sum_{j \neq i} a_{ij}x_j + \sum_i (b_i - a_{ii})x_i.$$

The quadratic term in E has been introduced in [6] in the particular case where $a_{ii} = b_i = 0$ and the linear term in [1] for the general case.

We proved there [1] that:

THEOREM 1. *If $A = (a_{ij})$ is a symmetric matrix with nonnegative diagonal elements, then any sequential iteration on F may only have fixed points.*

Proof. Let the sequential iteration be decomposed at time t into n successive steps, where at time $t + k/n$, $k = 1, \cdots, n$, element k only changes states. Then:

$$E[x(t+1)] - E[x(t)] = \sum_{k=1}^{n} \left(E\left[x\left(t+\frac{k}{n}\right)\right] - E\left[x\left(t+\frac{k-1}{n}\right)\right]\right)$$

and we proved that [1]:

$$x\left(t+\frac{k}{n}\right) \neq x\left(t+\frac{k-1}{n}\right) \Rightarrow E\left[x\left(t+\frac{k}{n}\right)\right] - E\left[x\left(t+\frac{k-1}{n}\right)\right] < 0.$$

COROLLARY 1. *Let $(i_t)_{t\geqq 0}$ be a visiting strategy of $\{1, \cdots, n\}$. Then under the assumptions of Theorem* 1, *the random iteration on F with strategy $(i_t)_{t\geqq 0}$ only has fixed points.*

Proof. The proof of Theorem 1 shows that along the random iteration on F, the energy can only decrease, thus ultimately leading to a fixed point.

Remark. If the random process does not visit sufficiently often (see below) certain elements in $[1, \cdots, n]$, it may happen that the fixed point of the random iteration is not a fixed point of the sequential iteration, i.e. not a fixed point of mapping F (the fixed points of the sequential iterations on F are exactly the fixed points of the mapping F, and thus of the parallel iteration on F). Moreover, such parasites can also be observed if the random iteration gets stabilized into a state for a long time and it switches to a different state as the random process visits an element it had avoided until then, should enough time be allowed. Thus, because of the halting problem, random iterations are not easily handled in practical simulations. In the following, we will thus report on simulations made for sequential iterations. The theoretical results will stand for random iterations as well, provided their visiting strategy is "well behaved": for example, visit any element at least once per time period of length T, for some given $T > 0$ (such iterations will be called *regular*).

We will now address the following problem:

> Given a set of vectors $X_S = \{x^s\}_{s\in S}$, is it possible to find a network F of threshold elements such that any regular random iteration on F will have the x^s's as fixed points?

Such a network will be called an *X_S-memory*. An interesting X_S-memory would be one such that, starting from any initial state $x(0)$ sufficiently close to $x^s \in X_S$, the random iteration would end up into x^s: this would allow partial information on x^s, as contained in $x(0)$, to retrieve x^s, thus implementing an associative memory or a content-addressable memory by making the x^s into addresses.

Thus we call *associative X_S-memory* any X_S-memory F such that:

$$\forall s \in S, \quad \exists V_s \subset \{0,1\}^n : x^s \in V_s \quad \text{and} \quad \forall x(0) \in V_s, \quad \exists T \geqq 0, \quad \forall t \geqq 0, \quad F_{(i_t)}^{T+t}[x(0)] = x^s$$

where $F_{(i_t)}$ is the random iteration on F associated to $(i_t)_{t\geqq 0}$ and V_s is a discrete neighborhood of x^s (see [16]).

Thus any X_S-memory such that all limit points are attractive in some neighborhood is an associative X_S-memory.

We will provide conditions in the following for X_S-memory (§ 3) and for associative X_S-memory (§ 4).

3. X_S-memory. In his paper [6], based on previous work by Little [11], J. Hopfield gives conditions by which he claims he can implement an associative X_S-memory, provided S is not too large:

L-H conditions.

1. Randomness of x^s:

$$\forall s \in S, \quad x^s \text{ has equidistributed 0's and 1's.}$$

2. Pseudo-orthogonality:

$$\forall s, \sigma \in S, \quad s \neq \sigma, \quad \sum_j x_j^\sigma (2x_j^s - 1) \textit{ has mean } 0.$$

3. $\forall i (1, \cdots, n)$, $b_i = 0$.
4. $\forall i, j (1, \cdots, n)$, $i \neq j$, $a_{ij} = \sum_{s \in S} (2x_i^s - 1)(2x_j^s - 1)$, $a_{ii} = 0$.

We will discuss these conditions:

First, deterministic devices, as used in this model, are very well suited for implementation on computers. Probabilistic elements [5], [14] may be used in some instances to prevent the process from staying in local minima, but then the dynamical behavior of the network can be erratic, eventually chaotic; this case corresponds to a nonzero temperature in the spin glass model (see Fig. 1).

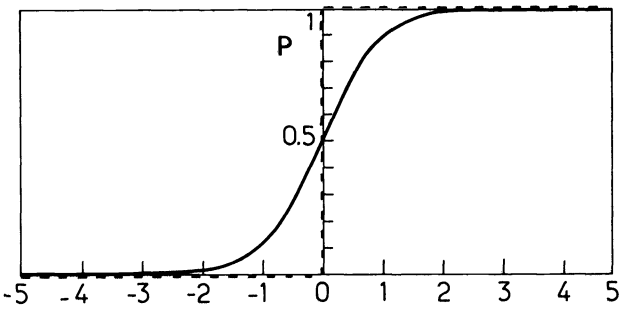


FIG. 1. *Probabilistic and deterministic threshold elements.*

The figure shows the probability that the element i will switch into state 1 when $\sum_{j \neq i} a_{ij} x_j$ *varies.*

The probabilistic threshold device is shown in heavy line, the deterministic one in dashed line.

The probabilistic threshold device is defined by the transition probability rule $P(x_i = 1) = [1 + e^{-\Delta E/T}]^{-1}$ *with* $\Delta E = \sum_{j \neq i} a_{ij} x_j$ *and* T *is the temperature.*

The figure shows the case $T = 1$ *and the deterministic element would thus correspond to the probabilistic case at temperature* $T = 0$.

Randomness of the x^s's is a relevant assumption when n is large: retrieving patterns with very few 0's or 1's could be made through very simple (low complexity) techniques: for example, build a look-up table with the number of 0's and search through it. However, a general purpose memory should also be able to handle the case when the x^s's are nonrandom and correlated. For example, in [9], the patterns x^s are the letters of the alphabet which clearly violate both *L-H* conditions 1 and 2. This will most certainly be the case in practical situations where the information to be stored is meaningful within a given framework and thus neither random nor orthogonal.

L-H condition 4 is based on the so-called *Hebbian* reinforcement principle, which was first introduced by the neuropsychologist Hebb: when two elements are in the

same state (both active or both inactive) in a given pattern, then their connection strength should be increased. In addition, this condition allows for easy updating to fit additional incoming information: add a new x^u to the existing set X_S just requires the updating of the a_{ij} by the following rule:

$$a_{ij} := a_{ij} + (2x_i^u - 1)(2x_j^u - 1).$$

Condition 4 will be referred to in the following as the *Hebb-rule.*

Remark. It is well known [10] that, when the x^s's are linearly independent, orthogonal vectors in $\mathbb{R}^n$, then the connection matrix $A = XX^+$ is just XX^t, which is almost exactly *L-H* condition 4 (except for the zero diagonal). Hopfield's learning process can thus be viewed as a particular case of Kohonen's, but the *retrieval* process is different: Hopfield uses threshold elements, where Kohonen uses linear elements (note that Kohonen also makes use of some thresholding to process discrete patterns, for example patterns in a discrete space representing a limited number of gray levels [10]).

LEMMA 1. *Let $F: \{0, 1\}^n \to \{0, 1\}^n$ be a deterministic threshold network with interaction matrix A and threshold b.*

Let $X_S = \{x^s\}_{s \in S}$ be a set of patterns in $\{0, 1\}^n$.

If A and b satisfy the following assumption:

$$\text{Max}\left\{\sum_{j \neq i} a_{ij}x_j^s \,\middle|\, s \in S: x_i^s = 0\right\} < b_i \leqq \text{Min}\left\{\sum_{j \neq i} a_{ij}x_j^s \,\middle|\, s \in S: x_i^s = 1\right\} \quad \forall i(1, \cdots, n).$$

Then all x^s, $s \in S$, are fixed points of any random iteration on F and F is thus an X_S-memory.

Proof. If i is changing state at time t then:

$$x_i(t+1) = 1 \Rightarrow \sum_{j \neq i} a_{ij}x_j(t) - b_i \geqq 0;$$

thus x^s is stable iff:

$$x_i^s = 1 \Rightarrow b_i \leqq \sum_{j \neq i} a_{ij}x_j^s,$$

$$x_i^s = 0 \Rightarrow b_i > \sum_{j \neq i} a_{ij}x_j^s. \qquad \square$$

Lemma 1 thus provides an algorithm by which a network F may be gradually adapted so as to fit an increasing number of states x^s, by modifying the thresholds.

In our simulations, we used the following algorithm:

Compute the bounds $b_{\max}(i)$ and $b_{\min}(i)$ of the condition in Lemma 1.
If $b_{\max}(i) \leqq 0 \leqq b_{\min}(i)$ *then* $b_i := 0$
else if $b_{\max}(i) < 0$ *then* $b_i := b_{\min}(i)$ *else* $b_i := b_{\max}(i) + 1$.

Note that this algorithm, unlike *L-H* condition 4, requires the full knowledge of the x^s: the thresholds b_i cannot be iteratively computed as can be the a_{ij}, but all computations are *local*, since only knowledge from neighbors is needed for each element *i*.

Because of our algorithm, networks F with few patterns memorized in A will have their threshold b equal to 0, thus satisfying *L-H* condition 3. But progressively adding more patterns will eventually force the choice of a nonzero b_i and finally end up in an infeasible set of conditions: hence the number of states x^s that can be memorized—the *storing capacity*—is bounded. This bound clearly depends on n, the size of the network, on the choice of A (by *L-H* condition 4 or otherwise), of b (by

our algorithm or otherwise) and on the patterns x^s. In particular, it might be possible to find an optimal threshold.

In Fig. 2, we present the results of some simulations which were run for different choices of A, nonrandom patterns (horizontal bars) and different values of n. In Fig. 3, we show the results for A chosen along L-H conditions, our algorithm to compute b and random patterns.

Comparing the two figures shows that the capacity of the network is increased if:

—L-H conditions are used instead of "local" L-H conditions ($a_{ij}=0, \forall j$, except for some neighbors j of i) or a non-Hebbian rule: $a_{ij}=\sum_s x_i^s+x_j^s$.

—The size of the network is increased.

—The memorized patterns are random and noncorrelated instead of nonrandom (we tested the case of horizontal bars, of random length and height in a square network).

In the following, we will mainly be interested in the Hebb rule, since it approximates the auto-associative matrix XX^+, known to be optimal [10].

We then have:

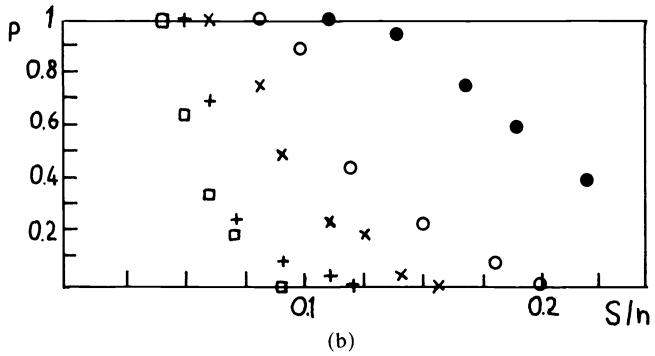
$$a_{ij}=\sum_{s\in S}(2x_i^s-1)(2x_j^s-1)$$

$$\Rightarrow \sum_{j\neq i} a_{ij}x_j^s=\sum_{j\neq i}\sum_{\sigma\in S}(2x_i^\sigma-1)(2x_j^\sigma-1)x_j^s$$

$$=\sum_{j\neq i}x_j^s(2x_i^s-1)(2x_j^s-1)+\sum_{j\neq i}x_j^s\sum_{\sigma\neq s}(2x_i^\sigma-1)(2x_j^\sigma-1)$$

$$=(2x_i^s-1)\sum_{j\neq i}x_j^s+\sum_{j\neq i}x_j^s\sum_{\sigma\neq s}(2x_i^\sigma-1)(2x_j^\sigma-1).$$

(a)



(b)

FIG. 2. *For 20 different initial patterns: horizontal bars of random length and origin in a square network of size $n'\times n'$ (Fig. 2(a) shows some examples), we have computed the conditions on thresholds of Lemma 1 (conditions (C)) for a_{ij} chosen by Hebb rule. For each set of S sequences, of given length $n'\times n'$, we compute the conditions (C) and the number $n_1(S)$ of sets (out of 20) which give networks which satisfy (C).*

In Fig. 2(b), we plot for different values of n and S/n the probability P for a network to satisfy (C): P is estimated by the frequency $n_1(S)/20$ (which thus gives the probability for a network to be an X_s-memory).

We have also run similar simulations for the same set of patterns to be memorized, but different choices of A.

For A non-Hebbian: $a_{ij}=\sum_{s\in S}x_i^s+x_j^s$, the network was always unable to memorize more than just one pattern (which is always possible under condition (C)).

For A Hebbian but local, i.e. $a_{ij}=\sum_{s\in S}(2x_i^s-1)(2x_j^s-1)$ for j in the Moore neighborhood of i (8 nearest neighbors) and 0 otherwise, the network could memorize just 4 or 5 patterns for n' varying from 6 to 10.

In Fig. 2(b) experimental points corresponding to various values of n' are indicated as follows: ● $n=6\times 6$, ○ $n=7\times 7$, × $n=8\times 8$, + $n=9\times 9$, □ $n=10\times 10$.

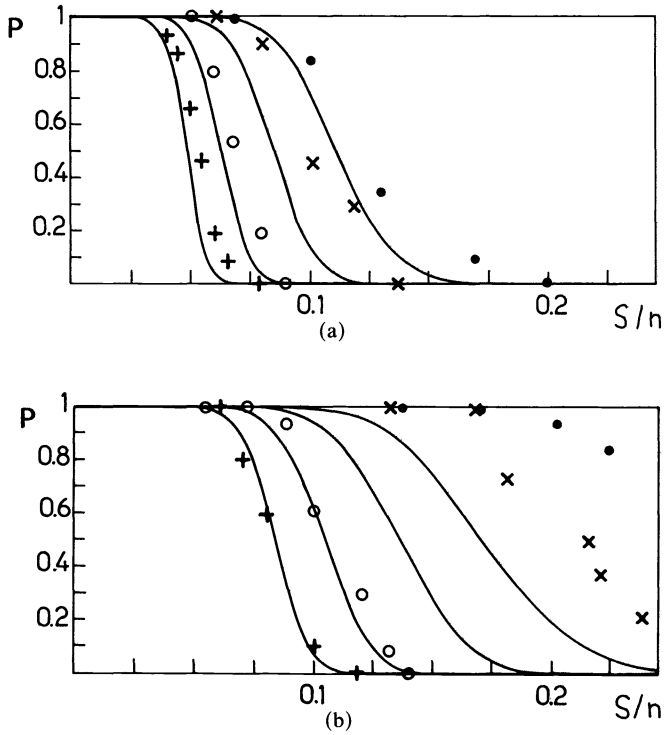


FIG. 3. *Probability for a network to be a X_s-memory.*

As in Fig. 2, we have computed the conditions (C) *for a_{ij} chosen by Hebb rule, for* 20 *different initial sequences set at random. For each set of S sequences, of given length n, we compute the number $n_1(S)$ (resp. $n_0(S)$) of sets (out of* 20) *which give networks which satisfy* (C) *(resp.* (C) *with* 0 *threshold).*

We plot for different values of n and S/n the probability P for a network to satisfy (C): *P is estimated by the frequency $n_0(S)/20$ in Fig.* 3(a) *(which thus gives the probability for a network with* 0 *threshold to be an X_s-memory) and by the frequency $n_1(S)/20$ in Fig.* 3(b).

In Fig. 3(a) *we also plot the theoretical value of P as estimated in the paper:* $P \sim \mathrm{erf}^{nS}([(n-1)/2(S-1)]^{1/2})$ *(in the figure, n increases from right:* $n=30$ *to left:* $n=200$*). There is a discrepancy between experimental and theoretical points, which is due to the simplifying assumption made in the text to derive the theoretical expression for P (independence of the x^s's, approximation of the binomial distribution by a Gaussian). However when n increases, these discrepancies are clearly reduced, showing a good accordance with the experimental data for large n.*

In Fig. 3(b) *we have plotted the theoretical values of P for a nonzero choice of the thresholds:* $b_i = \frac{1}{2}\sum_s \sum_{j\neq i}(2x_i^s-1)(2x_j^s-1)$. *Computations similar to those for $b_i=0$ then lead to* $P \sim \mathrm{erf}^{nS}([(n-1)/(S-1)]^{1/2})$. *There is also good accordance between experimental points (with b_i chosen along our algorithm; see Lemma* 1) *and theoretical values (for the above choice of b_i) for large n. Which proves in addition that the exact choice of b_i within the interval fixed by our conditions* (C) *does not change the shape of the curve.*

Notice that the curves in Fig. 3(b) *are shifted left of those in Fig.* 3(a): *imposing a* 0 *threshold clearly degrades the performances (in terms of the capacity S/n: by about a factor* 2 *at* $P=.5$).

● $n=30$, ○ $n=50$, × $n=100$, + $n=200$.

The condition of Lemma 1 is thus:

$$\mathrm{Max}\left[-\sum_{j\neq i} x_j^s + \sum_{j\neq i} x_j^s \sum_{\sigma\neq s}(2x_i^\sigma-1)(2x_j^\sigma-1)\,\middle|\, s: x_i^s=0\right] < b_i$$

$$\leqq \mathrm{Min}\left[+\sum_{j\neq i} x_j^s + \sum_{j\neq i} x_j^s \sum_{\sigma\neq s}(2x_i^\sigma-1)(2x_j^\sigma-1)\,\middle|\, s: x_i^s=1\right].$$

This condition is clearly satisfied for small values of S (where S denotes the cardinality of set S). Simulations (see Fig. 3) show that when S increases, the probability for this condition to be met falls, more and more sharply when n increases (for random patterns).

Let us denote

$$X_i^s = \sum_{j \neq i} x_j^s, \qquad Z_i^s = \sum_{j \neq i} x_j^s \sum_{\sigma \neq s} (2x_i^s - 1)(2x_j^\sigma - 1).$$

Assuming that the $(x_j^s)_j$, for any $s \in S$, are independent trials from a Bernoulli distribution of parameter $\frac{1}{2}$ and that the x^s and x^σ, $\sigma \neq s$, are independent, we have:

$$EX_i^s = (n-1)/2, \qquad \text{Var } X_i^s = (n-1)/4,$$
$$EZ_i^s = 0, \qquad \text{Var } Z_i^s = (n-1)(S-1)/2.$$

The distribution of Z_i^s is given by:

$$\begin{aligned} P(Z_i^s = z) &= \sum_p P(Z_i^s = z, X_i^s = p) \\ &= \sum_p P(Z_i^s = z / X_i^s = p) P(X_i^s = p) \\ &= \sum_p \binom{p}{[z + p(S-1)]/2} 2^{-p} \binom{p}{n-1} 2^{-(n-1)}. \end{aligned}$$

By using the approximation of the binomial distribution by a normal distribution, we then have, for n large enough:

$$P(Z_i^s = z) \sim [\pi(n-1)(S-1)]^{-1/2} \exp[-Z^2/(n-1)(S-1)].$$

The probability that 0 is a threshold is then:

$$\begin{aligned} P(b_i = 0) &= P \\ &= P(\text{Max}\{-X_i^s + Z_i^s \mid s \in S: x_i^s = 0\} < 0 \leqq \text{Min}\{X_i^s + Z_i^s \mid s \in S: x_i^s = 1\}) \\ &= P(\forall s \in S: x_i^s = 0, \forall \sigma \in S: x_i^\sigma = 1, -X_i^s + Z_i^s < 0 \leqq X_i^\sigma + Z_i^\sigma). \end{aligned}$$

Assuming that variables Z_i^s and Z_i^σ are independent (which is not exactly true in general), we then have:

$$\begin{aligned} P(b_i = 0) &= P(-X_i^s + Z_i^s < 0, \forall s \in S: x_i^s = 0) P(X_i^\sigma + Z_i^\sigma \geqq 0, \forall \sigma \in S: x_i^\sigma = 1) \\ &= P(-X_i^s + Z_i^s < 0)^{S_0} P(X_i^\sigma + Z_i^\sigma \geqq 0)^{S - S_0} \\ \Rightarrow P(b_i = 0) &\sim \text{erf}^S(\sqrt{(n-1)/2(S-1)}). \end{aligned}$$

Hence, the probability that all conditions on thresholds are met is:

$$P = \text{erf}^{nS}(\sqrt{(n-1)/2(S-1)}) \sim 1 - [nS/\sqrt{2\pi(n-1)/2(S-1)}] \exp[-(n-1)/4(S-1)].$$

Denoting $\beta = n/S$, we finally get:

$$\beta + 6 \text{ Log } \beta = 8 \text{ Log } n - 4 \text{ Log } [(1-P)\sqrt{\pi}].$$

Hence, S/n varies as $1/8 \text{ Log } n$ (see Fig. 3) for any given P.

Moreover, if $P_1 = .9$ and $P_2 = .1$ then

$$\beta_1 - \beta_2 = 4 \text{ Log } 9 \sim 4 \text{ Log } 10$$

which is independent of n and

$$\frac{S_2}{n} - \frac{S_1}{n} = -\frac{[\beta_1 - \beta_2]}{\beta_1 \beta_2} \sim \frac{4 \text{ Log } 10}{[8 \text{ Log } n]^2}$$

which shows, as was indeed found in the simulations, that the curves $P(S/n)$ fall down more and more sharply as n increases.

The scaling law we have provided fits relatively well our simulation results for random patterns: large values of n are necessary for good accuracy.

4. Associative X_S-memory. In the previous section, we have shown how it was possible to build an X_S-memory for a given set $\{x^s\}_{s\in S}$ of patterns. We now provide a sufficient condition to make it an associative memory (it is not a necessary condition).

We begin by some definitions and lemmas. The interested reader could refer to [16] where some of the tools used here were introduced for parallel iterations of Boolean networks.

Let $x\in\{0,1\}^n$, the *first neighborhood* of x is:

$$V_1(x)=\{x\}\cup C_1(x)$$

where $C_1(x)$ is the 1-*sphere* of x defined by:

$$C_1(x)=\{x_i^*=(x_1,\cdots,x_{i-1},\bar{x}_i,x_{i+1},\cdots,x_n),\ i=1,\cdots,n\}\cup\{x\}.$$

If x is a fixed point of a random iteration on F, with strategy $(i_t)_{t\geqq 0}$, then x is said to be *attractive* in its first neighborhood for $F_{(i_t)}$ iff:

- for all $y\in V_1(x)$, $F_{(i_t)}(y)\in V_1(x)$,
- for all $y\in V_1(x)$, $\exists r\geqq 0$: $F^r_{(i_t)}(y)=x$.

In [16], F. Robert gave necessary and sufficient conditions on the discrete derivative of a Boolean network F at a given fixed point x which ensured that for the parallel iteration on F, x should be attractive in its first neighborhood.

We will use the energy [1], [3] of a threshold network to provide a sufficient condition for the attractivity of a fixed point in its neighborhood, for sequential and random iterations.

Letting

$$E(x)=-\tfrac{1}{2}\sum_i\sum_{j\neq i}a_{ij}x_ix_j+\sum_i b_ix_i$$

then define the gradient of E as:

$$\operatorname{grad}E(x)=\left(-\sum_{j\neq i}a_{ij}x_j+b_i\right)_i$$

(this is the "continuous" gradient, different from the discrete gradient [16]).

It is easy to prove that:

$$\begin{aligned}E(x)-E(x_i^*)&=\operatorname{grad}E(x)\cdot(x-x_i^*)\\&=\operatorname{grad}_iE(x)\cdot(x_i-\bar{x}_i).\end{aligned}$$

Then, x is a fixed point if and only if

$$\begin{aligned}&x_i=1\Rightarrow\operatorname{grad}_iE(x)\leqq 0,\\&x_i=0\Rightarrow\operatorname{grad}_iE(x)>0\\&\qquad\Rightarrow E(x)-E(x_i^*)\leqq 0,\end{aligned}$$

which proves:

LEMMA 2. *Let x be a fixed point for the random iteration on a threshold network F; then x is a local minimum of the energy, in its first neighborhood.*

LEMMA 3. *Let F be a threshold network; then*

$$\operatorname{grad}_jE(x_i^*)=\operatorname{grad}_jE(x)+a_{ji}(x_i-\bar{x}_i)\quad\forall x\in\{0,1\}^n,\quad\forall i,j(1,\cdots,n).$$

Proof.

$$\begin{aligned}\operatorname{grad}_j E(x_i^*) &= -\sum_{k\neq j} a_{jk}(x_i^*)_k + b_j \\ &= -\sum_{k\neq i,j} a_{jk}x_k - a_{ji}\bar{x}_i + b_j \\ &= -\sum_{k\neq j} a_{jk}x_k + b_j + a_{ji}(x_i - \bar{x}_i).\end{aligned}$$ □

(This is very analogous to the classical theory of nonnegative definite quadratic forms. But here, in the discrete case, there may be several local minima.)

LEMMA 4. *Let F be a threshold network and $(i_t)_{t\geqq 0}$ be a random strategy. If:*

$$\operatorname{grad}_j E(x_i^*)\cdot(x_j - \bar{x}_j) \leqq 0 \quad \forall i,j(1,\cdots,m),$$

then $V_1(x)$ is stable under the random iteration on F of strategy $(i_t)_{t\geqq 0}$.

Proof.

• Let $x_i^* \in V_1(x)$; suppose that $i_t = j \neq i$.

Let us denote $C_2(x)$ the 2-sphere of x defined by:

$$C_2(x) = \{x_{ij}^* = (x_1,\cdots,\bar{x}_i,\cdots,\bar{x}_j,\cdots,x_n),\ i,j = 1,\cdots,n\}$$

and $V_2(x)$ the second neighborhood of x by: $V_2(x) = \{x\} \cup C_2(x) \cup C_1(x)$.

Then, either $F_{(i_t)}(x_i^*) = x_i^*$ or $F_{(i_t)}(x_i^*) = x_{ij}^*$.

Suppose $F_{(i_t)}(x_i^*) = x_{ij}^*$. As $E(x_i^*) - E(x_{ij}^*) = \operatorname{grad}_j E(x_i^*)\cdot(x_j - \bar{x}_j)$ and E is strictly decreasing along the random iteration, one thus has $E(x_i^*) - E(x_{ij}^*) > 0$, which is contradictory to our assumption. Hence $F_{(i_t)}(x_i^*) = x_i^* \Rightarrow F_{(i_t)}(x_i^*) \in V_1(x)$.

• If $i_t = i$, either $F_{(i_t)}(x_i^*) = x_i^*$ or $F_{(i_t)}(x_i^*) = x$ and in both cases $F_{(i_t)}(x_i^*) \in V_1(x)$. □

THEOREM 2. *Let $F = (A, b)$ be a deterministic threshold network and x be a fixed point of a random iteration on F of strategy $(i_t)_{t\geqq 0}$, visiting any element at least once per time period of length T.*

If $V_1(x)$ is stable under the random iteration, then x is attractive in $V_1(x)$.

Proof. Letting $x_i^* \in V_1(x)$, we have to show that:

$$\exists r \geqq 0 \colon F_{(i_t)}^r(x_i^*) = x.$$

Suppose $i_t = j$.

• If $j = i$, then either $F_{(i_t)}(x_i^*) = x$, which ends the proof,

$$\text{or} \quad F_{(i_t)}(x_i^*) = x_i^* \Rightarrow \sum_{j\neq i} a_{ij}x_j - b_i \geqq 0 \quad \text{if } \bar{x}_i = 1,$$

$$\sum_{j\neq i} a_{ij}x_j - b_i < 0 \quad \text{if } \bar{x}_i = 0.$$

As x is a fixed point, we have:

$$\sum_{j\neq i} a_{ij}x_j - b_i \geqq 0 \quad \text{if } x_i = 1,$$

$$\sum_{j\neq i} a_{ij}x_j - b_i < 0 \quad \text{if } x_i = 0,$$

which is contradictory. Hence:

$$(\exists t \colon i_t = i) \Rightarrow F_{(i_t)}(x_i^*) = x.$$

• If $j \neq i$, then either $F_{(i_t)}(x_i^*) = x_i^*$ or $F_{(i_t)}(x_i^*) = x_{ij}^*$, which is impossible since $V_1(x)$ is stable. Hence, for any sequential iteration associated to the permutation π, $F_\pi(x_i^*) = x$ and for any random iteration which visits each element at least once per time period of length T: $F_{(i_t)}^T(x_i^*) = x$, which ends the proof. □

Remark. The result we have just proved makes a sharp difference between parallel and sequential iterations of *threshold networks.*

In parallel iterations, the stability of $V_1(x)$ alone does not ensure the attractivity of x (see [16]), and, in addition, when x is attractive, the iteration may need several steps to reach x (see Fig. 4). In sequential iterations, if $V_1(x)$ is stable, then x is attractive and attracts any point in its first neighborhood in just one step, furthermore $V_1(x)$ is also stable and x attractive in one step for the parallel iteration.

THEOREM 3. *Let F be a threshold network, $(i_t)_{t\geq 0}$ be a random strategy visiting each element in $\{1, \cdots, n\}$ at least once per time period of length T.*

Let $X_S = \{x^s\}_{s\in S}$ be a set of patterns.

If A is a symmetric matrix with nonnegative diagonal elements, if A, b and X_S satisfy the following conditions:

$$\forall i, j(1, \cdots, n),$$

$$\operatorname{Max}\left\{\sum_{j\neq i} a_{ij}x_j^s \,\middle|\, s\in S: x_i^s = 0\right\} < b_i \leqq \operatorname{Min}\left\{\sum_{j\neq i} a_{ij}x_j^s \,\middle|\, s\in S: x_i^s = 1\right\}$$

$$[\operatorname{grad}_j E(x^s) + a_{ij}(x_i^s - \bar{x}_i^s)][x_j^s - \bar{x}_j^s] \leqq 0 \quad \forall s \in S,$$

then F is an associative X_S-memory.

Proof. Obvious from Lemma 1, Theorem 2 and Lemma 4. □

Depending on the rules to compute the a_{ij}, this condition may be more or less easily met.

Remarks. 1.

$$[\operatorname{grad}_j E(x^s) + a_{ij}(x_i^s - \bar{x}_i^s)][x_j^s - \bar{x}_j^s] = \left[-\sum_{k\neq j} a_{kj}x_k^s + b_j + a_{ij}(x_i^s - \bar{x}_i^s)\right][x_j^s - \bar{x}_j^s].$$

Hence the condition in Theorem 3 is more easily satisfied, for a given $s \in S$, when b_j is small (i.e. $b_j \to \operatorname{Max}[\sum a_{kj}x_k^s / s\in S: x_j^s = 0]$) for $x_j^s = 1$ or b_j is large (i.e. $b_j = \operatorname{Min}[\sum a_{kj}x_k^s / s\in S: x_j^s = 1]$) for $x_j^s = 0$.

These constraints on b_j may of course be contradictory for different $s \in S$, but we see that acting on the thresholds may help improve the attractivity of a pattern.

In our simulations (Fig. 5), we always used $b_i = 0$ when 0 was feasible, otherwise we took b_i as defined by the algorithm given previously.

2. The conditions in Theorem 3 not only ensure that all x^s, $s \in S$, are attractive in their first neighborhood:

$$\forall x \in V_1(x^s), \quad \exists T_x: \forall t > 0, \qquad F_{(i_t)}^{T_x+t}(x) = x^s,$$

but also that the trajectory never *leaves* this neighborhood. We thus have here a condition for attractivity in the first neighborhood in the sense of [16].

3. Simulations (Fig. 5) show that, for X_s-associative memory networks, all patterns x^s were usually attractive in neighborhoods much larger than $V_1(x^s)$. This is indeed of great importance for practical applications: one would like to be able to retrieve a given pattern x^s from any pattern y differing from x^s by a few bits, for example by 10% of its n bits.

4. Simulations show that such networks cannot deal with high-order predicates such as translation invariance for example: this is the main limitation of all systems of the perceptron family (see Fig. 6(c)).

We now give a necessary condition which ensures the attractivity in a dth neighborhood for $d > 1$.

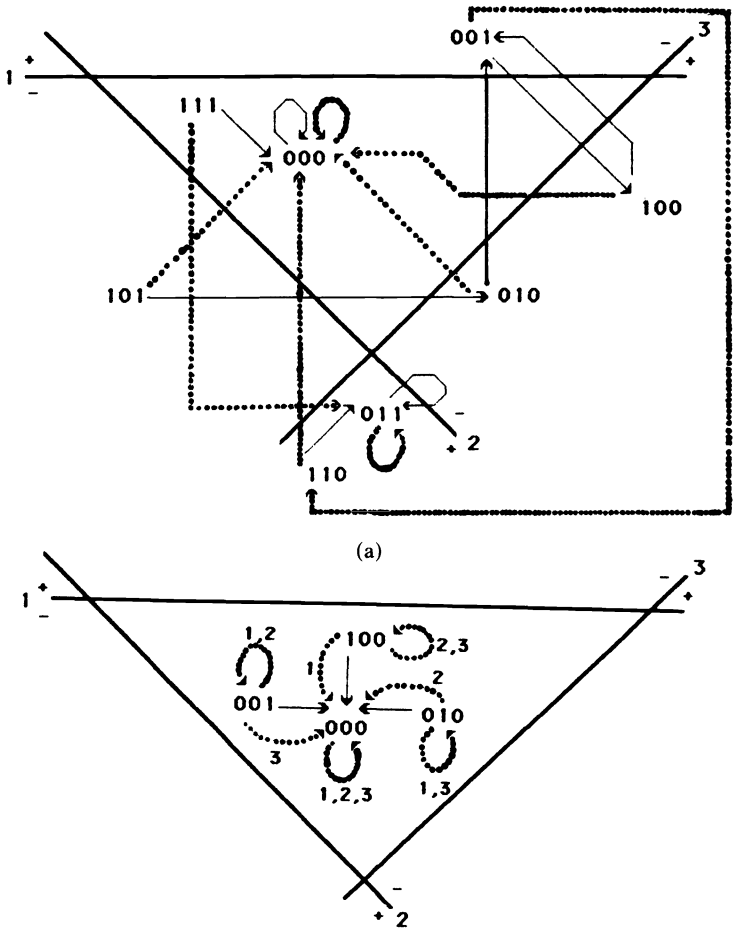


FIG. 4. *Comparing parallel and sequential iterations.*

The figure represents a network of $n = 3$ *threshold elements. For each element* i, *the hyperplane* $\sum_{j \neq i} a_{ij}x_j - b_i = 0$ *is represented: it separates the space into two half spaces. In one half space the above sum is positive (this space is marked* + *on the figure), in the other the sum is negative (which is marked* − *on the figure). For example in Fig.* 4(a), 001 *lies in a region where* $\sum_{j \neq i} a_{1j}x_j - b_1 \geqq 0$, $\sum_{j \neq i} a_{2j}x_j - b_2 < 0$, $\sum_{j \neq i} a_{3j}x_j - b_3 < 0$.

In Fig. 4(a) *the parallel iteration graph is drawn in solid lines:* 000 *is a fixed point with a stable first neighborhood, but it is not attractive in this neighborhood, since states* 001, 100 *and* 010 *(in this first neighborhood) are not attracted towards* 000 *but lead to the limit cycle* $001 \leftrightarrow 100$.

000 *is also a fixed point for the sequential iterations, but its first neighborhood is not stable; hence* 000 *is not attractive either (the iteration graph for the sequential iteration with identical permutation is shown in dashed lines).*

In Fig. 4(b) *we show a different network where* 000 *is attractive in its first neighborhood both for the parallel (solid line) and the sequential (dashed) iterations and in just one step. As was stated in the paper, this occurs because* 000 *has a first neighborhood stable under the sequential iteration. For the sequential iteration, we show the result of changing one element indicated by its number on the arrow: for example, when element* 1 *changes state,* $100 \to 000$, *when elements* 2 *or* 3 *are changed,* 100 *remains fixed. This information allows one to easily derive the iteration graph for any sequential iteration. For example in the sequential iteration associated to permutation* 3, 1, 2 *element* $100 \to 000$ *(because* $100 \to_3 100 \to_1 000 \to_2 000$).

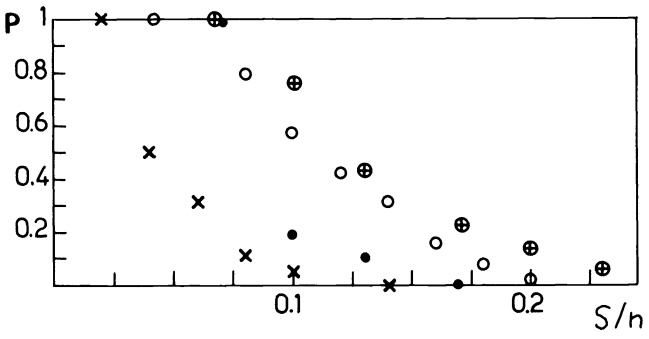


FIG. 5. *Probability for a network to be an associative X_S-memory.*

As in Fig. 3, for 20 different initial sequences set at random, we have computed, for each set S of sequences, the number $N_1(S)$ of sets of S sequences—out of 20—for which all S sequences were attractive in their first neighborhood. $N_0(S)$ is the same number in the case where the network is further constrained to have 0 threshold.

The figure shows the frequencies $N_1(S)/20$ and $N_0(S)/20$.

Note that, in terms of attractivity, allowing for nonzero threshold increases the performances: the experimental points for $N_0(S)/20$ are shifted left of those for $N_1(S)/20$.

0 threshold: ● $n=30$, × $n=50$. *Non 0 threshold*: ⊕ $n=30$, ○ $n=50$.

Let us denote

$$C_d(x)=\{y\in(0,1)\}^n/\exists i_1,\cdots,i_d(1,\cdots,n)\colon y=(x_1,\cdots,\bar{x}_{i_1},\cdots,\bar{x}_{i_d},\cdots,x_n)\}$$

and $V_d(x)$ the dth neighborhood defined by:

$$V_d(x)=\{x\}\cup C_1(x)\cup C_2(x)\cup\cdots\cup C_d(x).$$

COROLLARY 2. *Let F be a threshold network; then*

$$\forall x\in\{0,1\}^n,\quad \forall y\in C_d(x),\quad \forall j(1,\cdots,n),$$

$$E(y_j^*)-E(y)=-[\operatorname{grad}_j E(x)+a_{ji_1}(x_{i_1}-\bar{x}_{i_1})+\cdots+a_{ji_d}(x_{i_d}-\bar{x}_{i_d})][x_j-\bar{x}_j].$$

Proof. Immediate from Lemmas 2 and 3. □

Corollary 2 shows that the conditions in Theorem 3 will usually ensure the stability of some dth neighborhood of x^s, with $d>1$:

$$\text{Theorem 3}\Rightarrow\operatorname{grad}_j E(x^s)(x_j^s-\bar{x}_j^s)+a_{ij}(x_i^s-\bar{x}_i^s)(x_j^s-\bar{x}_j^s)\leqq 0$$

but since x^s is a fixed point:

$$\operatorname{grad}_j E(x^s)(x_j^s-\bar{x}_j^s)<0;$$

hence Theorem 3 implies that $a_{ij}(x_i^s-\bar{x}_i^s)(x_j^s-\bar{x}_j^s)$ is small with respect to $\operatorname{grad}_j E(x^s)(x_j^s-\bar{x}_j^s)$.

If d is small enough, then $\sum_{k=1}^{d} a_{ji_k}(x_{i_k}^s-\bar{x}_{i_k}^s)(x_j^s-\bar{x}_j^s)$ will also remain small with respect to $\operatorname{grad}_j E(x^s)(x_j^s-\bar{x}_j^s)$.

This will thus imply that $V_d(x^s)$ is stable under the random iteration on F. But x^s may still be nonattractive in $V_d(x^s)$: it may happen that $y\in V_d(x^s)$ is a fixed point, whereas $x_i^*\in C_1(x^s)$ could not (see proof of Theorem 2). In this case, the iteration cannot leave the dth sphere of x.

Simulations were run and show that associative X_S-memory networks built along Theorem 3 to memorize nonrandom patterns—such as letters—were usually attractive

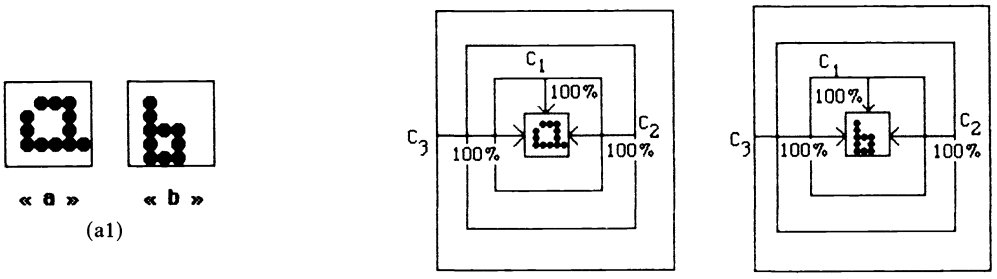


FIG. 6. *Associative X_S-memory for nonrandom patterns: letters of the alphabet.*

The figure shows, for two different sizes of the network, the structure of attractivity in the neighborhoods of memorized patterns.

In Fig. 6(a), $n=6\times6$ *and* $S=2$. *The memorized patterns represented here are* $\langle\langle a\rangle\rangle$ *and* $\langle\langle b\rangle\rangle$. *Different letters were tested giving similar results.*

For each point at distance d ($1\leqq d\leqq3$) *of one of the memorized patterns, we give its limit point. Simulations show here that all points within distance* $d\leqq3$ *of the memorized patterns are attracted back to these patterns, in usually just one step. This means that it is possible with probability* 1 *to retrieve the memorized patterns from any pattern containing as much as* 14% *of its n bits distorted.*

We did not fully explore larger neighborhoods since the size of a neighborhood at distance d grossly increases as n^d (*there are* 7140 *points in* C_3, *among which we tested only* 4000, *i.e.* 60%). *Nevertheless, it seems that the probability of going back to the memorized pattern falls down very sharply when d increases, but not uniformly in all directions: fault tolerance depends on the spatial position of a bit, some bits are very "sensitive", others rather indifferent.*

In Fig. 6(b), $n=8\times8$ *and* $S=3$. *The memorized patterns are* $\langle\langle X\rangle\rangle$, $\langle\langle b\rangle\rangle$ *and* $\langle\langle K\rangle\rangle$. *We tried different letters which give more or less the same results: however results are much better (for fault tolerance) if the letters are very uncorrelated (such as* $\langle\langle A\rangle\rangle$, $\langle\langle D\rangle\rangle$, $\langle\langle G\rangle\rangle$, $\langle\langle J\rangle\rangle\cdots$).

The patterns we have chosen here are relatively correlated (branches in the $\langle\langle X\rangle\rangle$ *and the* $\langle\langle K\rangle\rangle$, *and vertical bar in the* $\langle\langle b\rangle\rangle$ *and the* $\langle\langle K\rangle\rangle$).

The structure of attractivity is represented as in Fig. 6(a).

For $d=1$, *all patterns within distance* 1 *were retrieved (in one step).*

For $d=2$, *all neighbors at distance d of* $\langle\langle b\rangle\rangle$ *and* $\langle\langle K\rangle\rangle$ *were retrieved, but only* 84% *of the neighbors of* $\langle\langle X\rangle\rangle$.

For $d=3$ (*there are* $n(n-1)(n-2)/6=41\,664$ *points in* C_3), *we tested* 4000 *points of* C_3 (i.e. 10% *of the elements of* C_3). $\langle\langle b\rangle\rangle$ *and* $\langle\langle K\rangle\rangle$ *were retrieved with less than* 1% *failure,* $\langle\langle X\rangle\rangle$ *was lost in about* 40% *of the cases.*

This example has performances which are not as good as those of Fig. 6(a) *contrary to what one would expect from a larger network: this is because of the correlations between memorized patterns that we have already noted.*

Note that the spurious patterns which attract the d-neighbors as soon as d is large are usually made from a superposition of parts of the memorized patterns (or a complement of such).

Finally we show in Fig. 6(c) *that our network cannot achieve translation invariance: a letter* $\langle\langle X\rangle\rangle$ *translated horizontally does not allow one to retrieve the original* $\langle\langle X\rangle\rangle$.

in more than just the first neighborhood: typically, in the 4th or 5th neighborhood for $n=36$. This result has been previously noted in [6].

Furthermore, the attraction was always very fast: usually in just one iteration step, the memorized pattern was retrieved (see Fig. 6).

This example—with letters—shows that such networks may eventually be used to build a character recognition machine. However, nonorthogonality of letters leads to relatively poor performances for such purposes. Linear associative models, such as Kohonen's [10], would do much better, since we know that, when the patterns are not independent nor orthogonal, the generalized inverse X^+ cannot be approximated any more by X^t.

However *L-H* conditions lead to a computational burden much smaller than for exactly computing X^+. It is thus a very attractive model for a recognition machine

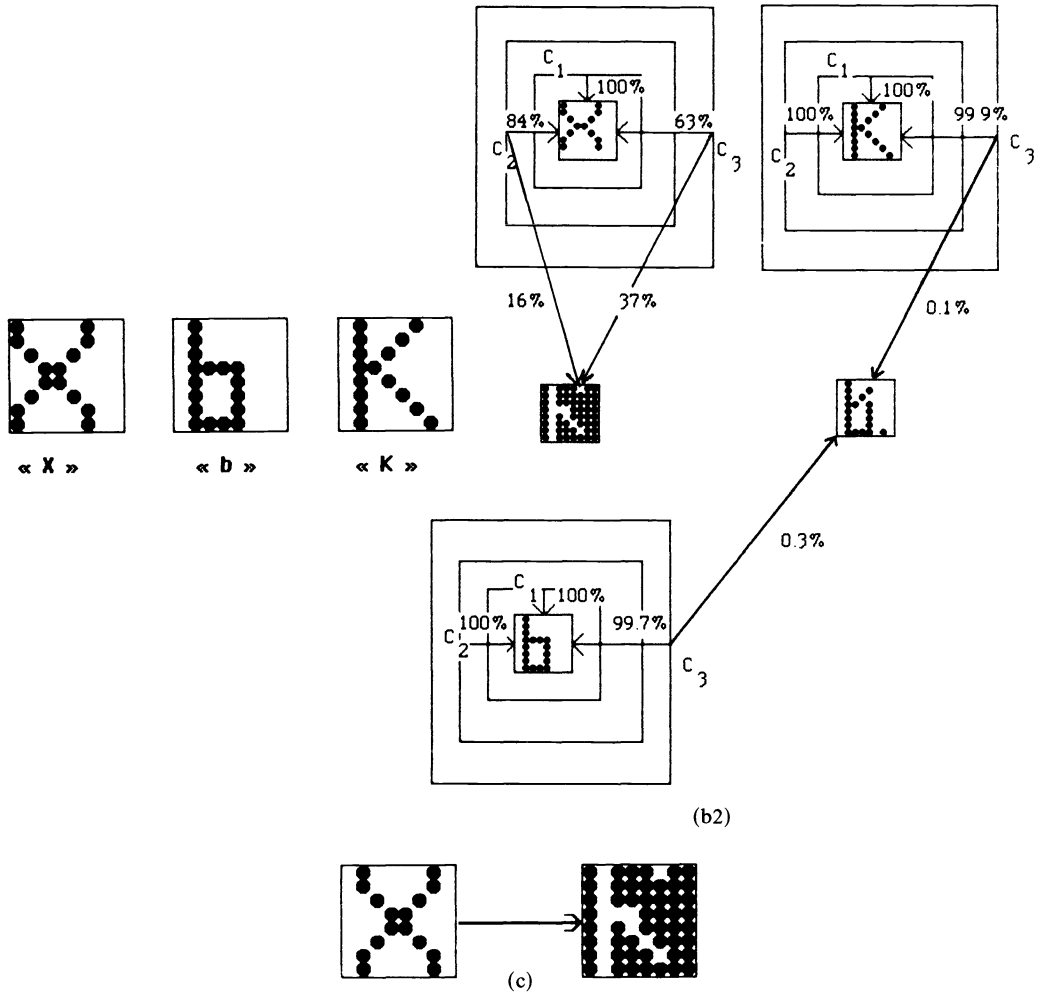


FIG. 6 (cont.)

when the patterns are not too correlated, which could be achieved by preprocessing the data (using a Laplace operator for example): see also [10]. One more nice feature in the model based on L-H conditions compared to linear associators such as Kohonen's is that its retrieval process leads to the *exact* stored pattern: stored patterns are made attractive fixed points and thus there is an exact match between the retrieved pattern and the pattern closest to the noised item. In linear associators (see Kohonen [10] and Hinton and Anderson [4]) the retrieved pattern, being the projection of the noised pattern on the linear space generated by the memorized patterns, is usually not identical to the closest pattern: linear associators achieve only best matching. However, for patterns too far away from a stored item, linear associators still lead to some relatively good matching, where L-H systems lead to *spurious* attractors completely different from the memorized item. Exact characterization of spurious attractors is still an open problem.

5. Conclusions. We have discussed a model for associative memory based on a threshold network. We have proved that by acting on the thresholds it was possible to force a given set X_S of patterns as fixed points of the random iterations of the

network, provided that the number of these patterns was not too large. We have given a scaling law for this capacity in the case where the patterns were random and show that this law seems to fit relatively well for nonrandom patterns also (but with a systematic degradation of performances).

We have given a sufficient condition for the patterns to be attractive in their first neighborhood and proved that this condition was most probably satisfied for random patterns. Simulations have shown that the sufficient condition was usually met in the case of nonrandom patterns such as the letters of the alphabet and that the attractivity of the fixed points usually stood for much larger neighborhoods.

Additional research must be dedicated to the problem of designing an optimal choice of the threshold and of characterizing the exact attractivity of the fixed points: simulations seem to show that the attraction basins are usually (i.e. for nonrandom patterns) nonsymmetric and thus that certain bits are more "meaningful" than others. More work is also needed in order to achieve higher order predicates that this class of networks is definitely unable to handle. Work on Boltzmann machines [5] have shown one way, but this should be further investigated.

Work on local connections have shown that the performances to be expected were not as good as for global connections. Further work in this direction is needed, in order to obtain efficient networks implementable on chips. Such networks could then be well adapted to fast pattern recognition: our rules for computing the a_{ij} and the b_i can be computed locally by the elements of the network and thus the network can be trained to learn a given set of patterns and later on recognize any pattern sufficiently close to one of the learned patterns. This would thus provide an adaptive network for fast pattern recognition.

Note. All simulations presented in this paper were run on a VAX or a PDP 11-23.

Acknowledgments. We have benefited from discussions with B. Huberman and L. Monier at Xerox, and from a very stimulating report from an anonymous referee.

REFERENCES

[1] F. FOGELMAN, E. GOLES AND G. WEISBUCH, *Transient length in sequential iterations of threshold functions*, Discrete Appl. Math., 6 (1983), pp. 95-98.

[2] K. FUKUSHIMA AND S. MIYAKE, *A self organizing neural network with a function of associative memory: Feedback type cognitron*, Biol. Cybern., 28, 4 (1978), pp. 201-208.

[3] E. GOLES CHACC, F. FOGELMAN SOULIE AND D. PELLEGRIN, *Decreasing energy functions as a tool for studying threshold networks*, Discrete Appl. Math., 12 (1985), pp. 261-277.

[4] G. E. HINTON AND J. A. ANDERSON, eds., *Parallel Models of Associative Memory*, Erlbaum, Hillsdale, NJ, 1981.

[5] G. E. HINTON, T. J. SEJNOWSKI AND D. H. ACKLEY, *Boltzmann machines: Constraint satisfaction networks that learn*, Tech. Report CMU-CS 84-119, Carnegie Mellon Univ., Pittsburgh, PA, 1984. Also in Cognitive Science, 9 (1985), pp. 147-169.

[6] J. J. HOPFIELD, *Neural networks and physical systems with emergent collective computational abilities*, Proc. Nat. Acad. Sci. U.S.A., 79 (1982), pp. 2554-2558.

[7] B. A. HUBERMAN AND T. HOGG, *Adaptation and self-repair in parallel computing structures*, Phys. Rev. Lett., 52 (1984), pp. 1048-1051.

[8] T. HOGG AND B. A. HUBERMAN, *Understanding biological computation: Reliable learning and recognition*, Proc. Nat. Acad. Sci. U.S.A., 1984.

[9] D. d'HUMIÈRES AND B. A. HUBERMAN, *Dynamics of self-organization in complex adaptive networks*, J. Statist. Phys., 34 (1984), pp. 361-379.

[10] T. KOHONEN, *Self Organization and Associative Memory*, Springer-Verlag series in Information Sciences, no. 8, 1984.

[11] W. A. LITTLE, *Existence of persistent states in the brain*, Math. Biosci., 19 (1974), pp. 101-120.

[12] C. A. MEAD AND L. A. CONWAY, *Introduction to* VLSI *Systems*, Addison-Wesley, Reading, MA, 1980.

[13] M. MINSKY AND S. PAPERT, *Perceptrons*, MIT Press, Cambridge, 1969.
[14] P. PERETTO, *Properties of neural networks, a statistical physics approach*, Biol. Cybernet., 50 (1984), pp. 51-62.
[15] F. ROBERT, *Contraction en norme vectorielle: convergence d'itérations chaotiques pour des équations non linéaires de point fixe à plusieurs variables*, Linear Algebra Appl., 13 (1976), pp. 19-35.
[16] ———, *Discrete Iterations*, Springer-Verlag, Berlin-New York-Heidelberg, 1986.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987

© 1987 Society for Industrial and Applied Mathematics

THE EQUIVALENCE PROBLEM FOR SINGLE-VALUED TWO-WAY TRANSDUCERS (ON NPDTOL LANGUAGES) IS DECIDABLE*

KAREL CULIK II† AND JUHANI KARHUMÄKI‡

Abstract. The equivalence problem for deterministic two-way transducers was a longstanding problem finally shown decidable by E. M. Gurari. We show a generalization of this result, namely that it is decidable whether two single-valued two-way transducers are equivalent on an NPDTOL language. We obtain this result by showing that it is even decidable whether a two-way transducer is single-valued on an NPDTOL language. Our solution is based on the compactness property of the systems of equations over a finitely generated free monoid shown recently by J. Lawrence and M. Albert and also V. S. Guba. This was a longstanding open problem known as the Ehrenfeucht Conjecture. Our approach also shows that for one-way transducers their single-valuedness on a given HDTOL language can be tested, and thus the equivalence of two single-valued one-way transducers on an HDTOL language is decidable, too.

Our results imply the decidability of the HDOL and HDTOL sequence equivalence problems, also longstanding open problems solved recently by K. Ruohonen and the authors, respectively.

Key words. equivalence problem, two-way transducer, single-valued transducer, unambiguous transducer, Ehrenfeucht Conjecture, DTOL language

AMS(MOS) subject classifications. 68F10, 20M35, 03D05

1. Introduction. The equivalence problem for deterministic two-way transducers was a longstanding open problem that was finally shown decidable in [10]. Our main result is a generalization of this problem. It is generalized in two directions, single-valued rather than deterministic two-way transducers are considered and their comparison is restricted to NPDTOL languages. Especially the latter extension is substantial since even a much simpler problem, namely that of testing equivalence of morphisms on PDOL languages, which is only a special case of the HDOL sequence equivalence problem, was open until recently [18], [6]. Also the decidability of the HDTOL sequence equivalence problem, shown recently in [6], follows as a special case of our results.

We demonstrate the decidability of the equivalence problem by showing that it is even decidable whether a given two-way transducer is single-valued on an NPDTOL language. Our proof uses some properties of NPDTOL languages and a recent important algebraic result, the validity of the Ehrenfeucht Conjecture [1] and [9], a problem which originated in formal language theory [7] and [14].

Let L be an arbitrary language over Σ. We say that a finite subset F of L is a *test set* (with respect to morphisms) for L if whenever two morphisms agree on F they agree on L as well. The *Ehrenfeucht Conjecture* states that each language possesses a test set. In [5] it was shown that the Ehrenfeucht Conjecture can be equivalently stated in algebraic terms as follows. Every infinite system of equations, with a finite number of unknowns, over a free monoid has an equivalent finite subsystem. The validity of the Ehrenfeucht Conjecture, in the latter form, was established in [1] and [9]. We will use this algebraic form to demonstrate the validity of a generalization of the former

* Received by the editors August 8, 1985; accepted for publication (in revised form) March 13, 1986. This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant A-7403.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

‡ Department of Mathematics, University of Turku, Turku, Finland. This work was done during the author's visit at the University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

version, namely that each language possesses a finite test set with respect to two-way single-valued transducers with a bounded number of states (Theorem 4).

Then we show that a finite test set exists effectively if L is an NPDTOL language [17]. This step relies on the authors' earlier result that the equivalence problem for systems of equations, with a finite number of unknowns, over a free monoid is decidable, which, in turn, is based on the deep result of Makanin [15] that the solvability of such systems is decidable.

Clearly, the effective existence of a test set for NPDTOL languages, with respect to single-valued two-way transducers with a bounded number of states, implies that it is decidable whether single-valued two-way transducers agree on an NPDTOL language. Since the domain of every two-way transducer is a regular set and since the family of regular sets is included in the family of NPDTOL languages we have as a corollary the result in the title of this paper.

Now, a natural question is whether the property of single-valuedness (on an NPDTOL language) itself is effectively testable. The answer is affirmative and to demonstrate it we use a minor modification of the arguments outlined above for the solution of the equivalence problem. Actually, it is easy to see that the decidability of the single-valuedness (on an NPDTOL language) implies the decidability of the equivalence of single-valued transducers (on an NPDTOL language). Hence, in the following we prove the decidability of the former. However, here in the introduction we outlined the direct proof of the decidability of the equivalence problem because it seemed more understandable.

A two-way transducer is called unambiguous if it is single-valued and its underlying acceptor is unambiguous. The unambiguous two-way transducers are properly between the deterministic and the single-valued ones. We show that it is decidable whether a two-way transducer is unambiguous. Our test for unambiguity is much simpler than that for single-valuedness.

The decidability of the equivalence and single-valuedness problems for one-way transducers (on the whole domain) was shown in [19] and later independently in [2], cf. also [3], [11], [20] and [21] for related results. A generalization of these results, namely the decidability of the equivalence or single-valuedness for one-way transducer on a context-free language was shown in [4]. The same problems for HDTOL languages, or even DOL languages, were open; however, their decidabilities follow from our results. Indeed, if instead of two-way transducers one-way transducers are considered, then our above decidability results hold also for the family of HDTOL languages, and not only for the family of NPDTOL languages. For example, it is decidable whether a given one-way transducer is single-valued on a given HDTOL language (Theorem 11).

We note that the equivalence problem for the nondeterministic transducers is undecidable, this holds even if only ε-free one-way transducers (gsm's) with one-letter input (or output) alphabet are considered [13]. We also note that the single-valuedness problem for push-down transducers is undecidable. This problem can be easily reduced to the problem of deciding the emptiness of the intersection of two context-free languages.

2. Preliminaries. A *two-way (sequential) transducer* (twt) is a 5-tuple $M = (K, \Sigma, \Delta, H, s_0)$ where K, Σ and Δ are finite nonempty sets of states, inputs and outputs, respectively, s_0 in K is the initial state and H is a finite subset of $K \times \Sigma^2 \times \Delta^* \times K \times \{-1, 0, 1\}$.

Our definition differs from the usual one (cf. [8]) in the sense that our reading head sees simultaneously two symbols of the input tape. This modification is introduced

to facilitate the proof of Lemma 3 but does not affect the decidability results since for a twt in the usual form it is easy to construct an equivalent one in our form, and vice versa (ignoring the short words). As in [8] we do not have final states or endmarkers; however, as far as our problems are concerned the model with final states and/or endmarkers is clearly equivalent. We also assume, without loss of generality, that the initial state s_0 is not entered in any computation. Informally a twt functions as follows. It starts by reading the leftmost two tape symbols while in state s_0, and a transducer terminates when the twt moves right from the rightmost tape symbol to the blank tape where no further moves are defined, i.e., where the reading head sees a blank on the right.

We will describe a *configuration* of a twt by a string $xapby$ where $x, y \in \Sigma^*$, a, $b \in \Sigma$ and $p \in K$, indicating that the finite control is in state p and the reading head above symbols a and b (to avoid defining H on "blanks" we assume that the input word is at least of length two). When $(p, ab, w, q, k) \in H$ then M in state p may read ab, write w, transit to state q and move k squares right on the input tape. Thus M may go from configuration $xapby$ into configuration $xqaby$, $xaqby$ or $xabqy$ for $k = -1$, 0, 1, respectively.

The input-output relation realized by a twt M is called two-way transduction and is denoted by $\tau(M)$. Two twt's are *equivalent* if they define the same transduction. A twt M is *single-valued* if $(x, y) \in \tau(M)$ and $(x, z) \in \tau(M)$ implies $y = z$.

A DTOL *system* (resp. PDTOL *system*) is a tuple $S = (\Sigma, h_1, \cdots, h_n, w)$ where Σ is a finite alphabet, $h_i : \Sigma^* \to \Sigma^*$ is a morphism (resp. ε-free morphism) for $i = 1, \cdots, n$ and $w \in \Sigma^+$. The language generated by S is denoted by $L(S)$ and defined as the closure of $\{w\}$ under morphisms $h_1, \cdots, h_n$. An HDTOL (resp. NPDTOL) *language* is a morphic image of a DTOL language (resp. a morphic image of a PDTOL language under an ε-free morphism). For properties of these and related languages see [17]. As a general reference to the theory of formal languages we give [12].

Let Σ be a finite alphabet and N another finite set disjoint from Σ. The *equation* over Σ^* with unknowns N is a pair $(u, v) \in (\Sigma \cup N)^* \times (\Sigma \cup N)^*$, usually written as $u = v$. A *system of equations* is any collection of equations. A *solution* of a system S of equations is a morphism $h : (\Sigma \cup N)^* \to \Sigma^*$ such that $h(a) = a$ for all a in Σ and $h(u) = h(v)$ for (u, v) in S. Since $h(a) = (a)$ for a in Σ any solution can be identified with an n-tuple from $(\Sigma^*)^n$, where n denotes the cardinality of N. Finally, we say that two systems of equations are *equivalent* if they have exactly the same solutions.

The following important algebraic result has been recently proved in [1] and [9].

THEOREM 1. *For each infinite system of equations, with a finite number of unknowns, over a finitely generated free monoid there exists (noneffectively) an equivalent finite subsystem.*

This settles the famous Ehrenfeucht Conjecture that was originally formulated in the language form below. The algebraic form was proposed and shown equivalent to the language form in [5].

Let L be any language over a finite alphabet Σ and F a finite subset of L. We say that F is a *test set* for L (with respect to morphisms) if for any pair of morphisms (g, h), $g(x) = h(x)$ holds for all x in L if and only if $g(x) = h(x)$ holds for all x in F. The language form of the Ehrenfeucht Conjecture is the following.

THEOREM 1'. *Every language over a finite alphabet possesses a test set.*

3. Single-valuedness test sets. In this section we establish our main result (Theorem 3) which implies that it is decidable whether a twt is single-valued on an NPDTOL language. Some applications of this result are shown in § 4. Our proof goes via several lemmas and theorems some of which, we believe, are of interest on their own.

We start by fixing our specific terminology. Let $T_n(\Sigma, \Delta)$ (resp. $SVT_n(\Sigma, \Delta)$) denote the family of (resp. single-valued) twt's with input alphabet Σ, output alphabet Δ and at most n states. A *transducer schema* over Ω is a twt with the following properties:

(i) if $(p, ab, u, q, k) \in H$ and $(p, ab, v, q, k) \in H$, then $u = v$ and $u \in \Omega$;

(ii) if $(p_1, a_1b_1, u, q_1, k_1) \in H$ and $(p_2, a_2b_2, v, q_2, k_2) \in H$ with $(p_1, a_1b_1, q_1, k_1) \neq (p_2, a_2b_2, q_2, k_2)$, then $u \neq v$.

Clearly, for a fixed alphabet Σ and fixed number of states, there exists only a finite number of distinct transducer schemata (up to the renaming of outputs), so under these assumptions Ω can be fixed. For our purposes it is illustrative to call Ω the *set of unknowns.*

Let $S_n(\Sigma, \Omega) = \{M \in T_n(\Sigma, \Omega) \mid M \text{ is a transducer schema}\}$. For a mapping $i: \Omega \to \Delta^*$ and S in $S_n(\Sigma, \Omega)$, we denote by $i(S)$ the twt obtained from S by replacing in H each output (unknown) c by $i(c)$. We say that $i(S)$ is an *interpretation* of S via i. Let $I(S)$ be the set of all the interpretations of S and $I(S_n(\Sigma, \Omega)) = \bigcup_{S \in S_n(\Sigma, \Omega)} I(S)$. Clearly, $I(S_n(\Sigma, \Omega)) \subsetneq T_n(\Sigma, \Delta)$. The inclusion is proper because of condition (i) in the definition of a transducer schema. However, $I(S_n(\Sigma, \Omega))$ contains $SVT_n(\Sigma, \Delta)$, assuming that we identify the transducers which differ from each other only because of useless transition rules. Consequently, $S_n(\Sigma, \Omega)$ serves as a "finite base" for the family $SVT_n(\Sigma, \Delta)$ in the sense that each twt in this family is obtained from a transducer schema in $S_n(\Sigma, \Omega)$ via an interpretation. Observe, however, that transducer schemata are not necessarily single-valued and hence $SVT_n(\Sigma, \Delta) \subsetneq I(S_n(\Sigma, \Omega))$.

Finally, we define the notion of the single-valuedness test set for a language L with respect to a class θ of twt's. We say that a finite subset F of L is a *single-valuedness test set for L with respect to θ* if any twt in θ is single-valued on L if and only if it is single-valued on F.

The terminology and notation introduced above is used throughout the paper. Now, we are ready for our first auxiliary result.

LEMMA 1. *For any language $L \subseteq \Sigma^*$ and for a transducer schema S there exists a single-valuedness test set F for L with respect to $I(S)$.*

Proof. Consider a word w in $L \cap \text{dom}(S)$. Let $\text{comp}(w)$ denote the set of all computations of w in S. We say that a computation u contains a loop if it contains a subcomputation from a configuration into itself. Let the set of loop-free computations of w be denoted by $\text{comp}_{lf}(w)$. Furthermore, let $\text{comp}_l(w)$ denote the set of all subcomputations of computations of w which are loops but do not contain any loops as proper subcomputations. Clearly, both $\text{comp}_{lf}(w)$ and $\text{comp}_l(w)$ are finite. The output of a computation u in S is, of course, a unique word in Ω^*, let us denote it by $\text{output}(u)$.

Now, we associate the word w with the following system of equations over the free monoid Δ^* with Ω as the set of unknowns:

$$\text{output}(u) = \text{output}(u') \quad \text{for all } u, u' \in \text{comp}_{lf}(w),$$

$$\text{output}(v) = \varepsilon \quad \text{for all } v \in \text{comp}_l(w).$$

Clearly, this system is finite and let us denote it by E_w. It is straightforward to see that E_w characterizes these transducers in $I(S)$ which are single-valued on w. That is to say, each solution of E_w defines such a twt (or a class of twt's), and vice versa.

Similarly, the (possibly infinite) system

$$E_L = \bigcup_{w \in L \cap \text{dom}(S)} E_w$$

of equations characterizes these twt's which are single-valued on L. But, by Theorem 1, E_L has a finite equivalent subsystem. Hence, there exists a finite subset F of $L \cap \mathrm{dom}\,(S)$ such that $E_F = \bigcup_{w \in F} E_w$ is equivalent to E_L. It follows that any twt in $I(S)$ which is single-valued on F is so also on L, proving the lemma. □

The following result generalizes Lemma 1.

THEOREM 2. *Given a language $L \subseteq \Sigma^*$ and a natural number n, there exists (non-effectively) a finite subset F of L which is single-valuedness test set for L with respect to $T_n(\Sigma, \Delta)$.*

Proof. By Lemma 1, for each transducer schema S there exists a finite subset $F(S)$ of L which tests the single-valuedness for twt's in $I(S)$. Since there exists only a finite number of different transducer schemata it follows that the set $F' = \bigcup_{S \in S_n(\Sigma,\Omega)} F(S)$ is a single-valuedness test set for transducers in $I(S_n(\Sigma, \Omega))$.

Let T be a twt in $T_n(\Sigma, \Delta) - I(S_n(\Sigma, \Omega))$. Then T is of the form $(K, \Sigma, \Delta, H_1 \cup H_2, s_0)$ where $(K, \Sigma, \Delta, H_1, s_0)$ is in $I(S_n(\Sigma, \Omega))$ and H_2 is a finite union of sets of transitions of the form

$$\{(p, ab, w_j, q, k) \mid j = 1, \cdots, t\} \tag{1}$$

where $a, b \in \Sigma$, $p, q \in K$, $k \in \{-1, 0, 1\}$, $w_j \in \Delta^*$ and $t \geqq 2$. Identifying those T's which have the same domain as well as the sets in (1) with the vectors (p, ab, q, k) we have only a finite number of possibilities for T. For each such possibility we choose, for each set of rules in (1), a word (if such exists) from $L \cap \mathrm{dom}\,(T)$ such that these rules can be used in a computation on this word. Let a finite set of words thus obtained be F''. Then it is straightforward to see that F can be chosen to be $F' \cup F''$. □

In order to prove that Theorem 2 holds effectively in certain special cases, we need a few more lemmas.

LEMMA 2. *For two finite languages L_1 and L_2, with $L_1 \subseteq L_2 \subseteq \Sigma^*$, and for an integer n, it is decidable whether or not L_1 is a single-valuedness test set for L_2 with respect to $T_n(\Sigma, \Delta)$.*

Proof. We first show that the lemma holds with respect to $I(S_n(\Sigma, \Omega))$. To prove this it is clearly enough to prove that the lemma holds with respect to all interpretations of a fixed transducer schema.

Let S be a fixed transducer schema in $S_n(\Sigma, \Omega)$. As in the proof of Lemma 1 we associate the languages L_1 and L_2 with systems of equations E_{L_1} and E_{L_2} such that E_{L_i} characterizes those transducers in $I(S)$ which are single-valued on L_i. It was shown in [5] (using a deep decidability result of [15]) that the equivalence of two finite systems of equations over a free monoid is decidable. Hence, we can decide whether L_1 is a single-valuedness test set for L_2 with respect to $I(S)$.

In order to finish the proof, we recall the considerations at the end of the proof of Theorem 2. It follows from these and from the first part of this proof that the problem of Lemma 2 is reduced to the problem of deciding whether or not a given transition rule of a given twt can be used in any computation on a given word. Of course, this problem is decidable, and hence the lemma follows. □

Our next lemma resembles the known fact that an inverse morphism does not increase the complexity of a regular language (measured as the number of states of an automaton needed to recognize the language).

LEMMA 3. *Let $h: \Gamma^* \to \Sigma^*$ be an ε-free morphism and T a* twt *in $T_n(\Sigma, \Delta)$. There exists a* twt *T_h in $T_n(\Gamma, \Delta)$ satisfying the following conditions:*

(i) $\mathrm{dom}(T_h) = h^{-1}(\mathrm{dom}\,(T))$,

(ii) *$T_h(w) \subseteq T(h(w))$ for each word w in Γ^*,*

(iii) *T_h is single-valued on $L \subseteq \Gamma^*$ if and only if T is single-valued on $h(L)$.*

In particular, if T is single-valued so is T_h and in this case $T_h(w) = T(h(w))$ for all w in Γ^*.

Proof. Clearly, the last sentence of the lemma follows from the previous ones.

In order to define T_h we need some notation and preliminary considerations. For letters a and b in Γ and for a state q in the state set Q of T we say that a computation u of T is (a, q, b)-*minimal* if

(a) it starts from the configuration $\cdots h(a)qh(b)\cdots$,

(b) it ends in a configuration either of the form $\cdots h(a)h(b)q'\cdots$ or of the form $\cdots q'h(a)h(b)\cdots$ for some q' in K,

(c) no configuration with the exception of the last one is of the form (b).

Clearly, for a triple (a, q, b) the set of (a, q, b)-minimal computations might be infinite. In order to keep it finite we define the notions of loop-free and single-loop computations: computation u above is *loop-free* (resp. *single-loop*) (a, q, b)-*minimal* if in addition to (a)-(c) it satisfies:

(d) no configuration is repeated (resp. repeated but once) in the computation.

Recalling that the initial state s_0 is never entered in any computation of T, we can modify, in an obvious way, the notions of loop-free and single-loop computations for computations starting from a configuration of the form $a's_0\beta\cdots$, with a' in Σ, β in Σ^+ and $a'\beta = h(a)$ for some a in Γ, and ending in a configuration of the form $h(a)q'\cdots$ for some q' in Q.

Using the above notation, we now construct the transitions H_h of the transducer T_h. For each loop-free or single-loop (a, q, b)-minimal computation ending in the configuration of the form $\cdots h(a)h(b)q'\cdots$ (resp. $\cdots q'h(a)h(b)\cdots$) H_h contains a transition $(q, ab, w, q', 1)$ (resp. $(q, ab, w, q', -1)$) where w is the output produced by T in the considered computation. For the state s_0 the transitions are defined analogously using above modifications. The set of states of T_h is the set of these states encountered when the above procedure is carried over all loop-free and single-loop (a, q, b)-minimal computations. Moreover, the initial and final states of T_h are those of T (belonging to the state set of T_h).

Clearly, T_h is a well-defined transducer in $T_n(\Gamma, \Delta)$. It is also a straightforward consequence of the construction that T_h satisfies the requirements (i)-(iii). □

As an application of Lemma 3 we derive the following lemma.

LEMMA 4. *Let $h: \Gamma^* \to \Sigma^*$ be an ε-free morphism and F a single-valuedness test set for a language $L \subseteq \Gamma^*$ with respect to $T_n(\Gamma, \Delta)$. Then $h(F)$ is a single-valuedness test set for $h(L)$ with respect to $T_n(\Sigma, \Delta)$.*

Proof. It follows readily from Lemma 3. Indeed, if Lemma 4 would not hold then there exists a transducer T in $T_n(\Sigma, \Delta)$ such that it is single-valued on $h(F)$ but not on $h(L)$. Then the transducer T_h of Lemma 3 is in $T_n(\Gamma, \Delta)$ and is single-valued on F but not on L, a contradiction. □

Now, finally we are ready for an effective subcase of Theorem 2.

LEMMA 5. *Given a* PDTOL *language $L \subseteq \Sigma^*$ and a natural number n, there effectively exists a single-valuedness test set F for L with respect to $T_n(\Sigma, \Delta)$.*

Proof. Let L be generated by a PDTOL system $(\Sigma, h_1, \cdots, h_n, w)$. Define, for each $i \geqq 0$,

$$L_0 = \{w\}, \qquad L_{i+1} = L_i \cup h_1(L_i) \cup \cdots \cup h_n(L_i).$$

Clearly, each L_i is finite so that we can, by Lemma 2 and Theorem 2, find effectively an integer i_0 such that L_{i_0} is a single-valuedness test set for L_{i_0+1}. Now, an obvious modification of the argument originally presented in the proof of Theorem 3.2 in [5], cf. also the proof of Theorem 4 in [6], shows that L_{i_0} is a single-valuedness test set for

L with respect to $T_n(\Sigma, \Delta)$, too. In this concluding argument a crucial role is played by Lemma 3 and its consequence Lemma 4. □

Now we are ready for our main result which is not only interesting on its own, but has, as we shall see in the next section, several important consequences.

THEOREM 3. *Given an* NPDTOL *language* $L \subseteq \Sigma^*$ *and a natural number n, there effectively exists a single-valuedness test set F for L with respect to* $T_n(\Sigma, \Delta)$.

Proof. Assume that $L = (h(L'))$ for a PDOL language L' and an ε-free morphism $h: \Gamma^* \to \Sigma^*$. Let F' be, according to Lemma 5, a single-valuedness test set for L' with respect to $T_n(\Gamma, \Delta)$. Then, by Lemma 4, we can choose $F = h(F')$. □

4. Applications of the main result. In this section we apply our main result as well as Theorem 2 to other problems. Our first applications deal with test sets for testing equivalence of single-valued twt's. Let $L \subseteq \Sigma^*$ be a language and θ a family of transducers on Σ^*. We say that a finite subset F of L is a *test set for L with respect to* θ if for any two transducers in θ they are equivalent on L if and only if they are equivalent on F. Recalling that $SVT_n(\Sigma, \Delta)$ denotes the family of single-valued twt's with input alphabet Σ, output alphabet Δ and at most n states, we have:

THEOREM 4. *For each language* $L \subseteq \Sigma^*$ *and a natural number n, there exists* (*noneffectively*) *a test set F with respect to* $SVT_n(\Sigma, \Delta)$.

Proof. Consider two single-valued twt's T_1 and T_2 with at most n states. Let $T_1 \cup T_2$ be a transducer satisfying $(T_1 \cup T_2)(w) = T_1(w) \cup T_2(w)$ for all w. Clearly, such a twt exists and can be chosen to contain no more than $2n$ states. It is also obvious that T_1 and T_2 are equivalent on L if and only if the following two conditions are satisfied:

(i) $L \cap \text{dom}(T_1) = L \cap \text{dom}(T_2)$,

(ii) $T_1 \cup T_2$ is single-valued on L.

To test (i) for T_1 and T_2 it is enough to include in F one word (if such exists) from $(L \cap (\text{dom}(T_1) - \text{dom}(T_2))) \cup (L \cap (\text{dom}(T_2) - \text{dom}(T_1)))$. Let us call this word $w(\text{dom}(T_1), \text{dom}(T_2))$. Since the set of languages $\{\text{dom}(T) \mid T \in SVT_n(\Sigma, \Delta)\}$ is finite, the set of words thus obtained when T_1 and T_2 ranges over $SVT_n(\Sigma, \Delta)$ can be chosen finite, too. Let the set of words thus obtained be F'. Thus the theorem follows if we choose $F = F' \cup F''$, where F'' is a single-valuedness test set for L with respect to $T_{2n}(\Sigma, \Delta)$. □

We also have the following effective subcase of Theorem 4.

THEOREM 5. *For an* NPDTOL *language L and a natural number n, there effectively exists a test set F with respect to* $SVT_n(\Sigma, \Delta)$.

Proof. The result follows directly from Theorem 3 and the proof of Theorem 4 providing we can show that the words $w(\text{dom}(T_1), \text{dom}(T_2))$ in this proof can be found effectively. To show this, let T_1 and T_2 be (single-valued) two-way transducers. Then $\text{dom}(T_1)$ and $\text{dom}(T_2)$ are regular languages, and hence by the closure properties of HDTOL (or equivalently EDTOL) languages, cf. [17], $(L \cap (\text{dom}(T_1) - \text{dom}(T_2))) \cup (L \cap (\text{dom}(T_2) - \text{dom}(T_2)))$ is an EDTOL language. It can be effectively found and since the emptiness for EDTOL language is decidable we can actually find effectively the word $w(\text{dom}(T_1), \text{dom}(T_2))$. Hence, the proof is complete. □

Theorems 3 and 5 have the following two interesting corollaries.

THEOREM 6. *Given an* NPDTOL *language L and a two-way transducer T, it is decidable whether or not T is single-valued on L.*

THEOREM 7. *Given an* NPDTOL *language L and single-valued two-way transducers* T_1 *and* T_2, *it is decidable whether or not* T_1 *and* T_2 *are equivalent on L.*

Next we apply our results to more classical problems of formal languages. Some of these results can possibly be obtained by modifying the arguments of [10]. We first

generalize a decidability result of Schützenberger [19], cf. also [2], which says that it is decidable whether or not a given one-way transducer is single-valued.

THEOREM 8. *It is decidable whether or not a given two-way transducer T is single-valued.*

Proof. Follows immediately from Theorem 6. Indeed, since dom (T) is regular, we can, as is straightforward to see, cf. also [16], find effectively an NPDTOL language L such that $L = \text{dom}\ (T)$. □

Similarly to above we deduce from Theorem 7 a proper strengthening of the main result in [10].

THEOREM 9. *The equivalence problem for single-valued two-way transducers is decidable.*

Note that, if we replace the condition (i) in the proof of Theorem 4 by the condition (ii) $L \cap \text{dom}\ (T_1) \subseteq L \cap \text{dom}\ (T_2)$, then we can easily modify Theorems 7 and 9 to test the inclusion of two twt's instead of their equivalence. Therefore, we also proved the following.

THEOREM 10. *The inclusion problem for single-valued two-way transducers is decidable.*

We conclude this section with the following remarks. Even the special cases of Theorems 6 and 7, where the transducers are one-way transducers, are new results. Moreover, for one-way transducers we can state these results in a stronger form as follows.

THEOREM 11. *Given an* HDTOL *language L and a one-way transducer T, it is decidable whether or not T is single-valued on L.*

Proof. In the case of one-way transducer Lemma 3 holds for all morphisms, and not only for ε-free ones. This is essentially the known result that if a language L is recognized by a finite automaton with n states then $h^{-1}(L)$, where h is an arbitrary morphism, is recognized by a finite automaton with no more than n states. After this observation the proof of Theorem 11 is a straightforward modification of that of Theorem 6. □

Similarly we have

THEOREM 12. *Given an* HDTOL *language L and single-valued one-way transducers T_1 and T_2, it is decidable whether or not T_1 and T_2 are equivalent on L.*

Note that Theorem 12 is a proper generalization of both the HDOL and the HDTOL sequence equivalence problems, which were for a long time open until shown decidable in [18] and [6], respectively. It also solves, as a special case, the (open) problem of deciding whether or not two deterministic one-way transducers are equivalent on a given HDTOL language.

5. Test for unambiguity. Our algorithm for testing the single-valuedness of a twt is complicated. If instead of the single-valuedness the stricter requirement of unambiguity is considered, then the situation is much easier; not only the algorithm but also the proof of its existence is simpler as we now show, cf. also [3], [11], [20] and [21].

We say that a twt is *unambiguous* if it is single-valued and its underlying acceptor is unambiguous. Thus we can restrict our consideration to two-way acceptors rather than twt's since the testing of single-valuedness of a twt with unambiguous underlying acceptor is easy. We recall that a two-way acceptor is *unambiguous* if for each word there is at most one successful computation.

THEOREM 13. *It is decidable whether or not a given two-way acceptor is unambiguous.*

Proof. Let $A = (K, \Sigma, E, q_0)$ where $E \subseteq K \times \Sigma^2 \times K \times \{-1, 0, 1\}$, be a two-way acceptor. We will write the transition rules in E in the form $(p, ab) \to (q, k)$. We say

that two transition rules $(p, ab) \to (q, k)$ and $(p, ab) \to (r, j)$ create a *fork* if $(q, k) \neq (r, j)$. Clearly, A is ambiguous if and only if there exists at least one such fork in E which causes the ambiguity, or more precisely, there exist two distinct computations on input $uabv$ that both reach the same configuration $uapbv$ and then split along the two distinct branches of the fork.

There is only a finite number of forks, and they can be easily listed. So the theorem holds if we can decide whether a given fork causes ambiguity.

Consider the above fork. We construct three modifications of acceptor A. Let $A_i = (K_i, \Sigma \cup \{\hat{a}, \hat{b}\}, E_i, q_i)$ for $i = 1, 2, 3$ be two-way acceptors defined as follows. Acceptor A_1 simulates A (identifying $\hat{a}$ with a and $\hat{b}$ with b) as long as the reading head is in between $\hat{a}$ and $\hat{b}$ in state p. Then it moves into a new state that moves the reading head over any word in Σ^* to the right and out of the tape.

Acceptor A_2 starts in a new initial state and moves its reading head to the right until it hits $\hat{a}\hat{b}$. Then it simulates the first branch of the fork and continues as A, interpreting $\hat{a}$ as a and $\hat{b}$ as b. Acceptor A_3 is like A_2 except it follows the second branch of the fork.

Clearly, $u\hat{a}\hat{b}v \in L(A_1)$ if and only if there exists a computation of A from the initial configuration to the configuration $uapbv$. Further, $u\hat{a}\hat{b}v \in L(A_2)$ (resp. $u\hat{a}\hat{b}v \in L(A_3)$) if and only if there is a computation of A_2 (resp. A_3) from the configuration $uapbv$ to an accepting configuration with the first atomic move following the first (resp. the second) branch of the fork. Therefore, the considered fork causes ambiguity if and only if $L(A_1) \cap L(A_2) \cap L(A_3) \neq \emptyset$. But each $L(A_i)$, $i = 1, 2, 3$, is an effectively given regular language, see [12], hence the validity of the above inequality can be tested easily. □

Theorem 13 deserves one final comment. Although we were able to use the decidability of the single-valuedness to solve also the equivalence problem for single-valued transducers, it seems not to be possible to have the same reduction for the case of unambiguity. So the simple proof of Theorem 13 does not give a simple proof of the decidability of the equivalence problem for unambiguous two-way transducers.

REFERENCES

[1] M. H. ALBERT AND J. LAWRENCE, *A proof of Ehrenfeucht's conjecture*, Theoret. Comput. Sci., to appear.

[2] M. BLATTNER AND T. HEAD, *Single valued a-transducers*, J. Comput. System Sci., 15 (1977), pp. 310-327.

[3] T. H. CHAN AND O. IBARRA, *On the finite-valuedness problem for sequential machines*, Theoret. Comp. Sci., 23 (1983), pp. 95-101.

[4] K. CULIK II, *Some decidability results about regular pushdown translations*, Inform. Process. Lett., 8 (1979), pp. 5-8.

[5] K. CULIK II AND J. KARHUMÄKI, *Systems of equations over a free monoid and Ehrenfeucht's conjecture*, Discrete Math., 43 (1983), pp. 139-153.

[6] ———, *The decidability of the* DTOL *sequence equivalence problem and related decision problems*, Research Report CS-85-05, Univ. Waterloo, Dept. Comput. Sci., 1985.

[7] K. CULIK II AND A. SALOMAA, *Test sets and checking words for homomorphisms equivalence*, J. Comput. Systems Sci., 19 (1980), pp. 379-395.

[8] R. EHRICH AND S. YAU, *Two-way sequential transductions and stack automata*, Inform. and Control, 18 (1971), pp. 404-446.

[9] V. S. GUBA, personal communication, 1985.

[10] E. M. GURARI, *The equivalence problem for deterministic two-way transducers is decidable*, this Journal, 11 (1982), pp. 448-452.

[11] E. M. GURARI AND O. IBARRA, *A note on finite-valued and finitely ambiguous transducers*, Math. Systems Theory, 16 (1983), pp. 61-66.

[12] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[13] O. IBARRA, *The unsolvability of the equivalence problem for ε-free* NGSM's *with unary input (output) alphabet and applications*, this Journal, 4 (1978), pp. 524–532.

[14] J. KARHUMÄKI, *The Ehrenfeucht Conjecture: A compactness claim for finitely generated free monoids*, Theoret. Comput. Sci., 29 (1984), pp. 285–308.

[15] G. S. MAKANIN, *The problem of solvability of equations in a free semigroup*, Mat. Sb., 103 (1977), pp. 147–236. (English Transl. in: Math. USSR Sb., 32 (1977), pp. 129–198.)

[16] M. NIELSEN, G. ROZENBERG, A. SALOMAA AND S. SKYUM, *Nonterminals, homomorphisms and codings in different variations of* OL *systems.* I *and* II, Acta Inform., 3 (1974), pp. 357–364; 4 (1974), pp. 87–106.

[17] G. ROZENBERG AND A. SALOMAA, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

[18] K. RUOHONEN, *Test sets for iterated morphisms*, manuscript, 1984.

[19] M. P. SCHÜTZENBERGER, *Sur les relations rationnelles*, in Lecture Notes in Computer Science 33, Springer, Berlin, 1975, pp. 209–213.

[20] R. STEARNS AND H. HUNT III, *On the equivalence and containment problems for unambiguous regular expressions*, this Journal, 14 (1985), pp. 598–611.

[21] A. WEBER AND H. SEIDL, *On the degree of ambiguity of finite automata*, manuscript, 1985.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987

© 1987 Society for Industrial and Applied Mathematics
002

THE OPTIMALITY OF DISTRIBUTIVE CONSTRUCTIONS OF MINIMUM WEIGHT AND DEGREE RESTRICTED SPANNING TREES IN A COMPLETE NETWORK OF PROCESSORS*

E. KORACH†‡, S. MORAN† AND S. ZAKS†

Abstract. In a previous paper we showed that the distributive construction of a spanning tree in a complete network of processors can be done in $O(n \log n)$ messages. We show in this work that if the spanning tree is required to satisfy certain properties, then the complexity of its construction increases: First we show that the construction of a minimum weight spanning tree requires, in the worst case, at least $\Omega(n^2)$ messages, and then we show that the construction of a spanning tree where the maximum degree is at most k may require at least $\Omega(n^2/k)$ messages in the worst case. Actually, in both cases the lower bounds are shown for the number of edges used in the worst case. Moreover, the results are valid for both asynchronous and synchronous networks, and are independent of the lengths of the messages. On the other hand, there are algorithms for the above tasks which achieve these lower bounds, up to a constant factor, and use messages of $O(\log n)$ length.

Key words. distributed algorithm, spanning tree, complete graph, communication complexity

AMS(MOS) subject classifications. 05C05, 68M10, 68R10, 68Q25, 68Q10

1. Introduction.

Finding spanning trees became a central research issue in distributed networks and communication protocols, as most algorithms for these environments, in a general network structure, assume an underlying structure of a spanning tree, on whose edges all communication activities take place. Using such a tree it is easy to distinguish one processor as a leader, pass a token in the network (a technique widely used in these environments to assure mutual exclusion, to check termination, etc.), broadcast messages, and other related applications. In this work we present two lower bounds on the amount of communication activities needed in order to construct spanning trees with special properties in a complete network.

The model under investigation is a network of n processors with distinct identities. Each processor has some communication lines, connecting it to some others (to all others in a complete network); it knows the (nonnegative) cost associated with each such line, but not the identities of its neighbors. The communication is done by sending messages along the communication lines. The processors all perform the same algorithm, that includes operations of (1) sending a message to a neighbor, (2) receiving a message from a neighbor and (3) processing information in their (local) memory.

We assume that the messages arrive, with no error, in a finite time, and are kept in order in a queue until processed. We also assume that any nonempty set of processors may start the algroithm; a processor that is not a starter remains asleep until a message reaches it.

The communication network is viewed as an undirected graph $G=(V, E)$ with $|V|=n$, without loops and parallel edges, and we assume that the graph G is connected. We refer to algorithms for a given network as algorithms acting on the underlying graph. An edge in this graph is considered *unused* during a certain execution of an algorithm as long as no message was sent along it (otherwise it is *used*).

* Received by the editors August 16, 1984; accepted for publication (in revised form) January 27, 1986.
† Computer Science Department, Technion-Israel Institute of Technology, Haifa, Israel 32000.
‡ This work was done while the author was with IBM Israel Scientific Center, Technion City, Haifa, Israel.

Working within this model, when no processor knows the value of n, a minimum weight spanning tree (MST) is found in [1] in $O(n \log n + |E|)$ messages for a general graph. It is pointed out in [1] that for general graphs no algorithm to find a spanning tree (ST) which is more efficient than their algorithm to find an MST is known. In [4] it is shown that $O(n \log n)$ messages are always sufficient (and $\Omega(n \log n)$ messages are necessary in the worst case) to construct a spanning tree in a complete network of processors. We show here that for such a network, the complexities of the following problems are strictly larger than the above $O(n \log n)$ upper bound:

(a) Finding an MST.

(b) Finding an ST with maximum degree at most $k = k(n)$, where $k(n) = o(n/\log n)$.

More precisely, we show that any algorithm for finding an MST in a complete graph uses, in the worst case, at least $\Omega(n^2)$ edges, and that any algorithm that finds a spanning tree with maximum degree at most k uses, in the worst case, at least $\Omega(n^2/k)$ edges. This implies that these lower bounds are also lower bounds on the number of messages used by any algorithm for these problems, where no assumption is made on the length of a message. The first lower bound is achieved, up to a constant factor, by the algorithm in [1], and the second by a composition of the leader finding algorithm in [4] and a straightforward serial construction.

The importance of a MST in a network is well known; the second problem is motivated by the fact that in certain applications it is desired to construct a spanning tree with nodes of bounded degrees, in order to prevent bottleneck effects and overcome physical restrictions in the communication network.

2. Minimum spanning tree. The main result in this section is the following theorem:

THEOREM 1. *Any distributed algorithm to find an* MST *on a complete weighted graph* $G = (V, E)$ *with* n *vertices uses, in the worst case,* $\Omega(n^2)$ *edges of the graph.*

Proof. We make use of the following basic property of distributed algorithms in the above model: A vertex cannot distinguish between unused edges of equal costs adjacent to itself (since it does not know the identities of the neighbors across these edges). The proof of this theorem, as well as the one in the next section, are described as a game between the network and an outside *adversary* performed on the graph G. Whenever a vertex wants to send a message along an unused edge of a certain cost, the adversary chooses such an edge for it (if more than one exists). The goal of the adversary is to demonstrate that the algorithm is not correct if it stops before using $\binom{m}{2}$ edges, where $m = \lfloor |V|/2 \rfloor$. The adversary may change the terminals of unused edges as long as the sets of costs of the edges adjacent to each vertex do not change, since such a change cannot be detected by any processor. Equivalently, this game may be viewed as follows: at the beginning of the execution the adversary cuts all the edges; then, whenever a processor wants to send a message along a previously unused edge, the adversary makes a connection according to some strategy, as long as the construction of the network can be consistently completed.

For example, if we have as part of the graph the subgraph shown in Fig. 1a, where the shown edges are unused, then the adversary can change the costs of the edges to these (or, alternatively, cut the edges and reconnect them to the form) shown in Fig. 1b; no vertex should be able to tell that any change was made in the network, since its local view of the network remained unchanged. This entails that the algorithm can now complete its task if and only if it could have done it before this exchange was made.

We take G to be the complete graph with vertex set $V = X \cup Y$, where $X \cap Y = \Phi$. We assume that $|V|$ is even and $|X| = |Y| = n \geq 2$. The proof is essentially the same for

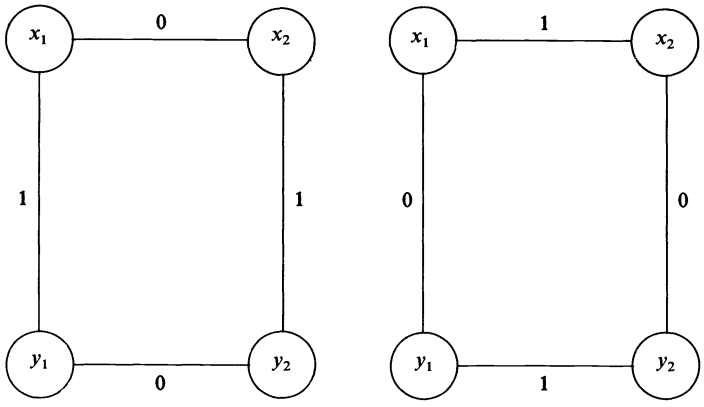


FIG. 1. *Basic exchange.* (a) *The configuration before the exchange.* (b) *The configuration after the exchange.*

$|V|$ odd. At the beginning the adversary assumes that all the edges within X and within Y have a cost of 0, and all the edges connecting X and Y have a cost of 1. It is important to note that this *initial structure* of the graph G is known *only* to the adversary; as far as each single vertex is concerned, it only knows that there are n edges of cost equal 1 and $n-1$ edges of cost equal 0 adjacent to itself.

The adversary can exchange costs of edges as described above. In fact, the adversary will make at most one exchange of costs of the type depicted in Fig. 1 (where $x_1, x_2 \in X$ and $y_1, y_2 \in Y$) at the end of the algorithm. Therefore any MST in the resulting graph will have a cost equal to 0, and will use the edge (x_1, y_1) or the edge (x_2, y_2) or both.

When the algorithm starts, some vertex eventually asks for an (unused) edge of cost 0 or 1, and the adversary chooses such an edge at random from the initial structure above and gives it to this vertex. This process is repeated each time a vertex asks for an unused edge.

CLAIM. *When the algorithm stops the set of unused edges does not contain a cycle of four edges like the one shown in Fig.* 1a, *called XY-square* (*where* $x_1, x_2 \in X$, *and* $y_1, y_2 \in Y$).

Proof. The MST produced by it uses at least one edge of cost 1 (connecting X and Y), and if such a square exists, then by making at this point an exchange as described above the adversary creates an ST of cost 0, which proves the algorithm to be incorrect. (Alternatively, the incorrectness of the algorithm follows from the fact that the same execution is possible for the new graph, which results in a spanning tree that is not of a minimum cost.)

Thus, the algorithm must use a set of edges that intersects all such cycles before it stops. As follows from the next lemma—that was chosen, because of its combinatorial flavor, to be stated in a form that is slightly more general than needed here—such a set contains at least $\binom{n}{2}$ edges. This will complete the proof of the theorem.

Let ν denote the maximum number of XY-squres which are pairwise edge disjoint, and τ denote the minimum number of edges that meet every XY-square.

LEMMA. *Let* $G=(V, E)$ *be a complete graph on* $2n$ *vertices where* $V=X \cup Y$, $X=\{x_1, x_2, \cdots, x_n\}$ *and* $Y=\{y_1, y_2, \cdots, y_n\}$, $n \geqq 2$. *Then*

(i) $\nu=\tau=\binom{n}{2}$,

(ii) $E(X)$ *and* $E(Y)$ *are the only sets of cardinality* τ *that meet all XY-squares.*

Proof. (i) Clearly $|E(X)| \geqq \tau \geqq \nu$. It is easy to see that **S** is a collection of pairwise edge disjoint XY-squares, where

$$\mathbf{S} = \{\{(x_i, x_j), (x_j, y_i), (y_i, y_j), (y_j, x_i)\}: (x_i, x_j) \in E(X)\}.$$

The cardinality of **S** is $|E(X)|$, which implies that $\nu \geqq |E(X)|$. We have

$$|E(X)| \geqq \tau \geqq \nu \geqq |E(X) = \binom{n}{2}.$$

It follows that $\nu = \tau = \binom{n}{2}$.

(ii) Let T be a minimum set of edges that meets all XY-squares (i.e. $|T| = \tau$). If T contains an edge e from X to Y, relabel X and Y (if necessary) so that $e = (x_1, y_1)$. Clearly e does not meet any XY-square in **S**, hence $|T| > \binom{n}{2}$, a contradiction. If T meets both $E(X)$ and $E(Y)$ then (since $|E(X)| = |E(Y)| = |T|$) we have an edge in $E(X)$ and an edge in $E(Y)$ both not in T. It follows that these two edges are in an XY-square that does not meet T. This implies (ii). □

Remark 1. In the case where all the edge-weights are distinct, and arbitrarily long messages are allowed, an MST can be constructed in the following way:

(1) Choose a leader using the logarithm in [4] ($O(n \log n)$ messages).

(2) Each node transfers to the leader the weights of the edges adjacent to it ($O(n)$ "long" messages).

(3) The leader reconstruct the weighted graph by assigning to the edge (i, j) the unique weight that was transferred to it by both i and j.

(4) The leader finds the MST of G, and transfer to each node the weights of the MST edges adjacent to it ($O(n)$ messages).

This observation was independently made in [3].

The above remark can be generalized to show that if we have, in a given graph G, k edges of nondistinct weights, then an MST of G can be constructed in $O(\max\{k, n \log n\})$ "long" messages.

3. Degree restricted spanning trees. Spanning trees in distributed networks are generally used to simplify the communication in the network. If in such a tree the degree of a node is large, it might cause an undesirable communication load in that node. In such cases, the construction of spanning trees in which the degree of a node cannot exceed some given value k (which might depend on the size of the network n), is needed. The known $O(n \log n)$ algorithms for constructing a spanning tree in a complete network of processors may not meet such a requirement. In fact, the following theorem shows that for $k = o(n/\log n)$ any $O(n \log n)$ algorithm must fail in certain cases. It should also be noted that, unlike the MST problem, the corresponding problem for general graphs is NP-hard [2]. This make the focus on complete networks more interesting.

THEOREM 2. *For all n and k, $1 < k < n$, any distributed algorithm that constructs a spanning tree with maximum degree bounded by k in a complete graph on n vertices, uses, in the worst case, $\Omega(\max\{n \log n, n^2/k\})$ edges.*

Proof. The $\Omega(n \log n)$ lower bound follows from the lower bound on the number of messages required for the construction of any spanning tree [4]. It remains to prove the $\Omega(n^2/k)$ lower bound for the case where $k \leqq n/\log n$. In the rest of the proof we assume this case, and we again use the adversary approach. We also assume that any algorithm for this problem uses all the edges of the constructed tree (this might increase the number of edges used by it by at most $n-1$). Let A be such an algorithm, and assume that the vertices of the graph are $\{1, \cdots, n\}$. Whenever a vertex j wishes to

send a message along an unused edge, the adversary supply it with the edge (j, i) with the smallest possible i.

Assume that the required tree T was constructed by the algorithm. We take vertex 1 to be the root of T and direct the edges to get an arborescence rooted at 1 (i.e., there is a directed path from 1 to every vertex in T). Let D be the set of edges of T on which the first message was sent in the direction of the edge, and let B be the set of the remaining tree edges. Let $d=|D|$ and $b=|B|=n-1-d$. Assume first that $d \geqq b$ (and hence $d \geqq (n-1)/2$), and let $S=\{s_1, \cdots, s_{|S|}\}$ be the subset of vertices which are a *tail* of an edge in D. For each $s_i \in S$ let m_i be the maximal vertex to which an edge of D is directed from s_i. Without loss of generality assume that $m_1 \leqq m_2 \leqq \cdots \leqq m_{|S|}$.

CLAIM 1. $m_{|S|} \geqq d+1$.

Proof. This follows from the fact that in T the indegree of each vertex, other than the root, is one, and hence $m_{|S|}$ is the maximal integer in a set of d positive integers where the minimal is at least 2 (since 1 is the root and has indegree zero). □

CLAIM 2. *For* $i=1, \cdots, |S|-1$, $m_{|S|-i} \geqq d-i \cdot k+1$.

Proof. At most k edges in D leave each vertex in S. Hence $m_{|S|-i}$ is the maximal vertex in a set of at least $d-i \cdot k$ vertices, which, again, does not contain the root. □

By the adversary rule and the definition of m_i, at least m_i-1 edges adjacent to s_i were used (by a message to or from s_i). Thus, the total number of used edges adjacent to vertices in S is at least

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{|S|} \max\{(d-k\cdot(i-1)+1-1), 0\} &\geqq \frac{1}{2}[d+(d-k)+(d-2k)+\cdots+d \bmod (k)] \\ &= \Omega(d^2/k) \\ &= \Omega(n^2/k) \qquad (\text{since } d \geqq (n-1)/2).\end{aligned}$$

This completes the proof in the case where $d \geqq b$.

Assume now that $b>d$, and let $U=\{u_1, \cdots, u_{|U|}\}$ be the set of vertices which are a head of an edge in B. Let n_i be the maximal vertex from which an edge of B is directed to u_i, and assume that $n_1 \leqq n_2 \leqq \cdots \leqq n_{|U|}$. Since the degree of each vertex is at most k, we have that:

$$\begin{aligned}&1 \leqq n_1 \leqq n_2 \leqq \cdots \leqq n_k, \\ &2 \leqq n_{k+1} \leqq n_{k+2} \leqq \cdots \leqq n_{2k}, \\ &\qquad \vdots \\ &i+1 \leqq n_{ik+1} \leqq n_{ik+2} \leqq \cdots \leqq n_{(i+1)k}, \\ &\qquad \vdots \\ &t+1 \leqq n_{tk+1} \leqq n_{tk+2} \leqq \cdots \leqq n_b,\end{aligned}$$

where t is such that $b=tk+r$, $1 \leqq r \leqq k$.

Using the adversary rule again, we get that the total number of edges adjacent to nodes in U that were used by the algorithm is at least

$$\begin{aligned}\tfrac{1}{2}k(1+2+\cdots+t) &= \Omega(k\cdot t^2) \\ &= \Omega(k\cdot(b/k)^2) \\ &= \Omega(n^2/k) \qquad (\text{since } b>(n-1)/2).\end{aligned}$$

This completes the proof of the theorem. □

Remark 2. A spanning tree with maximum degree at most k can be constructed in the following way:

(1) Choose a leader using the algorithm in [4] ($O(n \log n)$ messages).

(2) The leader chooses k descendants and asks one of them, say v_1, to choose $k-1$ yet unchosen descendants.

(3) v_1 chooses its descendants by serially asking its neighbors to be its descendants, and waiting for their positive or negative response. Then v_1 asks one of its chosen descendants, say v_2, to choose $k-1$ yet unchosen descendants, and so on.

Clearly, the leader sends k messages, v_1 sends less than $2k$ messages (and receives less than $2k$ responses), v_2 sends less than $3k$ messages (and receives less than $3k$ responses), etc. The total number of messages sent is less than $k+4k+\cdots+2\lceil n/k\rceil k = (n^2/k)+O(n)$ (and the total number of edges used is at most $(n^2/2k)+O(n)$).

Remark 3. In virtue of the $(n^2/k)+O(n)$ upper bound it should be pointed out that the analysis in the proof shows that the number of edges that must be used by the algorithm is at least $n^2/16k-O(n)$.

Remark 4. Using the same notations as in the proof of Theorem 2, but a less detailed analysis, a shorter proof can be given, but with a constant worse in a factor of 4 (compare with Remark 3): Let F be the directed forest formed by edges of T both ends of which are numbered $\geqq n/2k$. Clearly, F has at least $n/2$ edges. Associate with the tail of each backward edge a count equal to the label of the head. Since each label is $\geqq n/2k$, we can assume that at least $n/4$ of the edges are forward (otherwise, the total count is $\Omega(n^2/k)$). At least $n/8$ of these have heads numbered at least $n/8$. Hence at least $n/8k$ vertices have a forward descendant numbered at least $n/8$. Thus the total count is at least $\Omega(n^2/k)$.

Remark 5. One can see that the proofs of the two lower bounds, including the roles of the adversaries, remain valid also for the case where the execution is synchronous and no upper bound is assumed for the length of a message.

Acknowledgments. We would like to thank an anonymous referee for a number of suggestions including Remark 4, and Reuven Bar-Yehuda for some helpful comments.

REFERENCES

[1] R. G. GALLAGER, P. A. HUMBLET AND P. M. SPIRA, *A distributed algorithm for minimum spanning tree*, Trans. Programming Languages and Systems, 5(1) (1983), pp. 66–77.

[2] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, CA, 1979.

[3] E. GAFNI, private communication.

[4] E. KORACH, S. MORAN AND S. ZAKS, *Tight lower and upper bounds for distributed algorithms for a complete network of processors*, Proc. Third ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, August 1984, pp. 199–207.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics

FAST ALGORITHMS FOR BIPARTITE NETWORK FLOW*

DAN GUSFIELD†, CHARLES MARTEL‡ AND DAVID FERNANDEZ-BACA‡

Abstract. We discuss network flow in a bipartite graph G with node sets S and T. We show that Karzanov's algorithm runs in time $O(|S|^2|T|)$ on G, and that the MPM algorithm can be modified to run in the same bound. For the common case that the degree of each node in T is bounded by a constant, we show that a modified version of the MPM algorithm runs in time $O(|S|^3+|S||T|)$ on G. The importance of these results comes from the common occurrence of problems that can be modeled and solved using network flow on bipartite graphs where $|S| \ll |T|$. In many applications involving bounded degree on the nodes in T, $|T|=\theta(|S|^2)$, and our results reduce the best bounds on bipartite flow from $O(|S|^6)$ to $O(|S|^3)$. We use our results on bipartite flow to obtain the fastest known bounds on several combinatorial problems, and mention several additional problems involving bipartite flow where $|S|$ is much smaller than $|T|$.

Key words. network flow, graph theory, scheduling, combinatorial optimization

AMS (MOS) subject classifications. 90B10, 90B35, 90C35, 68Q25, 05C99

1. Introduction.

The maximum flow and minimum cut problem is used both as an algorithmic and mathematical tool to model and solve hundreds of interesting and practical problems in combinatorics and combinatorial optimization, and the importance of the network flow problem is well established. A large subset of these network flow problems are modeled and solved on graphs which are *bipartite*, and which have the property that the set of nodes, S, on one side of the graph is much smaller than the set of nodes, T, on the other side of the graph; we list and discuss several such problems in § 4.

1.1. Main results.

In this paper we show that the flow problem (finding the maximum flow and the minimum cut) on such bipartite graphs can be solved far more efficiently than has been previously established. In particular, we show that the bipartite flow problem can be solved in time $O(|S|^2|T|)$ when $|S| \leqq |T|$, and for the class of graphs where the degree of each node in T is bounded by a constant, bipartite network flow can be solved in time $O(|S|^3+|S||T|)$. Many network flow problems of importance have exactly this bounded degree structure, as we will later illustrate. We then express the main result in terms of the size of the maximum independent set in the graph, and show that our results imply faster network flow algorithms for graphs with one large independent set of nodes; the bipartite graphs are a special case of this class. We also mention simpler, slower network flow algorithms which can be shown to run faster on bipartite graphs than the general bounds suggest.

Presently known general upper bounds for network flow imply $O((|S|+|T|)^3)$ or $O((|S|+|T|)|E|\log|E|)$ time bounds for bipartite flow. For most of the applications we consider, $|T|=\theta(|S|^2)$, and so our results reduce the best bounds from $O(|S|^6)$ to $O(|S|^4)$ and often to $O(|S|^3)$. For many of the problems discussed, there are known methods for solving the problems which do not use bipartite flow, and which previously looked superior to solution methods based on bipartite flow. For each problem discussed in this paper, our bounds on bipartite flow yield methods that beat or equal the best known bounds of any of the other solution methods.

* Received by the editors July 10, 1985; accepted for publication (in revised form) April 17, 1986. This work was partially funded by the National Science Foundation under grant number MCS-81/05894, and by the IBM Corporation and the State of California through project MICRO.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520.

‡ Department of Computer Science, University of California, Davis, California 95616.

The better running times presented in this paper are achieved by a more careful analysis of well-known flow algorithms, and by simple modifications of these algorithms: we use a new charging argument to count the work in the well-known MPM and Karzanov algorithms; we establish the $O(|S|^2|T|)$ bound in MPM by the use of F-heaps in the implementation, but it is the charging argument that shows the introduction of F-heaps is useful; we establish the same bound for Karzanov's algorithm without any modification of the algorithm; we establish the $O(|S|^3+|S||T|)$ bound, for the case of bounded degree, using another modification of the MPM algorithm, but without the use of F-heaps.

1.2. Applications of these results. The importance of these algorithms and bounds are due to the common occurrence, both in practice and as problem models, of bipartite flow problems where $|S| \ll |T|$. The payoffs from our results are two-fold.

First, they lead to the fastest known solutions for certain combinatorial problems which can be modeled as bipartite flow. We illustrate this with two problems: for a well-known scheduling problem, we reduce the previous best running time from $O(m^{2.5}n^3)$ to $O(mn^3)$, where m is the number of machines, and n is the number of jobs; and for a large class of 0–1 integer programming problems, we reduce the best upper bounds from $O((|S|^3|T|+|T|^2|S|^2)\log^2|T|)$ to $O(|S|^3|T|)$. In the second application, $|T|$ is often tremendously larger than $|S|$; the only bound on $|T|$ is $2^{|S|}$, although $|T|$ is polynomially bounded in $|S|$ for most of the interesting problems in this class.

Second, our results allow the use of bipartite flow as an efficient model for many problems that can be modeled both as bipartite flow and as flow in a nonbipartite graph. For most of these problems, the bipartite model is conceptually simpler, but the competing nonbipartite model typically has fewer nodes: $|S|$ versus $|S|^2$. For that reason, the belief reflected in the literature has been that the nonbipartite model permits faster execution of network flow algorithms—$O(|S|^3)$ time versus the unattractive $O(|S|^6)$. The results of this paper show that flow in the competing bipartite models can be solved in $O(|S|^3)$ time, and hence the accepted logic for preferring the more complex nonbipartite models is flawed, although we of course don't know which model ultimately permits the most efficient flow computation. In the applications section of this paper we will discuss several problems that have such competing bipartite and nonbipartite solutions. One such problem, the subgraph density problem, is also an example of the class of 0–1 integer programming problems mentioned above. Building on these results, we note that the time needed to solve weighted generalizations of the subgraph density problem can be reduced from the reported bound of $O(|n|^6)$ to $O(|n|^4)$.

More generally, our results encourage the consideration of bipartite flow models where more complex nonflow models are known, and demonstrate the utility of looking beyond general upper bounds.

1.3. Outline of the paper. In § 2 we give the basic definitions, review the MPM and Karzanov algorithms, and analyze the MPM algorithm applied to the bipartite flow problem. Using F-heaps and a new charging argument, we obtain the $O(|S|^2|T|)$ time bound. We also show the same time bound for Karzanov's algorithm. In § 3 we examine the case of bounded degree on each T-node (excluding the source or sink); we modify the MPM algorithm and show that it has running time $O(|S|^3+|S||T|)$ for this class of graphs. In § 4 we discuss applications of these results to a variety of problems.

2. Computing maximum bipartite flow in time $O(|S|^2|T|)$. In this section we briefly describe and analyze the MPM maximum flow algorithm, with the modification that

we use Fibonacci Heaps [7] in its implementation; this is the only modification we make to obtain the $O(|S|^2|T|)$ bound.

We will also show that Karzanov's algorithm without modification runs in $O(|S|^2|T|)$ time on bipartite flow graphs. Our main contribution is the *analysis* of these algorithms on bipartite graphs.

2.1. The modified MPM algorithm. We assume familiarity with the s-t flow problem and the standard MPM algorithm but outline the algorithm in order to establish notation and introduce an important modification. The algorithm iterates *phases*, where a phase is the following:

Phase. Given a graph $G=(V, E)$ with source node s, sink t, and the current flow F, construct a directed *layered* graph $G(F)$, and find the *maximal* s-t flow in $G(F)$. A maximal s-t flow is one where at least one edge of every s-t path is saturated (a maximal flow is not necessarily a maximum flow). If the maximal flow is zero, then F is the *maximum* flow in G; otherwise use the maximal flow to update F.

We will not describe how to construct a layered graph and update the flow, but it is well known [3] that both those operations can be done in time linear in the number of edges in the graph. In $G(F)$ the nodes are partitioned into layers which are ordered so that edges only run from a layer to its successor. It is well known that the number of layers in the layered graph increases in each consecutive phase.

Finding a maximal flow. Now we describe in more detail how to find the maximal flow in a layered graph; our analysis of MPM will concentrate on these details. We need the following definitions.

DEFINITION 2.1. For a node v in a layered graph $G(F)$ define the *forward potential* of a node v, $f(v)$ to be the sum of capacities on the edges out of v, the *backward potential*, $b(v)$ to be the sum of the capacities of the edges into v, and the potential $p(v)=\min[f(v), b(v)]$. Define $p=\min[p(v)]$.

The algorithm finds a maximal flow in $G(F)$ by iterating the following *augmentation* step:

Augmentation. Find the node w of minimum potential, and push $p(w)$ units of flow in $G(F)$ from w to node t as follows. Choosing some ordering of the edges out of w, push flow along the edges in order, saturating each used edge, if possible, until $p(w)$ units of flow are pushed out of w. Note that after this step, at most one edge out of w which is used in this augmentation is partially filled (has flow strictly between zero and its capacity). For each node that receives incoming flow, push that flow out in the same way as above, and continue such pushes until $p(w)$ units of flow reach node t. In a similar way, push $p(w)$ units from w back to s.

The facts that w is the node of minimum potential and $G(F)$ is a layered graph assure that $p(w)$ units will be successfully pushed from s to t via node w. Edges with flow at their capacity are said to be *saturated*. After each augmentation, all saturated edges are deleted. Notice that this either cuts all paths from s to w, or all paths from w to t, hence node w can be removed also.

When we change the flow in an edge, we will call this a *saturating edge push* if the edge becomes saturated; otherwise we call it a *nonsaturating edge push*. Note that an augmentation changes the potential of all nodes incident with an edge whose flow is changed. In order to repeatedly find the minimum potential node efficiently, we will store and update the node potentials using the Fibonacci Heaps of Fredman and Tarjan [7]. We will refer to these as F-heaps. The introduction of F-heaps here, instead of some other priority queue, is the only modification we make to the standard MPM algorithm.

Thus the work for each phase is:

(1) Build the layered network.
(2) Compute all node potentials and store them in an F-heap.
Iterate the following steps until a maximal flow is achieved.
(3) Find the node of least potential.
(4) Push flow through the network, updating the flow and capacities on changed edges.
(5) Update the node potentials, and rebuild the heap of node potentials.

2.2. Analysis of the modified MPM algorithm. In the following analysis we will assume that $|T| > |S|$; however, a similar result will hold when $|T| < |S|$. Let E be the set of edges in the flow network and let $V = S \cup T \cup \{s\} \cup \{t\}$ be the set of nodes in the network. Since $G(F)$ is bipartite, every layer (except the $\{s\}$ and $\{t\}$ layers) consists exclusively of nodes of S or exclusively of nodes of T; we call a layer an S-layer or a T-layer depending on the type of nodes it contains.

THEOREM 2.1. *The algorithm described in* § 2.1 *finds a maximum flow in* $O(|S|^2|T|)$ *time on a bipartite flow network, for* $|T| < 2^{|S|}$.

Note that the standard analysis would give $O((|S|+|T|)^3)$, so our analysis shows there is a substantial savings when $|T| \gg |S|$. The assumption that $|T| < 2^{|S|}$ is no restriction in our applications and is not needed in the analysis of Karzanov's algorithm.

Proof. We first prove a general lemma about algorithms which work in phases on a bipartite flow graph.

PHASE LEMMA 2.1. *At most* $O(|S|)$ *phases are required to find a maximum flow in a bipartite flow graph.*

This is in contrast to the $O(|S|+|T|)$ phases needed for a general graph.

Proof of phase lemma. In any layered graph $G(F)$, every other layer (except $\{s\}$ and $\{t\}$) is an S layer; hence $G(F)$ has at most $2|S|+2$ layers. □

Before going on, we should note that application of the phase lemma alone reduces the bound on the Sleator, Tarjan flow methods [25], [26] from $O((|S|+|T|)|E| \log |E|)$ to $O(|S||E| \log |E|)$, which is within $\log |E|$ of the bounds we are establishing for the modified MPM and Karzanov algorithms.

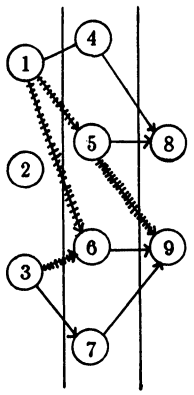
We will now show that each phase of the modified MPM algorithm can be run in $O(|S||T|)$ time. Steps (1) and (2) are done once per phase and can be done in $O(|E|)$ time, which is $O(|S||T|)$. Using F-heaps, step (3) needs only time $O(|V| \log |V|) = O(|T||S|)$ for $|T| < 2^{|S|}$. The heart of the $O(|S||T|)$ bound is in the analysis of steps (4) and (5). In order to analyze the work in step (4), we need to bound the number of edge pushes used. Each edge push either saturates the edge or is a nonsaturating push. Clearly, the total number of saturating edge pushes in a phase is $O(E) = O(|S||T|)$, so the key issue is how many nonsaturating edge pushes there are in a phase. Standard analysis of MPM gives $O((|S|+|T|)^2)$. Our key observation is that this can be reduced to $O(|S||T|)$. The proof of Theorem 1 will be completed with the following two lemmas.

CHARGING LEMMA 2.2. *Let* A *be the number of augmentations in a phase. Then the total number of nonsaturating edge pushes in a phase is at most* $4A|S|+|E|$, *which is* $O(|S||T|)$.

Proof. We examine the pushes from x, the minimum potential node, towards t. The pushes towards s are handled in a similar manner. In any augmentation there is at most one nonsaturating edge push from any node. Each such nonsaturating edge push will be charged to either an S-node or to a saturated edge.

Let (v, w) be an edge which has its flow increased without being saturated. If v is an S-node, then (v, w) is charged to the node v. If v is a T-node, then there must

be some edge (u, v) that has its flow increased by the current augmentation, and u must be an S-node. If (u, v) is saturated by the current augmentation, then (v, w) is charged to (u, v). If (u, v) is not saturated, then (v, w) is charged to node u. Note that an edge (v, w) incident from the T-node v may be eligible to be charged to several edges or nodes or both. In this case it can be charged to any one of them. Figure 1 illustrates an allocation of charges.



Edge	Charged to
(1,4)	node 1
(3,7)	node 3
(4,8)	node 1
(5,8)	edge (1,5)
(6,9)	edge (3,6)
(7,9)	node 3

FIG. 1. *Charge allocation for nonsaturating edge pushes. Only augmented arcs are shown. Edges saturated in the current augmentation are shown as dashed.*

In each augmentation, each *S-node* can be charged at most twice: once for the edge leaving it which has a nonsaturating edge push, and once for the edge leaving the T-node incident to this edge. Hence in A augmentations, there are at most $2A|S|$ charges to S-nodes per phase. In a phase, each *edge* can be charged at most once, and this can happen only in the augmentation which saturates it. So, there are $O(|E|) = O(|S||T|)$ charges to edges per phase. Thus the total number of charges per phase is at most $2A|S|+|E| = O(|S||T|)$, since $A = O(|T|)$ and $|E| = O(|S||T|)$. Repeating the analysis for pushes from x to s, and observing that every edge in the graph is saturated at most once, the bound of $4A|S|+|E|$ is proved. □

POTENTIAL LEMMA 2.3. *In a phase, the total number of node potential changes is at most* $4A|S|+|E| = O(|S||T|)$, *which bounds the time needed for step* (5).

Proof. A node potential changes only when flow is pushed in an incident edge. Since potentials only decrease, the F-heap maintains the node potentials in time linear in the number of changes [7], which is, by the charging lemma, at most $4A|S|+|E| = O(|S||T|)$ per phase. Note that without the charging lemma, we would have to count $O((|S|+|T|)^2)$ for the potential changes, even using F-heaps. □

2.3. An $O(|S|^2|T|)$ bound for Karzanov's algorithm. In this section we will show that for bipartite graphs Karzanov's algorithm [16], unmodified, has the same $O(|S|^2|T|)$, assuming only that $|S| \leq |T|$. We assume familiarity with Karzanov's algorithm, but outline it here in order to permit the analysis.

Karzanov's algorithm, like MPM, iterates phases, where each phase finds a maximal flow in a layered network by repeated augmentations. During a phase, the flow in an edge may be increased several times, but it can be reduced at most once, after which the edge is declared *closed* and can be used no longer. We need the following:

DEFINITION 2.2. A *preflow* is a function f on the edges of a layered graph such that for each edge u, $f(u) \leqq$ capacity (u), and for any $v \in V$, $\sum_{e \in \alpha(v)} f(e) \geqq \sum_{a \in \beta(v)} f(a)$, where $\alpha(v)$ is the set of edges going into v and $\beta(v)$ is the set of edges leaving v.

A node is *unbalanced* if the flow entering it is greater than the flow exiting it. A preflow is similar to an ordinary flow, but it allows unbalanced nodes. Karzanov's algorithm repeatedly augments until there are no unbalanced nodes except s and t. An augmentation consists of two steps: *advance of preflow* and *balance of preflow*, which are described below. Let L_k denote the kth layer of the layered graph, $k = 0, \cdots, r$, where $L_0 = \{s\}$ and $L_r = \{t\}$, and let j be the index of the layer from which the augmentation starts. Initially, all edges are open, $j = 1$, and the edges directed out of s are saturated.

Advance of preflow. The purpose of this step is to establish a new preflow in the layered graph. Repeat the following operation for each L_i, $i = j, \cdots, r$, until either the last layer is reached or it is impossible to push any flow into the next layer:

For each unbalanced node $u \in L_i$, balance it, if possible, by pushing excess flow into the next layer.

Outgoing edges are used in a fixed order, in such a way that the advance leaves at most one partially filled edge per node. Notice that it is not necessary to consider all the unbalanced nodes in a given layer; it suffices to look at those that have just had their incoming flow increased or their outgoing flow decreased.

Balance of preflow. Let L_k be the highest layer containing unbalanced nodes in the graph. For each unbalanced node $v \in L_k$, reduce the flow in its incoming edges, starting from the last edge that was used, and proceeding in reverse order of usage until v is balanced. The flow in an edge is decreased only if the flow in other, more recently used, edges has already been reduced to zero. Declare closed all edges in $\alpha(v)$. The next advance starts at L_j, where $j = k - 1$.

We say a node v is *blocked* if every path from v to t contains at least one saturated edge. It can be proved [16] that an unbalanced node becomes blocked after an advance, that a blocked node remains blocked throughout the entire phase, and that a node is balanced at most once in a phase. Furthermore, from the description of the algorithm, we see that every augmentation, except the last, balances at least one node. These properties together show that Karzanov's method correctly computes a maximal flow in a layered graph using at most $|V| - 1$ augmentations per phase.

We summarize the steps in a phase of Karzanov's algorithm:

(1) Build the layered network.
Iterate the following steps until every node is balanced except s and t.
(2) Advance preflow.
(3) Balance the preflow.

Our analysis parallels that of MPM, the key argument being a modification of the charging lemma of § 2.2.

THEOREM 2.2. *Karzanov's algorithm finds a maximum flow in* $O(|S|^2|T|)$ *time, where* $|S| < |T|$.

Proof. We will show that each phase can be carried out in $O(|S||T|)$ time, which together with the phase lemma proves the theorem. Step (1) is done once per phase and takes $O(|S||T|)$ time. Steps (2) and (3) are repeated $O(|V|) = O(|T|)$ times. The total number of saturating edge pushes is $O(|E|) = O(|S||T|)$. The total number of edge flow reductions in both saturated and unsaturated edge is also $O(|E|)$. The problem now is to bound the number of edges that are used in step (2) but not saturated, and which are not subsequently closed in step (3). For this analysis, we divide a phase

into its augmentations. Each augmentation begins at a particular layer L and pushes flow from L to t. We charge all nonsaturating edge pushes out of layers higher than L exactly the same way as in the Charging lemma. However, the nonsaturating edge pushes out of L must be charged differently.

In the first augmentation, $L = L_1$; so, we charge all nonsaturating edge pushes out of nodes in L to saturated edges leaving s. For each subsequent augmentation, excess flow is pushed out of a node $v \in L$ only if there was at least one edge out of v that had its flow reduced in the previous balance step. Thus, we charge a nonsaturating edge push from v to the last edge emanating from v whose flow was reduced. Recall that such flow reduction is done at most once per edge in a phase, so there are at most $O(E)$ of these charges.

As was the case for MPM, all charges are either to S-nodes or to edges. During an augmentation, no S-node is charged more than twice, giving us $O(|S||T|)$ charges to S-nodes per phase. An edge is charged at most twice per phase, once when it is saturated and once when its flow is reduced; so, we have $O(|S||T|)$ charges to edges. The total amount of work in each phase is thus $O(|S||T|)$, yielding an overall running time of $O(|S|^2|T|)$. □

2.3.1. Corollaries to Theorems 2.1 and 2.2.

THEOREM 2.3. *Given a layered graph where S is the set of nodes in the even numbered layers, and T is the set of nodes in the odd numbered layers, both modified* MPM *and Karzanov's algorithms find a maximal flow in this layered graph in* $O(|S||T|)$ *time, provided that* $\min\{|S|, |T|\} \geqq \log(|S|+|T|)$.

Note that in the case of Karzanov's algorithm, the restriction is unneeded.

DEFINITION 2.3. A graph is called a *partitioned* flow network if its nodes can be partitioned into two sets S and I, so that no two nodes in I are connected by an edge. The partitioned graphs properly contain the bipartite graphs, and for a bipartite graph either of the two sides of the graph can take the role of I.

THEOREM 2.4. *Both modified* MPM *and Karzanov's algorithm find a maximum flow in* $O((|I|+|S|)|S|^2)$ *time on a partitioned flow network.*

Proof. At least one node from S must appear in any two consecutive layers except the first and the last two; so, there are $O(|S|)$ phases. Each nonsaturating edge push can be charged to either a node of S or to a saturated edge, just as in the charging lemma. The argument holds for partitioned graphs because edges incident to an I-node must be to or from nodes in S. Thus as in the bipartite case, a nonsaturating edge push out of a node in I can be charged to an S-node or to a saturated edge. Thus the work within a phase is $O(|E|)+O(|V||S|) = O((|I|+|S|)|S|)$, and the total work is $O((|I|+|S|)|S|^2)$. □

3. Bipartite flow with bounded degree on the *T*-nodes. In this section we consider the bipartite flow problem when the degree of each T-node of G, other than s or t, is bounded by the constant d. We will refer to this problem as the *bounded degree bipartite flow problem.* We show that a simple modification of the MPM algorithm, even without the use of F-heaps, results in a running time of $O(d|S|^3+d^2|S||T|) = O(|S|^3+|S||T|)$. Bounded degree bipartite flow problems are common; in fact, it is common that $d = 2$ (we mention several examples in the applications section). Note that since we have assumed that $|S| < |T|$, the case of bounded degree of S-nodes is trivial.

3.1. The key idea. The key idea in this section is that we will modify the definitions of node potentials and their use in the MPM algorithms so that, in each augmentation, we need search only the S-nodes for the node to which flow is pushed from s, and from which flow is pushed to t. Consequently, the number of augmentations inside a

phase is bounded by $|S|$ instead of $|S|+|T|$, and further, since we only need to find the minimum potential among the S-nodes, we can find the node in $O(|S|)$ time by simply scanning all the S nodes. Hence no F-heaps are needed. The assumption of bounded degree on the T-nodes is used to allow efficient updating of the node potentials.

DEFINITION 3.1. For a node y in a T layer of layered graph $G(F)$, we define $f(y)$ as before, as the sum of the capacities of all the edges out of y, and we define $b(y)$ as the sum of the capacities of all the edges into y.

DEFINITION 3.2. If (x, y) is a directed edge from an S-node x to a T-node y in $G(F)$, and the capacity of (x, y) is $c(x, y)$, then $\bar{c}(x, y)$ is defined to be $\text{Min}[c(x, y), f(y)]$. We call $\bar{c}(x, y)$ the *pseudo-capacity* of edge (x, y), and it is the amount that can be sent from x through edge (x, y) to the next S-layer closer to t.

DEFINITION 3.3. For an S-node x in $G(F)$, define $f(x)$ (the forward potential) as $\sum_{y \text{ in the layer after } x} \bar{c}(x, y)$, i.e. the maximum amount of flow that can be pushed from x to all the nodes of the next S layer in $G(F)$. Similarly, define $b(x)$ (the backward potential) as the amount of flow that can be pushed from the S layer preceding the layer containing node v, to node v. Let $p(v) = \min[f(v), b(v)]$, and let p be the minimum $p(v)$ value over all S-nodes of $G(F)$.

Note that the definitions of potentials for a T-node are standard, but the definitions of potentials for an S-node differ from the standard definitions in that we are looking two layers ahead and back. Further, the definition of p differs from the standard in that it is the minimum node potential taken only over the S-nodes of $G(F)$, hence there may be a T-node with forward or backward potential less than p.

LEMMA 3.1. *For $G(F)$ and p as above let v be an S-node such that $p(v) = p$. It is possible to push a flow of p units from s to t through v.*

Proof. As before, p units will be pushed from v to t, and from s to v. Consider an S-layer, K, of $G(F)$ which has received p units of flow from v. We show that p units can be pushed to the next S-layer. Take a node x in K and let $a(x)$ be the amount of flow pushed into x. Since $f(x) \geqq p \geqq a(x)$, that flow can be pushed forward to the next S-layer. Now consider a second node x' in K which receives $a(x')$ incoming flow, and let K' be the succeeding T layer. Certainly, $f(x') \geqq p \geqq a(x')$, but some of the edges used to compute $f(x')$ are used in the push from x; hence these relations alone do not assure that $a(x')$ units can also be pushed from x' to the next S layer. However, the edges from x' to K' are disjoint from the edges out of x, hence $a(x')$ units can be pushed from x' to K', and so at least $f(x') - a(x)$ units can be pushed from x' to the next S layer. But, $a(x') \leqq p - a(x) \leqq f(x') - a(x)$, so all $a(x')$ units can be pushed to the next S layer. Repeating this argument for all nodes in K shows that p units can be pushed from K to the next S layer, and hence p units can be pushed to t. The argument for flow from s to v is symmetric. □

Note that these pushes to and from v saturate either all paths from s to v or all paths from v to t, and hence there can be at most $|S|$ such augmentations per phase.

3.2. Modified MPM algorithm. We will discuss the modifications to the MPM algorithm that allow fast implementation of pushes towards t. The modifications needed to implement pushes towards s are symmetric.

Using the definitions above, we modify the MPM algorithm in two ways: First, in each augmentation only S nodes are considered for the selection of the node of minimum potential. Second, every time flow is pushed along an edge (x, y), not only is the capacity $c(x, y)$ changed (as in the original MPM algorithm), but all the forward potentials and $\bar{c}$ values that are affected are also updated. That is, all capacities, potentials and pseudo-capacities are modified at each edge push, to preserve the

relations given in Definitions 3.1, 3.2 and 3.3. As in the original MPM algorithm, all node potentials and capacities are computed at the start of a phase, but since they are modified after each push, the potentials are not recomputed after each augmentation, as they are in the original algorithm. If that recomputation were done, the best time bound for a phase would again be $O(|S||T|)$. We will use the Charging lemma to bound the number of needed updates of potentials, capacities and pseudo-capacities in a phase.

LEMMA 3.2. *If the nodes of T have bounded degree d, then each phase can be executed in time $O(d|S|^2+d^2|T|)$.*

Proof. We must verify that the pushes can be executed and the node potentials, capacities and pseudo-capacities can be maintained in the above time bound. We first discuss the edge pushes. We assume that in each augmentation, flow is pushed from S-layer to S-layer in a breadth-first manner, although this is not necessary. Let x be an S-node that receives inflow of amount $a(x)$. Lemma 3.1 says there is always sufficient capacity to push $a(x)$ to the next S-layer, but does not indicate how to efficiently find the edges to use. However, with the updates described above, the $a(x)$ units can be pushed out of x in exactly the same greedy manner that is used in the original MPM algorithm, except that the pseudo-capacities play the role of the capacities. That is, $\bar{c}(x, y)$ is always the current amount that can be pushed from x through (x, y) and be guaranteed to reach the next S-layer. Hence the time needed for these pushes (not including time for updates) is (again) the number of edges in $G(F)$ plus the number of nonsaturating edge pushes. Since there are at most $d|T|$ edges in $G(F)$, and at most $|S|$ augmentations in a phase, the Charging lemma shows that the number of edge pushes in a phase is at most $O(|S|^2+d|T|)$.

Now we consider the cost of maintaining the edge capacities, the node potentials and the pseudo-capacities during a phase. A push from an S-node x along edge (x, y) only affects the potentials of x and y and the capacity of edge (x, y). A push from a T-node y to a node w only affects the potentials of y, the capacity of (y, w), the potentials of the S-nodes adjacent to y, and the pseudo-capacities of the associated edges. Hence, since each T-node has bounded degree d, at most $O(d)$ updates are required after any edge push. Then since the total number of edge pushes per phase is bounded by $O(|S|^2+d|T|)$, the number of potential, capacity and pseudo-capacity changes is bounded by $O(d|S|^2+d^2|T|)$.

Note that in order to find p for each augmentation, we also need the current values of $b(v)$ for each node v. These can be maintained in the same time bound that the forward potentials are, in a symmetric way. Hence the lemma is proved. □

Lemmas 3.1 and 3.2 and the phase lemma then yield the following:

THEOREM 3.1. *With the above modifications of the* MPM *algorithm, bounded degree bipartite flow can be computed in time $O(d|S|^3+d^2|S||T|)=O(|S|^3+|S||T|)$.*

The case of $d=2$. For $d=2$ (a common case) the modification of MPM can be viewed as merging length two paths between consecutive S-layers into a single edge. Then there are only $|S|$ nodes in the resulting graph, and clearly then the phase runs in $O(|S|^2+|T|)$ time. No such simple interpretation is known for arbitrary values of d. We note that for $|E|=O(|T|)$, it is easily shown that the Dinic algorithm runs in $O(|S|^2|T|)$ time on bipartite graphs.

4. Applications. We will now present several problems which can be solved using network flow in a bipartite graph where $|T| \gg |S|$.

4.1. Multiprocessor scheduling with release times and deadlines. We are given n jobs and m processors. The jth job has a release time r_j and deadline d_j such that the

job cannot be started before time r_j and must be completed by time d_j. In addition, the jth job has a processing time p_j which is the time required to complete the jth job. Each processor can run only one job at a time and each job can only be run on one processor at a time. We also assume that a job can be *preempted* at any time, and be resumed immediately on a new processor or later on any processor at no cost. Our goal is to find a schedule which completes all jobs subject to their release time and deadline restrictions.

Horn [14] showed that this problem can be formulated and solved as a network flow problem as follows. Let $t_1 \leqq t_2 \leqq \cdots \leqq t_k$ be the distinct times in the multiset $\{r_1, d_1, r_2, d_2, \cdots, r_n, d_n\}$. The time period between two successive t_i values will be called an *interval.* The flow network is used to determine how much of each job is to be completed within an interval. If an interval has length Δ, and q_j, $j = 1, 2, \cdots, n$, is the amount of job j to be completed within this interval, then these processing amounts can be completed if and only if

(1) $\max \{q_j, j = 1, 2, \cdots, n\} \leqq \Delta$, and

(2) $\sum_{j=1}^{n} q_j \leqq \Delta m$.

Once the q_j values are determined, the schedule for this interval can be constructed in $O(n)$ time [21].

To find the processing amounts within an interval, we construct a flow network with a node for each job and a node for each interval. The source is connected to the jth job node with an edge of capacity p_j. The jth job node has an edge to the ith interval node if $r_j \leqq t_i$ and $d_j \geqq t_{i+1}$, and this edge has capacity $t_{i+1} - t_i$. The ith interval node has an edge to the sink with capacity $m(t_{i+1} - t_i)$. See Fig. 2. The flow network described is bipartite, with $|S| = n$ and $|T| = k$. By Theorem 2.1 and the symmetric roles of S and T, we can find a maximum flow in $O(|S||T|^2) = O(nk^2)$ time. This is an improvement on the previous $O(n^3)$ time bound in the common case when k is $o(n)$.

4.2. Uniform processors. We will now generalize the previous scheduling problem by allowing the speeds of the processors to be different. Let the m processors have speeds $s_1 \geqq s_2 \cdots \geqq s_n$. A job with processing requirement p takes p/s_i units of time to be completed to the ith processor. As in the previous problem, once we have determined the amount of processing to be done on the jobs within an interval, the schedule for that interval can be constructed quickly. Gonzalez and Sahni [10] have

Job	p_j	r_j	d_j
1	4	0	6
2	8	0	14
3	3	2	5

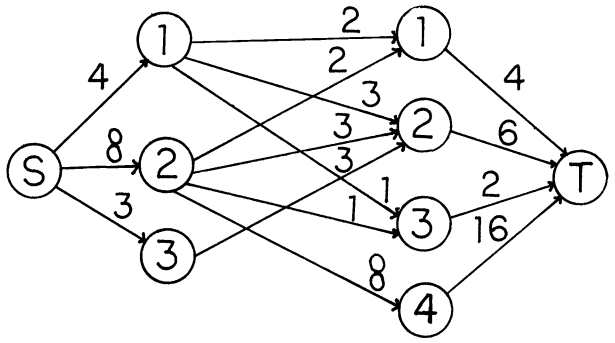


FIG. 2. *A flow network for a 2-processor, 3-job scheduling problem.*

shown that if $q_1 \geqq q_2 \cdots \geqq q_n$ are the amounts of processing to be done in an interval of length Δ, then a schedule can be constructed in $O(n \log m)$ time if and only if

$$(3)\quad \sum_{j=1}^{i} q_j \leqq \sum_{j=1}^{i} s_j \Delta, \text{ for each } i = 1, 2, \cdots, m-1, \text{ and}$$

$$(4)\quad \sum_{j=1}^{n} q_j \leqq \sum_{j=1}^{m} s_j \Delta.$$

Federgruen and Groenevelt [6] have shown that the amount of processing to be done within each interval can be found using the following flow network. As in the case for processors with identical speeds, there will be a source and n job nodes. However, we will have m *speed* nodes, $(i, 1), (i, 2), \cdots, (i, m)$, associated with the ith interval. The source is connected to the jth job node with an edge of capacity p_j. The jth job node is connected to node (i, k) if $r_j \leqq t_i$ and $d_j \geqq t_{i+1}$. This edge has capacity $(s_k - s_{k+1})(t_{i+1} - t_i)$, (where we let $s_{m+1} = 0$). Each speed node (i, r) is connected to the sink with an edge of capacity $(s_r - s_{r+1})(t_{i+1} - t_i)r$.

The number of intervals k can be as large as $2n-1$; so, there are $O(mn)$ speed nodes. A simple analysis of solving this network using the MPM algorithm results in an $O(m^3n^3)$ time bound. Federgruen and Groenevelt reduced this to $O(m^{2.5}n^3)$ using Cherkaski's flow algorithm [1]. However, since we have a bipartite flow network with $|S| = O(n)$ and $|T| = O(mn)$, Theorem 2.1 gives us a time bound of $O(|T||S|^2) = O(mn^3)$. In fact, Federgruen and Groenevelt show that if there are only r distinct processor speeds we can reduce the number of speed nodes to $O(rn)$ which gives us a time bound of $O(rn^3)$. Hence the bounds obtained using Theorem 2.1 of this paper beat the best previously established bounds.

4.3. Applications based on provisioning. A rich class of problems that can be modeled and solved as bipartite flow fall under the heading of *provisioning* or *shared fixed cost* problems. Picard and Queyranne [23] have shown that a large class of 0–1 integer programming problems can be solved this way. These problems are solved with a sequence of up to $|S|$ bipartite flow computations, each on a bipartite graph with $|S| \leqq |T| \leqq 2^{|S|}$. In that graph, each S node represents a variable in the integer programming problem, and each T-node represents a constraint. In general, $|T| \gg |S|$, although in most of the applications, the number of T-nodes is polynomial in $|S|$. For the class of problems defined in [23], Picard obtains the general time bound $O((|S|^3|T| + |T|^2|S|^2) \log^2 |T|)$ using complex flow algorithms. Theorem 2.1 in this paper shows that these problems can all be solved by MPM with F-heaps, or by Karzanov's algorithm in time $(|S|^3|T|)$. Below we will discuss one of the specific problems discussed in [23]: the maximum subgraph density problem.

4.3.1. Maximum subgraph density. Let $G = (V, A)$ be an undirected graph without parallel edges, where $|V| = n$, and $|A| = m$. Let $V' \subseteq V$ be a subset of vertices of V, and let $G(V')$ be the subgraph of G induced by V'. The *density* of $G(V')$ is defined as the number of edges of $G(V')$ divided by the number of nodes in V'. The maximum subgraph density problem is to find a set $V' \subseteq V$ to maximize the density of $G(V')$. The problem is of interest for its own sake, and has nondirect applications in problems of network vulnerability [11], [2], statistical clustering [19], [20], and decomposition of graphs into edge disjoint spanning trees [22], [29], [17]. Several generalizations of subgraph density involving weights on edges and nodes are introduced in [9]; we will discuss some of these below.

Surprisingly, the subgraph density problem can be solved efficiently, despite its similarity to the maximum clique problem. All of the proposed methods involved solving at most $|V|$ optimization problems called parametric problems; each has the following form: Given graph $G=(V, A)$ and a number λ, find the subset of vertices $W \subseteq V$ to minimize $\lambda|W|+|c(A(W))|$, where $c(A(W))$ is the set of edges in $A-A(W)$, and $A(W)$ is the set of edges in the subgraph $G(W)$ induced by W. The solution to the subgraph density problem is found by searching for the value of λ where the solution to the parametric problem is exactly $|A|$. Details of how the subgraph density problem reduces to these parametric problems are given in [23], [9] and omitted here. Each of the above parametric problems can be solved using network flow on either a bipartite or a nonbipartite graph. In either case, exactly the same sequence of parametric problems is solved. We will first describe a bipartite solution to the parametric problem found in [23], and then a nonbipartite solution derived from [2]. Other nonbipartite solutions appear or can be derived from [22], [9], [8].

In the bipartite solution, the graph $G(S, T)$ is constructed from G as follows. Node set S contains one node for every node in G, and node set T contains one node for every edge of G. A node u in S is connected to a node e in T if and only if u is an endpoint of the edge in G that e represents; each edge from S to T has infinite capacity. Node s is attached to each node in S, and each of these edges has capacity λ. Node t is attached to each node in T and each of these edges has capacity one. Then, the minimum s-t cut in $G(S, T)$ defines the optimal node set W: W contains every S node that is on the t side of the minimum s-t cut (see Fig. 3).

In the nonbipartite solution of the parametric problem, the graph NB is constructed by adding a source node, s, and a sink node, t, to the original graph G. There is one edge (s, v) connecting s to each node v in V; this edge has capacity λ. There is also one edge (v, t) connecting each node v to t; each such edge (v, t) has capacity $d(v)/2$,

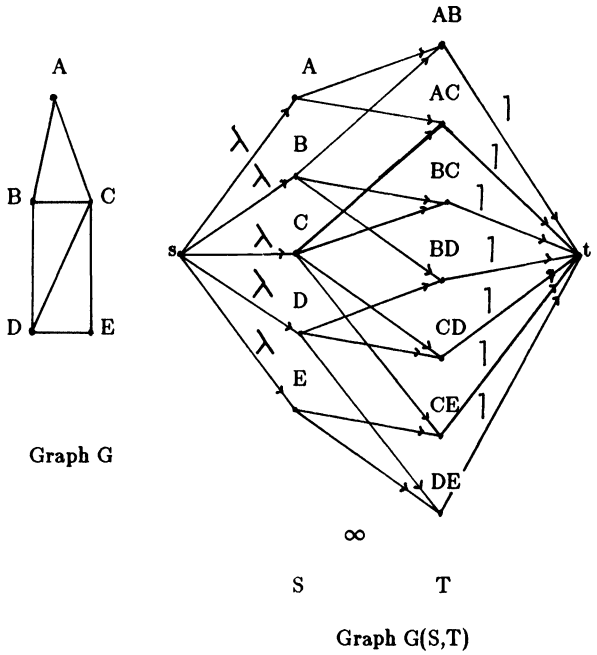


FIG. 3. *Subgraph density problem on G cast as a parametric network flow problem on a bipartite graph G(S, T). Each node in S corresponds to a node in G, and each node in T corresponds to an edge in G.*

where $d(v)$ is the degree of node v in G. Each original edge in G has capacity 1/2 (see Fig. 4). The optimal node set W is again the set of nodes of S that are on the t side of a minimum s-t cut.

It has been generally accepted in the literature that the subgraph density problem can be more efficiently solved using the nonbipartite graph NB than by using the bipartite graph $G(S, T)$. The implicit argument in the literature is that $|A| = \theta(|V|^2)$ in the worst case, so naive application of the general upper bounds on the running times of the Dinic, MPM and Karzanov algorithms on $G(S, T)$ yield guaranteed times (for each flow computation) of no better than $O(|V|^8)$, $O(|V|^6)$, and $O(|V|^6)$, respectively, while when run on NB, the corresponding upper bounds are the square root of the above bounds for $G(S, T)$. This argument, based on naive application of upper bounds, is strengthened by considering more complex algorithms [25], [26] which take advantage of the sparsity of $G(S, T)$ ($G(S, T)$ is sparse even when $|A| = \theta(|V|^2)$); naive application of the upper bounds gives a guaranteed worst case running time of $O(|V|^4 \log |V|)$ (reduced to $O(|V|^3 \log |V|)$ using the Phase lemma) for the bipartite graph, so even sophisticated algorithms naively appear to run slower on the bipartite graph than do the simpler algorithms on the nonbipartite graph.

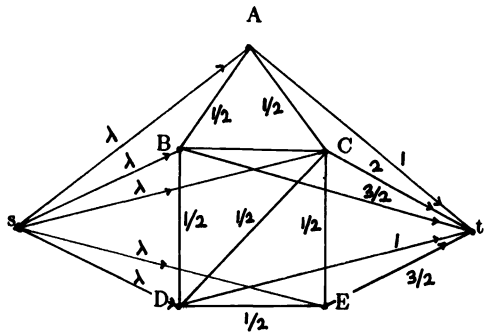


FIG. 4. *Subgraph density problem on G cast as a parametric network flow problem on the nonbipartite graph NB.*

The results of § 3 refute the above reasoning. In $G(S, T)$ every node in T has bounded degree two, and so both the Dinic and the modified MPM algorithms run as fast ($O(|V|^4)$, $O(|V|^3)$) on $G(S, T)$ as those algorithms run on NB. Further, it can be shown, using ideas similar to those in this paper, that the unmodified Karzanov algorithm also runs as fast ($O(|V|^3)$) on $G(S, T)$ as it does on NB. Hence, the accepted argument for the superiority of the nonbipartite model is incorrect, although we do not know which model actually permits more efficient solutions.

4.3.2. Weighted subgraph density problems. Several generalizations of the subgraph density problem were proposed in [9]; we mention here an observation that reduces the running time of the solutions. The most general density problem is that each edge e has a (real) weight $w(e)$ and each node v has a (real) weight $w(v)$, and the objective is to find a subset V' of the vertices to maximize the ratio $w(A(V'))/w(V')$, where $w(V')$ is the total weight of the vertices in V', and $w(A(V'))$ is the total weight of the edges in the graph induced by V'. Note that the subgraph density problem is just the problem where the weight of each edge and vertex is one.

The weighted density problem is again solved with a sequence of parametric optimization problems, each of the form: given λ, find the subset of vertices $W \subseteq V$

to minimize $\lambda w(W) + w[c(A(W))]$. Each of these problems is solved by network flow in a graph identical to $G(S, T)$ or NB, except for the edge capacities; in $G(S, T)$, each edge (s, v) has capacity $\lambda w(v)$, and each edge (e, t) has capacity $w(e)$.

The generalized density problem is solved by finding the value λ^* of λ for which the parametric problem has solution $w(A)$. The key issue then is how to vary λ to home in on λ^*. Goldberg [9] gives a method which never uses more than $O(|V|^3)$ trial values of λ, and hence finds the optimal V' in $O(|V|^6)$ arithmetic operations (on reals). We note here that at most $|V|$ trial values of λ are ever needed, and hence the weighted density problem can be solved in $O(|V|^4)$ time. This follows from a result in [4] that as λ varies from 0 to infinity, there can only be $|V|$ values of λ where the set of edges in the minimum cut changes. Further, the value of the minimum cut as a function of λ can be completely determined (traced out) in $O(|V|^4)$ time [4]; after that, λ^* can be found trivially. A similar theorem [27] applies to minimum cuts in the nonbipartite graph NB. Other weighted problems (over rationals) are also discussed in [9] and their solutions can similarly be sped up with the above observation.

4.4. Additional applications. We briefly sketch several more applications of flow in a bipartite graph with node sets S and T, where $|T| \gg |S|$ either as a structural feature of the model (as in subgraph density) or in frequent problem instances. Additional applications are mentioned in [12].

1. Find a maximum size set of edge disjoint spanning trees in an undirected graph [29], [17], [22], [23]: In this application $|T| = \theta((|S|)^2)$, and $d = 2$.

2. Network vulnerability [11], [2]: Given a connected graph G, and a set of edges W in G, define $c(W)$ as the number of additional components created by deleting edge set W from G. Find the best edge set W to maximize the ratio $c(W)/|W|$. Here again, $|T| = \theta(|S|^2)$, and $d = 2$.

3. Partitioning a data-base between fast and slow memory [4]: The data base consists of n pieces of data which will be distributed between fast (expensive) memory and slow (cheaper) memory. There are m possible queries that will be made on the data base, each defined by a subset of the data. A cost $c(i)$ is incurred for storing data i in fast memory, and a cost $c(j)$ is incurred when query j is made and some of the data needed for query j is in slow memory. Determine the minimum cost partition of the data. $|T| = m$ and $|S| = n$, and typically $m \gg n$.

4. Sportswriter's End-of-Season Problem [28], [15]: Given the win-loss record of a set of n teams in a league, and the schedule of the remaining games, determine whether there is an outcome of the remaining games so that a given team x wins the pennant[1] (wins more games than any other team). Here again, $|T| = \theta(|S|^2)$ and $d = 2$.

Acknowledgment. We would like to thank Gregory Sullivan at Yale for helpful and insightful comments during the middle stages of this work.

REFERENCES

[1] B. CHERKASKI, *An algorithm for the construction of maximal flow in networks with complexity* $O(|V|^2|E|^{1/2})$ *operations*, Math. Methods of Solutions of Economic Problems, 7 (1977), pp. 117-125 (in Russian).

[2] W. H. CUNNINGHAM, *Optimal attack and reinforcement of a network*, J. Assoc. Comput. Mach., 32 (1985), pp. 549-561.

[3] S. EVEN, *Graph Algorithms*, Computer Science Press, 1979.

[1] A harder problem of computing the probability that x wins is NP-hard [13].

[4] M. J. EISNER AND D. G. SEVERANCE, *Mathematical techniques for efficient record segmentation in large shared databases*, J. Assoc. Comput. Mach., 23 (1976), pp. 619-635.
[5] L. FORD AND D. FULKERSON, *Flow in Networks*, Princeton University Press, Princeton, NJ, 1962.
[6] A. FEDERGRUEN AND H. GROENEVELT, *Preemptive scheduling of uniform machines by ordinary network flow techniques*, preprint.
[7] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci Heaps and their uses in improved network optimization*, 25th Annual Symposium on Foundations of Computer Science, IEEE, 1984.
[8] S. FUJISHIGE, *Lexicographically optimal base of a polymatroid with respect to weight vector*, Math. Oper. Res., 5(1980), pp. 186-196.
[9] A. V. GOLDBERG, *Finding a maximum density subgraph*, Technical Report UCB/CSD 84/171, Computer Science Division, University of California, Berkeley, May 1984.
[10] T. GONZALEZ AND S. SAHNI, *Preemptive scheduling of uniform processor systems*, J. Assoc. Comput. Mach., 25 (1978), pp. 92-101.
[11] D. GUSFIELD, *Connectivity and edge disjoint spanning trees*, Inform. Process. Lett., February 26, 1983.
[12] D. GUSFIELD, C. MARTEL AND D. FERNANDEZ-BACA, *Fast algorithms for bipartite network flow*, Yale Computer Science Technical Report TR-356, Yale Univ., New Haven, CT, January 1985.
[13] D. GUSFIELD AND G. SULLIVAN, *Determining possible end-of-season rankings*, Yale Technical Report, New Haven, CT, in preparation.
[14] W. HORN, *Some simple scheduling algorithms*, Naval Res. Logist. Quart., 21 (1974), pp. 177-185.
[15] A. J. HOFFMAN AND T. J. RIVLIN, *When is a team mathematically eliminated?*, Symp. on Mathematical Programming, Princeton University Press, Princeton, NJ, 1970.
[16] T. C. HU, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA, 1982.
[17] C. ST. J. A. NASH-WILLIAMS, *Edge disjoint spanning trees graphs*, J. London Math. Soc., 36 (1961), pp. 445-450.
[18] V. M. P. KUMAR MALHOTRA AND S. MAHESHIVARI, *An $O(|V|^3)$ algorithm for finding maximum flows in networks*, Inform. Process. Lett., 7 (1978), pp. 277-278.
[19] D. MATULA, *K-components, clusters, and slicings in graphs*, SIAM J. Appl. Math., 22 (1972), pp. 459-480.
[20] ———, *Graph theoretic techniques for cluster analysis algorithms*, in Classification and Clustering, Academic Press, New York, 1977.
[21] R. MCNAUGHTON, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1-12.
[22] J. PICARD AND M. QUEYRANNE, *Selected applications of minimum cuts in networks*, INFOR, 20 (Nov. 1982), pp. 394-422.
[23] ———, *A network flow solution of some nonlinear 0-1 programming problems and applications to graph theory*, Networks, 12 (1982), pp. 141-159.
[24] J. RHYS, *A selection problem of shared fixed costs and network flows*, Management Sci., 17 (1970), pp. 200-207.
[25] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comput. System Sci., 24 (1983), pp. 362-391.
[26] ———, *Self-adjusting binary search trees*, J. Assoc. Comput. Mach., 32 (1985), pp. 652-686.
[27] H. S. STONE, *Critical load factors in two-processor distributed systems*, IEEE Trans. Software Engrg., 3 (1978), pp. 254-258.
[28] B. L. SCHWARTZ, *Possible winners in partially completed tournaments*, SIAM Rev., 8 (1966), pp. 302-308.
[29] W. T. TUTTE, *On the problem of decomposing a graph into connected factors*, J. London Math. Soc., 36 (1961), pp. 221-230.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
004

TENSOR RANK AND BORDER RANK OF BAND TOEPLITZ MATRICES*

D. BINI† AND M. CAPOVANI‡

Abstract. Let $\mathcal{B}_{n,h,k}$ be the class of $n \times n$ band Toeplitz matrices $A = (a_{i,j})$ such that $a_{i,j} = 0$ if $i - j \geqq k$ or $j - i \geqq h$, $k \leqq h$ and $\mathcal{S}_{n,k}$ the subclass of $\mathcal{B}_{n,k,k}$ made up by symmetric matrices. We show that $\mathrm{rk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) = n + h - 1$ if the field $\mathbf{F}$ contains a primitive $(n-1+h)$th root of unity, $\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) = n + k - 1$, $\mathrm{rk}_{\mathbf{R}}(\mathcal{S}_{n,k}) = \mathrm{brk}_{\mathbf{R}}(\mathcal{S}_{n,k}) = n + 2\lfloor (k-1)/2 \rfloor$ if $\mathbf{R}$ is the real field and $k \leqq n/2$.

Key words. tensor rank, border rank, Toeplitz matrices, computational complexity

AMS(MOS) subject classifications. 68C25, 15A57

1. Introduction. The concept of tensor rank of a set of $m \times n$ matrices $\{A_k, k = 1, 2, \cdots, p\}$, $A_k = (a_{i,j,k})$, or equivalently, of a three-way array $A = (a_{i,j,k})$, introduced by Hitchcock in 1927 [19], has had a great development in the framework of algebraic complexity. This number is related to the computational complexity of a set of bilinear forms $f_k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A_k \mathbf{y}$, $k = 1, 2, \cdots, p$, where $\mathbf{x}$, $\mathbf{y}$ are vectors of noncommuting indeterminates. We have in fact that the least number of nonscalar multiplications to compute the functions $f_k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A_k \mathbf{y}$, $k = 1, 2, \cdots, p$, coincides with the tensor rank of $\{A_1, A_2, \cdots, A_p\}$ [30]. General properties of tensor rank have been investigated in several papers [2], [20], [23]. Recently the concept of border rank has been introduced for describing the complexity of approximating algorithms [9], [12] and has been widely used to construct fast algorithms for matrix multiplication [7], [12], [13], [27], [28]. General properties of border rank have been analyzed in [1], [25], [29].

The knowledge of the value of the tensor rank and the border rank of specific classes of matrices, besides yielding the complexity of the evaluation, or approximation, of the corresponding set of bilinear forms, can be used for getting insights into more general properties of these functions. The case of $p \times q \times 2$ tensors is analyzed in [4] and [22], properties of $p \times p \times 3$ tensors can be found in [29], the tensors associated to the problem of quaternion product and vector product are analyzed in [16], [21] and [17], respectively, properties of tensors related to certain associative algebras are exploited in [15], the tensor rank of Toeplitz matrices (a matrix $A = (a_{i,j})$ is a Toeplitz matrix if $a_{i,j} = \alpha_{i-j}$) is investigated in [24], while in [31] the case of symmetric Toeplitz matrices is considered.

Recently the concept of tensor rank has been implicitly used for the construction of fast algorithms to compute the eigenvalues of banded Toeplitz matrices [6], [8] and the concept of border rank has been used for the fast parallel inversion of triangular Toeplitz matrices [11]. This nonstandard use of tensor and border rank is described in detail in [5].

In this paper we investigate the tensor and border rank of special classes of Toeplitz matrices. We consider the class $\mathcal{B}_{n,h,k}$ of banded Toeplitz matrices $A = (a_{i,j})$ defined by the condition $a_{i,j} = 0$ if $i - j \geqq k$ or $j - i \geqq h$, and the subclass $\mathcal{S}_{n,k}$ of $\mathcal{B}_{n,k,k}$ of symmetric matrices. By using the embedding technique, that is, looking for a class of (approximatively) simultaneously diagonalizable matrices which have a suitable submatrix generating $\mathcal{B}_{n,h,k}$ or $\mathcal{S}_{n,k}$, we can give upper bounds to the tensor (border) rank. The classes in which we embed $\mathcal{B}_{n,h,k}$ and $\mathcal{S}_{n,k}$ are the class of circulant matrices

* Received by the editors April 26, 1984; accepted for publication (in revised form) April 11, 1986.
† Dipartimento di Matematica, Università di Pisa, Via Buonarroti, 2, 56100 Pisa, Italia.
‡ Dipartimento di Informatica, Università di Pisa, Corso Italia, 40, 56100 Pisa, Italia.

[14], the class of upper triangular Toeplitz matrices, and a special algebra of $n \times n$ matrices, the class τ_n, defined by the condition $A=(a_{i,j}) \in \tau_n$ if $a_{i+1,j}+a_{i-1,j}=a_{i,j+1}+a_{i,j-1}$, $i,j=1,2,\cdots,n$, where we assume $a_{i,0}=a_{0,j}=a_{n+1,j}=a_{i,n+1}=0$. Lower bounds to the tensor and border rank are obtained by using some known criteria [10], [29].

We show that $\mathrm{rk}_{\mathbf{R}}(\mathcal{S}_{n,k})=\mathrm{brk}_{\mathbf{R}}(\mathcal{S}_{n,k})=n+2\lfloor (k-1)/2 \rfloor$, $k \leqq n/2$, where $\mathbf{R}$ is the real field and $\mathrm{rk}_{\mathbf{F}}$, $\mathrm{brk}_{\mathbf{F}}$ means tensor rank, respectively, border rank, over the field $\mathbf{F}$. Moreover if $k > n/2$ we have $\mathrm{rk}_{\mathbf{R}}(\mathcal{S}_{n,k}) \leqq n+2\lfloor (k-1)/2 \rfloor$. Furthermore we show that $\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,h,k})=n+\min(h,k)-1$ over any field $\mathbf{F}$ and $\mathrm{rk}_{\mathbf{F}}(\mathcal{B}_{n,h,k})=n+\max(h,k)-1$, if $\mathbf{F}$ contains a primitive mth root of unity, $m=n+\max(h,k)-1$. In particular, for the class $\mathcal{B}_{n,n,k}$ of Toeplitz matrices in k-Hessenberg form, we have $\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,n,k})=n+k-1$.

Such results yield fast algorithms for the computation of the eigenvalues and for the parallel inversion of banded Toeplitz matrices [5].

2. Preliminaries. Let $\mathbf{F}$, $\mathbf{G}$ be numeric fields such that $\mathbf{F} \subseteq \mathbf{G}$. We denote $\mathbf{F}[\lambda]$, $\mathbf{F}(\lambda)$ the ring of polynomials and the field of rational functions in λ over $\mathbf{F}$, respectively. Let $A_k=(a_{i,j,k})$, $a_{i,j,k} \in \mathbf{F}$, $k=1,2,\cdots,p$, be a set of p linearly independent $n \times n$ matrices and $\mathcal{A}$ be the linear space spanned by this set over $\mathbf{F}$. We denote by $\mathbf{e}_k$ the kth column of the $n \times n$ identity matrix.

The *tensor rank* of $\mathcal{A}$ over $\mathbf{G}$ is the least integer t such that there exist matrices $U=(u_{i,r})$, $V=(v_{j,r})$, $W=(w_{k,r})$ of dimension $n \times t$, $n \times t$, $p \times t$, respectively, with elements in $\mathbf{G}$ and

$$a_{i,j,k}=\sum_{r=1}^{t} u_{i,r}v_{j,r}w_{k,r}.$$

We set $t=\mathrm{rk}_{\mathbf{G}}(\mathcal{A})$. Observe that if $p=1$ then the concept of tensor rank coincides with the concept of rank of a matrix.

The *border rank* of $\mathcal{A}$ over $\mathbf{G}$ is the least integer t for which there exist $n \times n$ matrices $E_k(\lambda)$, $k=1,2,\cdots,p$, with elements in $\mathbf{G}[\lambda]$ such that $E_k(0)=0$ and $\mathrm{rk}_{\mathbf{G}(\lambda)}(\mathcal{A}(\lambda))=t$, where $\mathcal{A}(\lambda)$ is the linear space spanned by $A_k+E_k(\lambda)$, $k=1,2,\cdots,p$, over $\mathbf{G}(\lambda)$. We set $t=\mathrm{brk}_{\mathbf{G}}(\mathcal{A})$. We have $\mathrm{rk}_{\mathbf{G}}(\mathcal{A}) \geqq \mathrm{brk}_{\mathbf{G}}(\mathcal{A})$.

Lower bounds to the tensor rank and border rank can be obtained by using the following propositions.

PROPOSITION 1 [29]. *If $I \in \mathcal{A}$ then, for any $A, B \in \mathcal{A}$ we have*

$$\mathrm{brk}_{\mathbf{G}}(\mathcal{A}) \geqq n+\tfrac{1}{2}\mathrm{rk}(AB-BA).$$

PROPOSITION 2 [10]. *We have*

$$\mathrm{brk}_{\mathbf{F}}(\mathcal{A}) \geqq p-1+\min\{\mathrm{rk}(A): A \in \mathcal{A}, A \neq 0\}.$$

PROPOSITION 3 [10]. *If for any basis $A_1, A_2, \cdots, A_p$ of the linear space $\mathcal{A}$ there exists an index k such that $\mathrm{rk}(A_k) \geqq q$ then*

$$\mathrm{rk}_{\mathbf{F}}(\mathcal{A}) \geqq p-1+q.$$

In order to give upper bounds to the tensor rank and border rank of certain classes of Toeplitz matrices, we shall use the following classes and the following results.

Class τ_n. Let τ_n be the linear space, over $\mathbf{F}$, of all of the $n \times n$ matrices $A=(a_{i,j})$ satisfying the condition

$$\begin{aligned} &a_{i,j+1}+a_{i,j-1}=a_{i+1,j}+a_{i-1,j}, \\ &\qquad\qquad\qquad\qquad i,j=1,2,\cdots,n. \\ &a_{0,j}=a_{i,0}=a_{n+1,j}=a_{i,n+1}=0, \end{aligned} \tag{2.1}$$

According to relation (2.1), knowing the first row (or column) of any matrix $A \in \tau_n$ it is possible to build up all the elements of the matrix. Moreover (2.1) can be written in compact form as $AH - HA = 0$, where $H = (h_{i,j})$ is the $n \times n$ matrix defined by $h_{i,i+1} = h_{i+1,i} = 1$, $i = 1, 2, \cdots, n-1$, $h_{i,j} = 0$ otherwise. We call condition (2.1) *cross-sum condition.*

A special basis of the linear space τ_n, in which the matrices of the basis have elements in $\{0, 1\}$, is given by the set $\{A_k \in \tau_n: A_k \mathbf{e}_1 = \mathbf{e}_k, k = 1, 2, \cdots, n\}$. Observe that the first column of A_k is the kth column of the identity matrix and the other elements can be obtained by means of the cross-sum condition.

Below we show a representation of a matrix $A \in \tau_n$ in the basis $\{A_k, k = 1, 2, \cdots, n\}$ in the case $n = 4$

$$A = \sum_{k=1}^{n} a_k A_k = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_2 & a_1 + a_3 & a_2 + a_4 & a_3 \\ a_3 & a_2 + a_4 & a_1 + a_3 & a_2 \\ a_4 & a_3 & a_2 & a_1 \end{bmatrix}.$$

It is easy to show that any matrix belonging to τ_n is symmetric and persymmetric (i.e., symmetric with respect to the secondary diagonal).

Since H has simple real eigenvalues, if $\mathbf{F}$ is the real field $\mathbf{R}$, the class τ_n is the algebra generated by H over $\mathbf{R}$ (see [18, p. 223]). Therefore another basis of τ_n is given by $\{I, H, H^2, \cdots, H^{n-1}\}$. Moreover it is possible to give the following spectral decomposition to H:

$$H = QDQ,$$

$$D = \operatorname{Diag}(d_1, d_2, \cdots, d_n), \qquad d_i = 2\cos(\pi i/(n+1)),$$

$$Q = (\sqrt{2/(n+1)} \sin(\pi ij/(n+1))),$$

where Q is orthogonal and symmetric.

PROPOSITION 4. *If $A_k \in \tau_n$ is such that $A_k \mathbf{e}_1 = \mathbf{e}_k$ then we have*

$$A_k = \sum_{r=1}^{n} w_{k,r} \mathbf{u}_r \mathbf{u}_r^T,$$

$$\mathbf{u}_r = (u_{i,r}), \qquad u_{i,r} = \sqrt{2/(n+1)} \sin(\pi ir/(n+1)),$$

$$w_{k,r} = P_{k-1}(2\cos(\pi r/(n+1))),$$

where $P_k(x)$, $k = 0, 1, 2, \cdots, n-1$, are essentially the Chebyshev polynomials defined by

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_{k+1}(x) = xP_k(x) - P_{k-1}(x).$$

Moreover $\operatorname{rk}_{\mathbf{R}}(\tau_n) = n$.

Proof. The result holds in view of the spectral decomposition of H and by the relation

$$A_k = P_{k-1}(H), \qquad k = 1, 2, \cdots, n,$$

which can be easily verified by induction.

Circulant matrices. Let $C = (c_{i,j})$ be the $n \times n$ matrix defined by $c_{i,i+1} = 1$, $i = 1, 2, \cdots, n-1$, $c_{n,1} = 1$, $c_{i,j} = 0$ otherwise. The linear space $\mathscr{C}_n$ spanned by the matrices C^k, $k = 0, 1, \cdots, n-1$, over $\mathbf{F}$, is the set of the circulant matrices [14]. In fact $\mathscr{C}_n$ is isomorphic to the group algebra $\mathbf{F}\mathbf{Z}_n$ of the cyclic groups of order n. The following result is a special case of the Wedderbun Structure Theorem on (split) semisimple group algebras.

PROPOSITION 5 [14]. *Suppose that F contains a primitive nth root ω of unity, then*

$$C^{k-1} = \sum_{r=1}^{n} w_{k,r}\mathbf{u}_r\mathbf{u}_r^H, \qquad k = 1, 2, \cdots, n,$$

where $\mathbf{u}_r = (u_{i,r})$, $u_{i,r} = \sqrt{1/n}\,\omega^{(i-1)(r-1)}$, *and* $w_{k,r} = \omega^{(k-1)(r-1)}$.

Upper triangular Toeplitz matrices. Let $T = (t_{i,j})$ be the $n \times n$ matrix defined by $t_{i,i+1} = 1$, $i = 1, 2, \cdots, n-1$, $t_{i,j} = 0$ otherwise. The linear space $\mathcal{T}_n$ spanned over $\mathbf{F}$ by the matrices T^k, $k = 0, 1, \cdots, n-1$, is the set of upper triangular Toeplitz matrices. The following result holds:

PROPOSITION 6. *We have*

$$\mathrm{rk}_{\mathbf{F}}(\mathcal{T}_n) \geqq 2n-1, \quad \mathrm{rk}_{\mathbf{Q}}(\mathcal{T}_n) = 2n-1, \quad \mathrm{brk}_{\mathbf{F}}(\mathcal{T}_n) = n,$$

where $\mathbf{Q}$ *is the field of rational numbers.*

Proof. Applying Proposition 3 with $q = n$ yields the lower bound to the tensor rank over any field. From [24] we have the upper bound $2n-1$ over $\mathbf{Q}$. From Proposition 2 we get n as lower bound to the border rank. In order to show that $\mathrm{brk}_{\mathbf{F}}(\mathcal{T}_n) \leqq n$ observe that the matrix

$$T(\lambda) = T + \mathrm{Diag}(\lambda, \lambda^2, \cdots, \lambda^n),$$

is diagonalizable over $\mathbf{F}(\lambda)$ under a similarity transformation; that is, there exists a matrix U with elements in $\mathbf{F}(\lambda)$ such that

$$U^{-1}T(\lambda)U = \mathrm{Diag}(\lambda, \lambda^2, \cdots, \lambda^n).$$

Therefore, setting $E_k(\lambda) = T^k - T(\lambda)^k$, $w_{k,r} = \lambda^{rk}$, $V = (U^{-1})^T$, we get

$$T^k = \left(\sum_{r=1}^{n} w_{k,r}u_{i,r}v_{j,r} \right) + E_k(\lambda),$$

that is $\mathrm{brk}_{\mathbf{F}}(\mathcal{T}_n) \leqq n$.

3. Main results. Let $\mathcal{B}_{n,h,k}$ be the linear space of $n \times n$ Toeplitz matrices $A = (a_{i,j})$ such that $a_{i,j} = 0$ if $i - j \geqq k$ or $j - i \geqq h$. Any matrix belonging to $\mathcal{B}_{n,h,k}$ is called a band Toeplitz matrix. In the case $k = h$ the band is balanced and we call such a matrix a $(2k-1)$-diagonal Toeplitz matrix. If $h = n$ we have the set of Toeplitz matrices which are also in k-Hessenberg form. Let $\mathcal{S}_{n,k}$ be the subspace of $\mathcal{B}_{n,k,k}$ made up by symmetric matrices. In this section we determine the tensor rank and the border rank of the classes $\mathcal{B}_{n,h,k}$ and $\mathcal{S}_{n,k}$.

Class $\mathcal{S}_{n,k}$. In order to give lower bounds to the border rank of this class of matrices by means of Proposition 1, we want to analyze the structure of the linear space spanned by the matrices $C(A, B) = AB - BA$, $A, B \in \mathcal{S}_{n,k}$. For this purpose we have the following result.

LEMMA 1. *Let* $k \leqq n/2$. *The linear space spanned by the matrices* $C(A, B) = AB - BA$, $A, B \in \mathcal{S}_{n,k}$, *is made up by matrices* S *with the following structure*

$$S = \mathrm{Diag}(S_{k-1}, 0, -S_{k-1}^{\tau}),$$

where S_{k-1} *is any* $(k-1) \times (k-1)$ *skew-symmetric matrix (i.e.* $S_{k-1} = -S_{k-1}^T$*) and* S_{k-1}^{τ} *denotes the transposition of* S_{k-1} *with respect to the secondary diagonal.*

Proof. Consider the following basis of $\mathcal{S}_{n,k}$: $A_i = T^i + T^{iT}$, $i = 0, 1, \cdots, k-1$, where $T = (t_{i,j})$ is such that $t_{i,i+1} = 1$, $i = 1, 2, \cdots, n-1$, $t_{i,j} = 0$ otherwise. We have $C(A_i, A_j) = B_{i,j} - B_{i,j}^T$, where $B_{i,j} = T^iT^{jT} - T^{jT}T^i$, $i, j = 1, 2, \cdots, n-1$. By a direct computation we have, assuming $i \geqq j$, that $B_{i,j}$ has exactly $2j$ nonzero entries, which are all along the

line $\{(r, s)\colon s-r=i-j\}$; those meeting the first j rows (resp. last j columns) are equal to $+1$ (resp. -1). All other entries are zero. For the case $j>i$ observe that $B_{j,i}=B_{i,j}^T$. Therefore the matrices $B_{i,j}-B_{i,j}^T$ span the linear space generated by the matrices $S=\mathrm{Diag}\,(S_{k-1}, 0, -S_{k-1}^T)$, where S_{k-1} is any $(k-1)\times(k-1)$ skew-symmetric matrix.

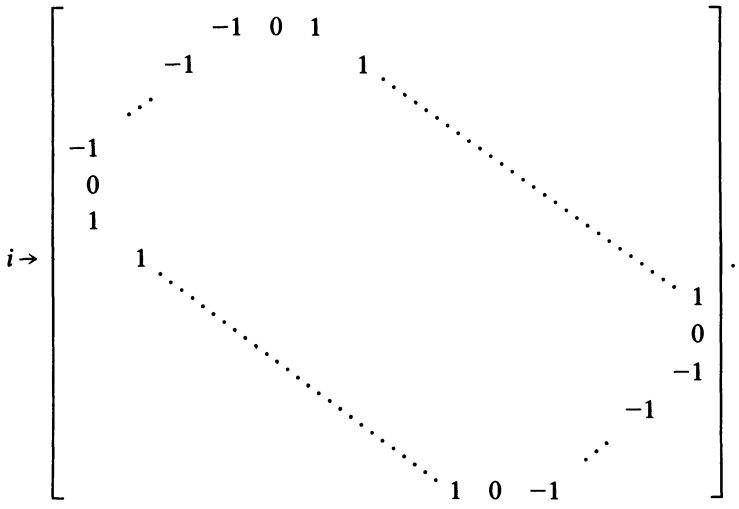
Since the maximum rank of an $m\times m$ skew-symmetric matrix is m if m is even, and $m-1$ if m is odd [26], we have that the maximum rank of the matrices belonging to the linear space spanned by $AB-BA$, $A, B\in\mathcal{S}_{n,k}$ is $4\lfloor(k-1)/2\rfloor$. Therefore applying Proposition 1 yields the following result.

PROPOSITION 7. *If $k\leqq n/2$ then we have*

$$\mathrm{brk}_{\mathbf{F}}(\mathcal{S}_{n,k})\geqq n+2\lfloor(k-1)/2\rfloor.$$

Now consider the matrices $A_i\in\tau_m$, $i=1,2,\cdots,k$, $m=n+2\lfloor(k-1)/2\rfloor$, such that the first column of A_i is given by $A_i\mathbf{e}_1=\mathbf{e}_i-\mathbf{e}_{i-2}$, $i=1,2,\cdots,k$, where we assume $\mathbf{e}_j=\mathbf{0}$ if $j\leqq 0$.

Let us observe that, by means of the cross-sum condition, any matrix A_i, $i\geqq 3$, has the following structure



Therefore the submatrices obtained by deleting the first and the last $\lfloor(k-1)/2\rfloor$ rows and columns of each of $A_1, A_2, \cdots, A_k$ are a basis of the linear space $\mathcal{S}_{n,k}$. Hence we can apply Proposition 4 to get an upper bound to the tensor rank of $\mathcal{S}_{n,k}$.

PROPOSITION 8. *We have*

$$\mathrm{rk}_{\mathbf{R}}(\mathcal{S}_{n,k})\leqq n+2\lfloor(k-1)/2\rfloor:$$

moreover, if $k\leqq n/2$ then

$$\mathrm{rk}_{\mathbf{R}}(\mathcal{S}_{n,k})=\mathrm{brk}_{\mathbf{R}}(\mathcal{S}_{n,k})=n+2\lfloor(k-1)/2\rfloor.$$

Proof. Let $A_i\in\tau_m$ be such that $m=n+2\lfloor(k-1)/2\rfloor$, $A_i\mathbf{e}_1=\mathbf{e}_i-\mathbf{e}_{i-2}$, where we assume $\mathbf{e}_j=\mathbf{0}$ if $j\leqq 0$. Let $\tilde{A}_i$ be the submatrix of A_i obtained by deleting the first and the last $\lfloor(k-1)/2\rfloor$ rows and columns. Then, for Proposition 4, we have

$$A_i=\sum_{r=1}^{m}(w_{i,r}-w_{i-2,r})\mathbf{u}_r\mathbf{u}_r^T,$$

$$\mathbf{u}_r=(u_{i,r}),\quad u_{i,r}=\sqrt{2/(m+1)}\sin(\pi ir/(m+1)),\quad w_{i,r}=P_{i-1}(2\cos(\pi r/(m+1))),$$

where $P_i(x)$ is the Chebyshev polynomial of degree i defined in Proposition 4. Therefore we have

$$\tilde{A}_i = \sum_{r=1}^{m} (w_{i,r} - w_{i-2,r})\tilde{\mathbf{u}}_r \tilde{\mathbf{u}}_r^T, \qquad i = 1, 2, \cdots, k,$$

$$\tilde{\mathbf{u}}_r = (\tilde{u}_{i,r}), \quad \tilde{u}_{i,r} = u_{i+\lfloor (k-1)/2 \rfloor, r}, \quad i = 1, 2, \cdots, k, \quad r = 1, 2, \cdots, m.$$

Class $\mathcal{B}_{n,h,k}$. Let us suppose $k \leqq h$. We can give a lower bound to $\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,h,k})$ by observing that any nonzero matrix $A \in \mathcal{B}_{n,h,k}$ has a nonsingular $q \times q$ submatrix, $q \geqq n - h + 1$, made up by the first q columns and the last q rows. Therefore, by Proposition 2, we get the following result.

PROPOSITION 9. *We have*

$$\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) \geqq n + k - 1.$$

We can show that $n + k - 1$ is also an upper bound to the border rank of $\mathcal{B}_{n,h,k}$, by using the embedding technique. In fact observe that any matrix belonging to $\mathcal{B}_{n,h,k}$ can be obtained by deleting the first $k-1$ columns and the last $k-1$ rows of a suitable $(n+k-1) \times (n+k-1)$ upper triangular Toeplitz matrix, as it is shown in the following example, in which $n = 5$, $k = 3$ and $h = 4$:

$$\left[\begin{array}{cc|ccccc} a & b & c & d & e & f & 0 \\ & a & b & c & d & e & f \\ & & a & b & c & d & e \\ & & & a & b & c & d \\ & & & & a & b & c \\ \hline & & & & & a & b \\ & & & & & & a \end{array}\right].$$

Therefore, from Proposition 6, we get the following result.

PROPOSITION 10. *We have*

$$\mathrm{brk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) = n + k - 1.$$

Applying Proposition 3 with $q = n + 1 - k$ yields the following lower bound:

PROPOSITION 11. *We have*

$$\mathrm{rk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) \geqq n + h - 1.$$

Embedding the class $\mathcal{B}_{n,h,k}$ into $\mathcal{C}_{n+h-1}$ we can prove the following.

PROPOSITION 12. *Let* $\mathbf{F}$ *contain a primitive* m*th root of unity* ω, $m = n + h - 1$; *then we have*

$$\mathrm{rk}_{\mathbf{F}}(\mathcal{B}_{n,h,k}) = n + h - 1.$$

Moreover if $\tilde{A}_i$, $i = 1, 2, \cdots, h + k - 1$, *is the basis of* $\mathcal{B}_{n,h,k}$ *such that* $\tilde{A}_i$ *has null elements except the elements of indices* (r, s) *such that* $r - s = i - 1$ *if* $1 \leqq i \leqq k$ *and* $s - r = i - k$ *if* $k + 1 \leqq i \leqq h + k - 1$, *then*

$$\tilde{A}_i = \sum_{r=1}^{m} w_{i,r} \mathbf{u}_r \mathbf{u}_r^H,$$

where $\mathbf{u}_r = (u_{i,r})$ *is such that* $u_{i,r} = \sqrt{1/m}\,\omega^{(i+h-1)(r-1)}$ *and* $w_{i,r} = \omega^{(i-1)(r-1)}$.

Proof. For the upper bound observe that if $A_i \in \mathcal{C}_{n+h-1}$ is such that $A_i \mathbf{e}_1 = \mathbf{e}_i$, $i = 1, 2, \cdots, k$ and $A_{i+k}\mathbf{e}_1 = \mathbf{e}_{n+1}$, $i = 1, 2, \cdots, h-1$, then the matrices $\tilde{A}_i$ obtained by deleting the first $h-1$ rows and columns of A_i have null elements, except the elements of indices (r, s), such that $r - s = i - 1$ if $i = 1, 2, \cdots, k$ and $s - r = i - k$ if $i = k + 1$,

$k+2, \cdots, k+h-1$. Therefore $\tilde{A}_i$, $i=1, 2, \cdots, h+k-1$ is a basis of $\mathcal{B}_{n,k,h}$. The proof is complete in the view of Proposition 5.

It is worth pointing out that, as the above results show, we have that the border rank of k-Hessenberg Toeplitz matrices is $n+k-1$ over any field.

REFERENCES

[1] A. ALDER, *Border rank*, 5th Oberwolfach Conf. on Complexity Theory, 1981, Oberwolfach.

[2] M. D. ATKINSON AND N. M. STEPHENS, *On the maximal multiplicative complexity of a family of bilinear forms*, Linear Algebra Appl., 27 (1979), pp. 1-8.

[3] M. D. ATKINSON AND S. LLOYD, *The ranks of $m \times n \times (mn-2)$ tensors*, this Journal, 12 (1983), pp. 611-615.

[4] D. BINI, *Border rank of $p \times q \times 2$ tensor and the optimal approximation of a pair of bilinear forms*, Lecture Notes in Comput. Sci., 85 (1980), pp. 98-108.

[5] ———, *Tensor rank and border rank of certain classes of matrices and the fast evaluation of determinant, inverse matrix and eigenvalues*, Calcolo, 22 (1985), pp. 209-228.

[6] D. BINI AND M. CAPOVANI, *Fast parallel and sequential computations and spectral properties concerning band Toeplitz matrices*, Calcolo, 20 (1983), pp. 177-189.

[7] D. BINI, M. CAPOVANI, G. LOTTI AND F. ROMANI, *$O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication*, Inform. Process. Lett., 8 (1979), pp. 234-235.

[8] D. BINI AND M. CAPOVANI, *Spectral and computational properties of band symmetric Toeplitz matrices*, Linear Algebra Appl., 52/53 (1983), pp. 99-126.

[9] D. BINI, G. LOTTI AND F. ROMANI, *Approximate solutions for the bilinear form computational problem*, this Journal, 9 (1979), pp. 692-697.

[10] D. BINI, *On commutativity and approximation*, Theoret. Comput. Sci., 28 (1984), pp. 134-150.

[11] ———, *Parallel solution of certain Toeplitz linear systems*, this Journal, 13 (1984), pp. 268-276.

[12] ———, *Relations between exact and approximate bilinear algorithms, applications*, Calcolo, 17 (1980), pp. 87-97.

[13] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, this Journal, 11 (1982), pp. 472-492.

[14] P. DAVIS, *Circulant Matrices*, John Wiley, New York, 1979.

[15] H. F. DE GROOTE, *Characterization of division algebras of minimal ranks and the structure of their algorithm varieties*, this Journal, 12 (1983), pp. 101-117.

[16] ———, *On the complexity of quaternion multiplication*, Inform. Process Lett., 3 (1975), pp. 177-179.

[17] D. DOBKIN AND J. VAN LEEUWEN, *The complexity of vector products*, Inform. Process. Lett., 4 (1976), pp. 149-154.

[18] F. R. GANTMACHER, *The Theory of Matrices*, Chelsea, New York, 1959.

[19] F. C. HITCHCOCK, *Multiple invariants and generalized rank of a p-way matrix of tensor*, J. Math. Phys. MIT, 7 (1927).

[20] T. D. HOWELL, *Global properties of tensor rank*, Linear Algebra Appl., 22 (1978), pp. 9-23.

[21] T. D. HOWELL AND J. C. LAFON, *The complexity of quaternion product*, Technical Report 75-245, Computer Sci. Dept., Cornell University, Ithaca, NY, 1975.

[22] J. JA' JA', *Optimal evaluation of pairs of bilinear forms*, this Journal, 8 (1979), pp. 443-462.

[23] J. B. KRUSKAL, *Three-way arrays: rank and uniqueness of trilinear decompositions with application to arithmetric complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95-138.

[24] J. C. LAFON, *Base tensorielle des matrices de Hankel (ou de Toeplitz), applications*, Numer. Math., 23 (1975), pp. 349-361.

[25] T. LICKTEIG, *Typical Tensorial Rank*, Mathematisches Institut der Universität Tübingen, 1983.

[26] T. MUIR, *A Treatise on the Theory of Determinants*, Dover, New York, 1960.

[27] V. PAN, *Field extension and trilinear aggregating, uniting and cancelling for the acceleration of matrix multiplication*, Proc. 20th ACM Symposium on the Foundations of Computer Science, San Juan, Puerto Rico, 1979, pp. 28-38.

[28] A. SCHÖNHAGE, *Partial and total matrix multiplications*, this Journal, 10 (1981), pp. 433-455.

[29] V. STRASSEN, *Rank and optimal computation of generic tensors*, Linear Algebra Appl., 52/53 (1983), pp. 645-685.

[30] ———, *Vermeidung von Divisionen*, J. Reine Angew. Math., 264 (1973), pp. 184-202.

[31] P. ZELLINI, *On the optimal computation of a set of symmetric and persymmetric bilinear forms*, Linear Algebra Appl., 23 (1979), pp. 101-119.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987

© 1987 Society for Industrial and Applied Mathematics
005

GRAPH MINIMAL UNCOLORABILITY IS D^P-COMPLETE*

JIN-YI CAI† AND GABRIELE E. MEYER‡

Abstract. In their excellent paper, C. H. Papadimitriou and M. Yannakakis [J. Comput. System Sci., 28 (1982), pp. 244–259] asked whether the minimal-3-uncolorability problem is, among other Critical Problems, D^P-complete. This paper gives an affirmative answer to the above question. We show that minimal-k-uncolorability is D^P-complete, for all fixed $k \geqq 3$. Furthermore, for $k = 3$, the reduction can be modified by using "sensitive" gadgets to resolve the planar case.

Key words. NP-completeness, D^P-completeness, graph coloring, planar graph coloring

AMS(MOS) subject classifications. 05C15, 05C35, 68E10, 68C25

1. Introduction. Graph coloring has fascinated generations of mathematicians with its apparent simplicity and intrinsic subtlety. Planar graph coloring is especially a topic that has attracted the most prominent people in the field. With the emergence of computer science, the subject has further flourished, not only in its esthetic appeal but in its practical applications as well. The theory is rich, the literature abundant.

There are many questions one can ask on graph coloring; some have been answered, and others are still open. Naturally, one would like to classify different problems on graph coloring according to their inherent computational complexity.

In 1972, R. Karp [K] proved that the graph colorability is, in general, hard. Technically, he showed that the colorability problem is NP-complete. Later, L. Stockmeyer [S] extended his result to planar graphs. In 1982, C. H. Papadimitriou and M. Yannakakis [PY] delighted the research community by introducing a new complexity class D^P, which attempts to characterize a wide class of combinatorial optimization problems. The class D^P is defined to be all languages that can be expressed as the intersection of a language in NP with another in coNP (or, equivalently as the difference between two languages in NP, hence the name D^P). Many interesting problems are easily shown to be in D^P, in particular, a class of *critical problems.*

According to [PY], "some of the hardest problems in graph theory concern critical graphs." (A critical graph is one that does not have a property, but deleting *any* node creates one that has.) In particular, they ask whether the minimal-3-uncolorability problem is D^P-complete. They pointed out that these problems seem to be extremely delicate—"too delicate to sustain any of the known reduction methods."

The class D^P has since been the subject of much study. (See the references.) However, D^P-completeness remained unknown to any *critical problem* until recently C. H. Papadimitriou and D. Wolfe [PW] proved the minimal unsatisfiability problem to be D^P-complete. It is a very nice proof, based on a reduction from SAT-UNSAT, a known D^P-complete problem.

We shall base our reduction on a slight variant of the minimal unsatisfiability problem, namely,

* Received by the editors June 17, 1985; accepted for publication (in revised form) April 17, 1986.

† Computer Science Department, Cornell University, Ithaca, New York 14853. The research of this author was supported by National Science Foundation grant DCR-8301766.

‡ Mathematics Department, Cornell University, Ithaca, New York 14853.

Minimal-3-Unsatisfiability (Min-3-Unsat). Given a Boolean formula f with each clause having exactly 3 literals. Is it true that f is unsatisfiable, but deleting any clause creates a satisfiable formula?

The standard reduction from SAT to 3SAT can be used to show that the Min-3-Unsat is D^P-complete. (We refer to [GJ] for details.)

We fix some notation and terminology.

For a Boolean formula f and a clause c, denote $f-c$ the formula obtained from f by deleting c. For a graph $G=(V, E)$ and a node $v \in V$, denote $G-\{v\}$ the graph obtained by deleting v and all adjacent edges from G. Given two graphs $G_i=(V_i, E_i)$, $i=1, 2$, denote the union graph $G_1 \cup G_2=(V_1 \cup V_2, E_1 \cup E_2)$; and the product graph $G_1 \times G_2=(V_1 \cup V_2, E_1 \cup E_2 \cup \{(v_1, v_2) | v_i \in V_i\})$. A coloring of a graph G is an assignment of vertices such that no two adjacent vertices are assigned the same.

In § 2, we will prove the minimal-3-uncolorability problem to be D^P-complete. The reduction will be used later repeatedly. In § 3, our result is generalized to the minimal-k-uncolorability problem, for an arbitrary $k \geqq 3$. We also notice that the problem remains D^P-complete for graphs with bounded degree. In § 4, we study the important case—planar graphs. The proof is quite involved; therefore we postpone certain technical details to § 5.

2. Minimal-3-uncolorability. We begin by considering the following well-known problem:

Minimal-3-Uncolorability (M3UC). Given a graph $G=(V, E)$, is it true that G is uncolorable with 3 colors, but deleting any node v from G, the resulting graph is 3-colorable?

It is obvious that the M3UC problem is in D^P, since there are only $|V|$ nodes and 3-colorability is in NP.

We will reduce the Minimal-3-Unsatisfiability problem, a known D^P-complete problem, to M3UC, thus proving

THEOREM 2.1. M3UC *is* D^P*-complete.*

The rest of § 2 is devoted to the proof of Theorem 2.1. Consider a Boolean formula $f=\bigwedge_{1 \leqq j \leqq m} c_j$ of variables $x_1, \cdots, x_n$, where c_j are clauses. Since f is a candidate for Min-3-Unsat, we assume that each c_j consists of 3 literals.

Let us construct a graph G_f as follows:

For the variables x_i, $1 \leqq i \leqq n$, there is a "truth-setting" component $T=(V', E')$, where $V'=\{x_i, \sim x_i | 1 \leqq i \leqq n\} \cup \{v_c\}$, and $E'=\{(x_i, \sim x_i), (x_i, v_c), (\sim x_i, v_c) | 1 \leqq i \leqq n\}$, i.e., $2n$ "literal nodes," each connected to its conjugates; and a "center node," connected to all of them.

There is an extra "truth-selecting node" v_s that is connected to v_c.

The graph constructed so far is seen in Fig. 2.1. Let us call it T^*. Note: Any 3-coloring of T^* corresponds naturally to a truth assignment of the variables, and vice versa—with v_s having the same color as the true literals.

For each clause $c_j=(p_{j1}, p_{j2}, p_{j3})$, there are 3 "fanout" components $O_j=(V_j, E_j)$, where $V_j=\{v_s, p_{jk}, a_{jk}, b_{jk} | 1 \leqq k \leqq 3\}$, and $E_j=\{(p_{jk}, a_{jk}), (a_{jk}, b_{jk}), (a_{jk}, v_s), (b_{jk}, v_s) | 1 \leqq k \leqq 3\}$. Please note here v_s and p_{jk}'s are previously constructed nodes, but now we add some new edges connecting those nodes to others.

For example, $c_1=(x_1, \sim x_2, x_3)$. The "fanout" components are depicted in Fig. 2.2.

Remark. Consider a 3-coloring on a single "fanout" component, using colors {T, F, C}. Suppose v_s is colored T. We have:

1) If the literal node is colored T, then there are two possible colorings as in Fig. 2.3.

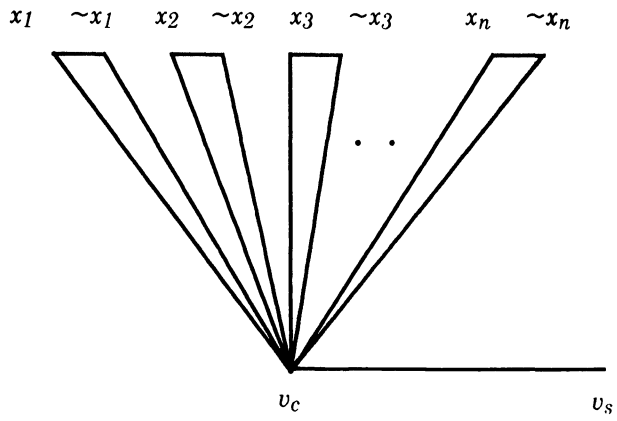


FIG. 2.1. *T^* truth-setting.*

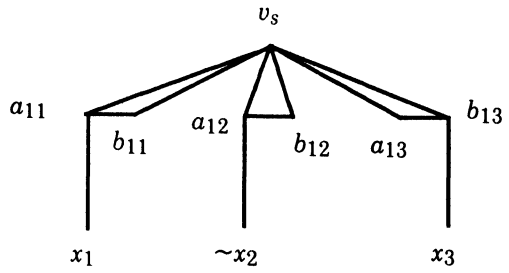


FIG. 2.2. *Fanout for c_1.*

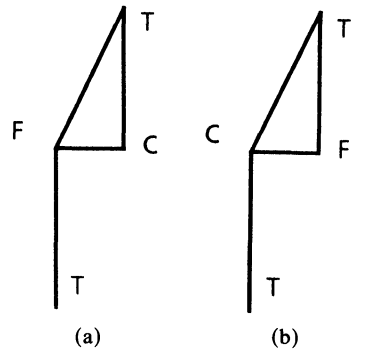


FIG. 2.3

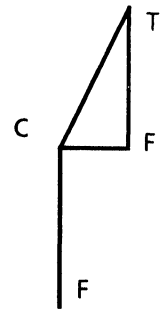


FIG. 2.4

2) If the literal node is colored F, then there is but one possible coloring as in Fig. 2.4.

We call the literals p_{jk} "input," and b_{jk} "output." Intuitively, for a fanout, if the "input" is T, the "output" can be either F or C; if the "input" is F, the "output" must be F, assuming v_s is colored T.

The final set of components are for "truth-checking" purposes, one for each clause c_j. $C_j = (V_j', E_j')$, where $V_j' = \{b_{jk}, t_{jk} | 1 \leqq k \leqq 3\}$, and $E_j' = \{(t_{j1}, t_{j2}), (t_{j2}, t_{j3}), (t_{j3}, t_{j1}), (b_{jk}, t_{jk}) | 1 \leqq k \leqq 3\}$. In other words, we will add a "triangle" $\{t_{j1}, t_{j2}, t_{j3}\}$, with each node joining appropriate b_{jk}'s, the output nodes constructed earlier (Fig. 2.5).

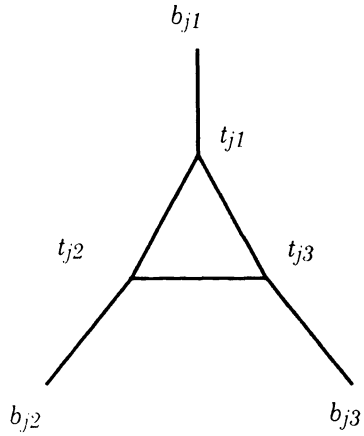


FIG. 2.5. *Truth-checking component.*

We note that it is impossible to color C_j with 3 colors such that $\{b_{j1}, b_{j2}, b_{j3}\}$ all receive the same color. (Otherwise, $\{t_{j1}, t_{j2}, t_{j3}\}$ forms a clique using two remaining colors.) On the other hand, it *is* possible to color C_j in 3 colors with $\{b_{j1}, b_{j2}, b_{j3}\}$ arbitrarily colored with exactly 2 colors. To see this, assume exactly {F, C} are used. By symmetry of C_j, we may assume b_{j1} is colored C, and b_{j2}, b_{j3} colored F. Then the following is a valid 3-coloring using {T, F, C} (Fig. 2.6).

The construction is completed by taking the union of all subgraphs: $G_f = (V, E) = T^* \cup_{1 \leqq j \leqq m} (O_j \cup C_j)$. It is certainly a polynomial time transformation. This is presented in Fig. 2.7.

We first establish

LEMMA 2.2. *f is satisfiable iff G_f is 3-colorable.*

Proof. Suppose f is satisfiable with ν a satisfying assignment. We will use colors {T, F, C}. Color each literal node p with $\nu(p)$; node v_c with color C and node v_s with T. This is a valid 3-coloring for T^*.

For each clause c_j, we color its fanout components as follows: c_j is satisfied by ν, at least one of the literals $\{p_{j1}, p_{j2}, p_{j3}\}$ is colored T. Choose an arbitrary one and color its fanout as in Fig. 2.3(a). For the other two fanout components of c_j, either

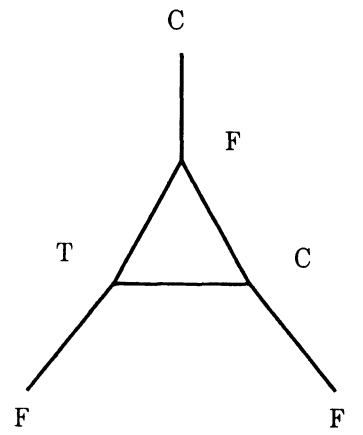


FIG. 2.6

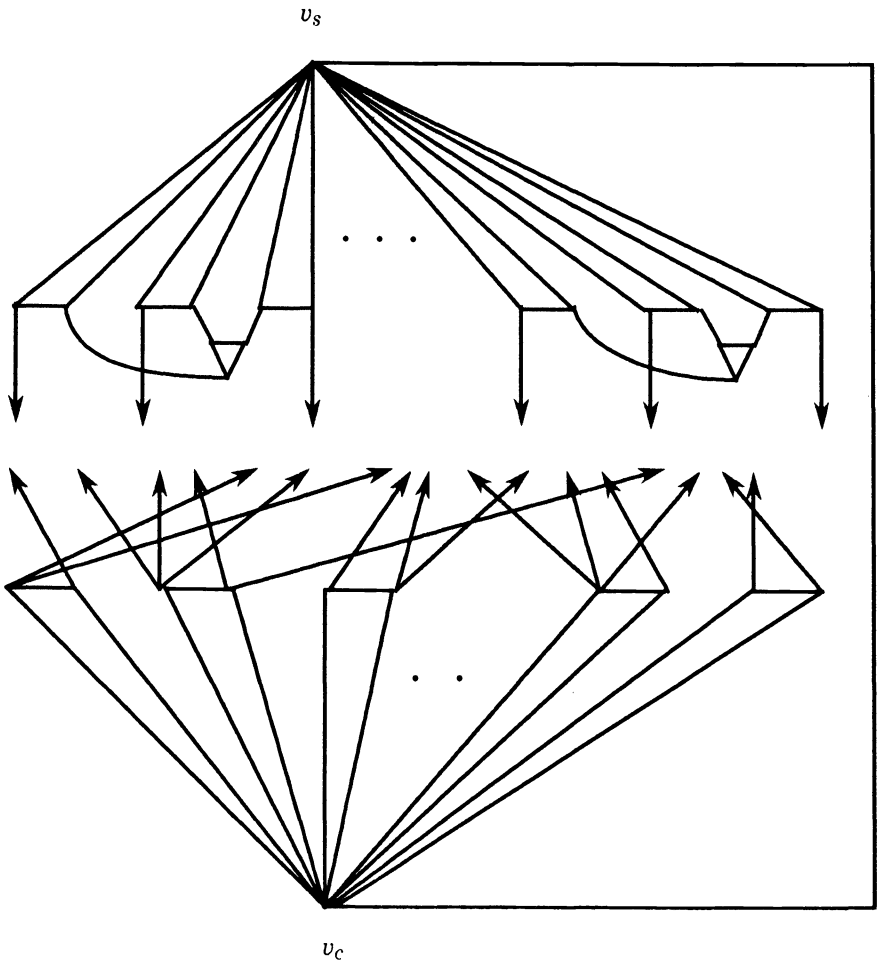


FIG. 2.7. *Graph* G_f.

the literal is F, in which case we color it as in Fig. 2.4; or the literal is T, in which case we color it as in Fig. 2.3(b). Now the 3 output nodes $\{b_{j1}, b_{j2}, b_{j3}\}$ are colored with exactly 2 colors. Then we can color the remaining 3 nodes $\{t_{j1}, t_{j2}, t_{j3}\}$ as in Fig. 2.6.

Hence, G_f is 3-colorable.

For the other direction, suppose G_f is 3-colorable. Consider such a 3-coloring. Call T the color on v_s, C the color on v_c. Then we assign a literal to be true *iff* it is colored T. (The false literals are colored by the third color, say F.) If for some clause c_j all its literals were assigned false, then in the fanout component O_j all three input and hence all three output would be colored F. (See Figs. 2.2 and 2.4.) But this is impossible. Hence f is satisfiable. QED

It remains to show that

f is minimally unsatisfiable $\Leftrightarrow$ G_f is minimally uncolorable with 3 colors.

Suppose f is minimally unsatisfiable. We already know that G_f is uncolorable with 3 colors. We need to show that deleting any vertex from G_f will result in a graph that is 3-colorable. We may show this by using one extra color on any vertex of G_f and coloring the rest with 3 colors {T, F, C}.

So suppose node x_1 is given a new color N. Since f is minimally unsatisfiable, we claim that x_1 must appear in at least one clause. Otherwise, $\sim x_1$ must appear in some clause c_j. Now $f - c_j$ has a satisfying assignment ν. But since x_1 appears nowhere in f, the following modified truth assignment μ also satisfies $f - c_j$:

$$\mu(x_i) = \nu(x_i) \quad \text{if } i > 1, \quad \text{and F if } i = 1.$$

But since $\sim x_1$ appears in c_j, μ also satisfies f, a contradiction. Hence x_1 appears in some clause c_k.

Let ν be a satisfying assignment for $f - c_k$. Since ν does not satisfy f itself, ν does not satisfy c_k. Therefore $\nu(x_1) = \text{F}$, or, $\nu(\sim x_1) = \text{T}$. Now color the literal nodes according to ν (except x_1), i.e., x_1 has N, $\sim x_1$ has T, x_i (or $\sim x_i$) has $\nu(x_i)$ (or $\nu(\sim x_i)$), for $i > 1$. Color v_c with C, and v_s with T.

If a clause does not contain x_1, it is a part of $f - c_k$; hence its fanout and truth checking components can be colored exactly as before. If a clause contains x_1 (this includes the case c_k), regardless of whether the other 2 literals are assigned T or F (in case of c_k, they must be F), color those 2 fanout components so that the 2 output nodes are colored F (Figs. 2.3(b) and 2.4), and the output of x_1 colored C (Fig. 2.8). Then we can color the truth checking component as in Fig. 2.6.

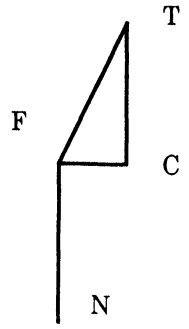


FIG. 2.8

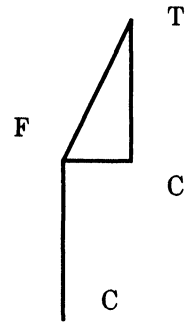


FIG. 2.9

We have shown that $G_f - \{x_1\}$ can be 3-colored. By symmetry, all other cases deleting a literal node are proven.

Now suppose v_c is given N. Still assume x_1 appears in clause c_k, and let ν be a satisfying assignment for $f - c_k$. Now color x_i (or $\sim x_i$) with $\nu(x_i)$ (or $\nu(\sim x_i)$), for $i > 1$, and color x_1 with C, $\sim x_1$ with T ($= \nu(\sim x_1)$). Color v_s with T. Again we need only to worry about clauses that contain x_1 (including c_k).

To such a clause, regardless of whether the other 2 literals were colored T or F, we can color those 2 fanout components so that the output nodes are both F. And the output for x_1 is C (Fig. 2.9). The rest of this case is the same.

The next case is when v_s gets N. This is easy. Just color x_i with T, $\sim x_i$ with F, v_c with C. For each fanout, since v_s now has N, we can arrange the output to be any one of {T, F, C}, regardless of the input. We will arrange the output to be C, F, F, respectively. The rest takes care of itself.

Next we consider an arbitrary clause c_l. Deleting a_{l1}, just as before, we can get a satisfying assignment for $f - c_l$. And we can color the 3 output nodes for O_l (fanout for c_l) with C, F, F, respectively (Fig. 2.10).

Deleting b_{l1}. We "prove by picture" Fig. 2.11.

Deleting t_{l1}. See Fig. 2.12.

By symmetry, done for $a_{l2}, a_{l3}, b_{l2}, b_{l3}, t_{l2}, t_{l3}$.

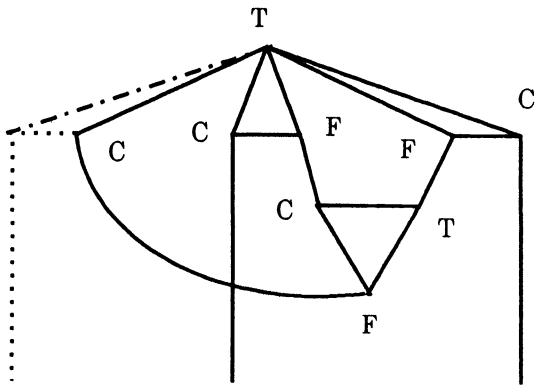


FIG. 2.10

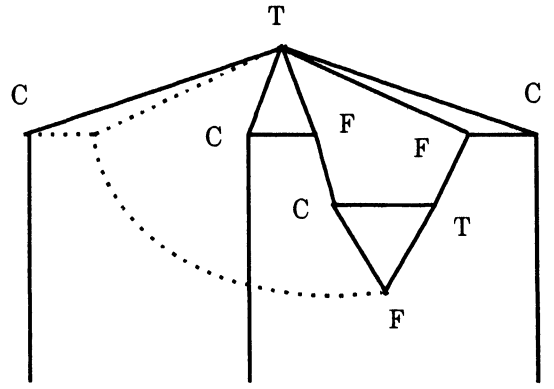


FIG. 2.11

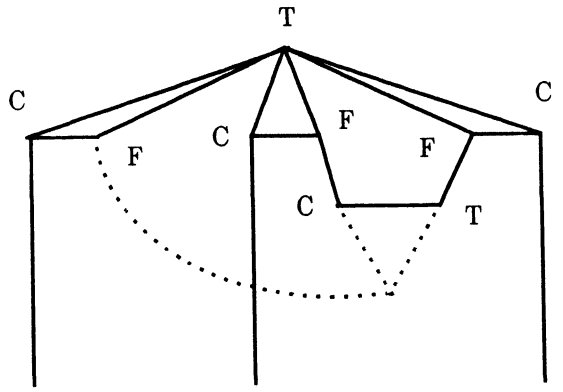


FIG. 2.12

We have proven that G_f is minimally uncolorable with 3 colors.

To complete the proof, we note that: if G_f is minimal-3-uncolorable, then, first, f is unsatisfiable; furthermore, suppose we delete clause c_j, then $f - c_j$ is satisfiable. To see this, we delete t_{j1} from G_f. By assumption $G_f - \{t_{j1}\}$ is 3-colorable. Exactly as in Lemma 2.2, one can extract naturally a satisfying assignment for $f - c_j$ from a 3-coloring.

This completes the proof of Theorem 2.1.

3. Extensions and restrictions. Some simple extensions are in order. First we consider k-colorability in general. We will show that for an arbitrary $k \geqq 3$, the following problem is also D^P-complete.

Minimal-k-Uncolorability (MkUC). Given a graph G, is it true that G is not colorable by k colors, but by deleting any node from G, the resulting graph is k-colorable?

THEOREM 3.1. MkUC *is* D^P*-complete.*

Proof. Again, membership in D^P is obvious.

For $k > 3$, we will modify our construction in the last section as follows: Given f, a Boolean formula, we construct $K_{k-3} \times G_f$: Add to G_f a clique of size $k-3$, and connect every node of the clique to every node of G_f.

We will verify that G_f is minimal-3-uncolorable *iff* $K_{k-3} \times G_f$ is minimal-k-uncolorable.

First, it is obvious that for any graph G,

(*) G is 3-colorable *iff* $K_{k-3} \times G$ is k-colorable.

Suppose G_f is M3UC. Let us delete any node v from $K_{k-3} \times G_f$. If $v \in K_{k-3}$, then the graph is $K_{k-4} \times G_f$ (if $k > 4$, or G_f if $k = 4$). G_f is 4-colorable, done.

If $v \in G_f$ is deleted, then we can color $G_f - \{v\}$ with 3 colors and color K_{k-3} with the other $k-3$ colors.

Now the other way, suppose $K_{k-3} \times G_f$ is MkUC. Delete a node v from G_f. $K_{k-3} \times (G_f - \{v\}) = (K_{k-3} \times G_f) - \{v\}$ is k-colorable, by assumption, therefore $G_f - \{v\}$ is 3-colorable, by (*). QED

In 1941, R. L. Brooks [B] proved the following: a connected graph with maximum degree equal to 3 is 3-colorable *iff* it is *not* K_4, the complete graph of size 4. Thus, 3-colorability for graphs with maximum degree $\leqq 3$ is solvable in P. Therefore, so is minimal-3-uncolorability.

On the other hand, M. Garey, D. Johnson and L. Stockmeyer [GJS] proved in 1976 that graph 3-colorability remains NP-complete for graphs with maximum degree $\leqq 4$.

We have the following theorem:

THEOREM 3.2. M3UC *for graphs with maximum degree* $\leqq 5$, *remains* D^P*-complete.*

4. Planar graph minimal-3-uncolorability. We consider in this section an important class of graphs, the planar graphs. Planar graphs are interesting both theoretically and practically. The famous 4-color problem concerns the 4-colorability of arbitrary planar graphs.

Our main result in this section is

THEOREM 4.1. *Minimal-3-uncolorability restricted to planar graphs is also* D^P*-complete.*

Our strategy is to transform our graph G_f to a planar graph G_f^* such that, G_f is M3UC *iff* G_f^* is M3UC.

The following "gadget" H is designed by M. Fischer [GJS] (Fig. 4.1). It has 13 vertices, and 4 "outlets" labeled α, α', β and β'. Here is the interesting property of H: There are essentially (up to isomorphism) 2 ways of coloring H with 3 colors. More specifically,

LEMMA 4.2 [GJS]. *Let* σ *be any coloring of* H *with* $\leqq 3$ *colors. Then* σ *uses* 3 *colors and must satisfy*:

(1) $\sigma(\alpha) = \sigma(\alpha')$,

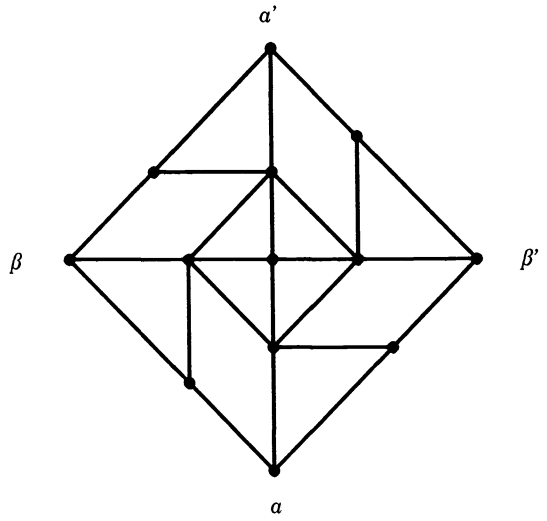


FIG. 4.1. *Graph H.*

(2) $\sigma(\beta) = \sigma(\beta')$, *and*

(3) $\sigma(\alpha) = \sigma(\beta)$ *and* $\sigma(\alpha) \neq \sigma(\beta)$ *are both possible.*

The two colorings are shown below (Figs. 4.2 and 4.3).

This gadget is used in [GJS] to effect a "crossover": To allow 2 crossing edges to be replaced by a planar subgraph H, such that the resulting graph is "equivalent" to the original one with respect to 3-colorability.

For our purpose, we need something that is more "sensitive." Recall that in the proof of Theorem 2.1, we argued that if f is minimally unsatisfiable, then deleting any node from G_f could make it 3-colorable. The idea behind this is that, deleting any node helps one to color the components of (at least) one clause.

We view the connection from a literal to its fanout component as an "input line," and the connection from an "output node" in a fanout to its truth-checking component as an "output line." Suppose two such "lines" are crossed over, and the crossing is in turn replaced by some "gadget."

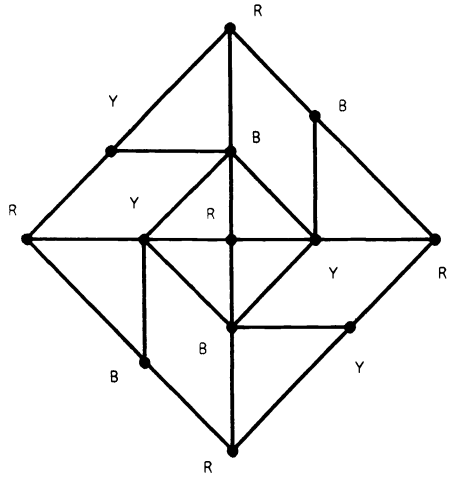


FIG. 4.2

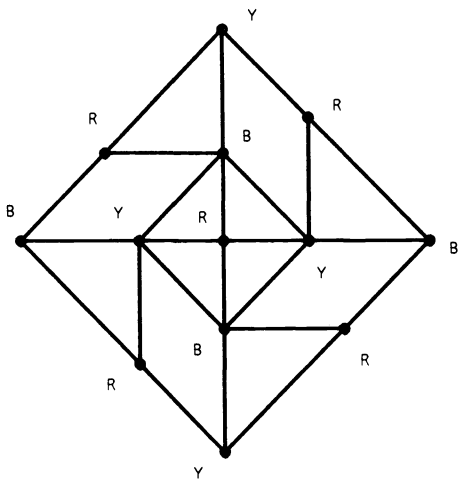


FIG. 4.3

Now let us delete a node from this gadget. We ask: Is it always possible to alter the color "at the other end" of the gadget, at least for one of the two directions? Unfortunately, H does not work.

The crossover we will use is the planar graph K shown in Fig. 4.4.

Consider any 3-coloring of K. $\{\alpha, \alpha_1, \alpha_2\}$ must be colored with 3 distinct colors. By Lemma 4.2, α_1, α_1' must be colored the same, and so are β, γ. Similarly α_2, α_2' must be the same color, and so are γ, β'. Hence, α', being different than both α_1', α_2', must be given the same color as α. Meanwhile, β and β' must also have the same color.

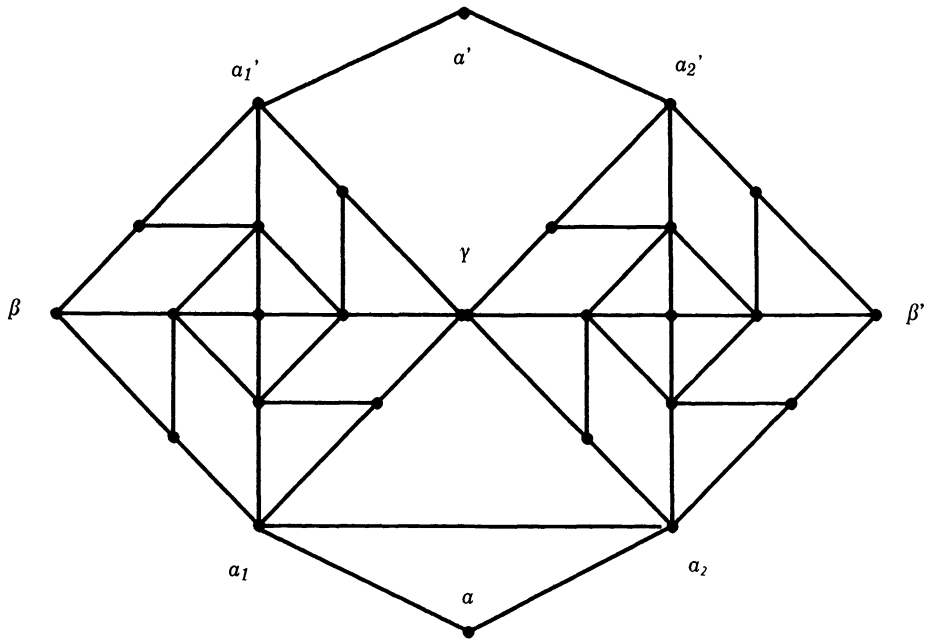


FIG. 4.4. *Graph K.*

Furthermore, by Lemma 4.2, we know that it is both possible to have the color of α, α' to be the same as β, β', or different from β, β'. Thus,

LEMMA 4.3. *The statement made in Lemma* 4.2, *about H, is valid for K.*

Let us examine our graph G_f more closely. If we draw it as in Fig. 2.7, we find that the only crossing present are those between 2 input lines and, between one output line and one input line, one per each clause. For each such crossing we wish to substitute it by our graph K.

We construct G_f^* from G_f as follows:

(a) Embed G_f on the plane as in Fig. 2.7, such that no edge touches a vertex of G_f other than its endpoints, the only crossings are as described before, and all crossings are simple in the sense that no more than 2 edges cross at a point.

(b) To each crossed edge in G_f, add new vertices, one between each endpoint and the nearest crossing to it, and one between each pair of adjacent crossings (Fig. 4.5(a), (b)).

(c) Replace each crossing in the graph by a copy of K, identifying $\alpha, \alpha', \beta, \beta'$ with 4 nearest new vertices. For each such crossing at least one edge is an input line.

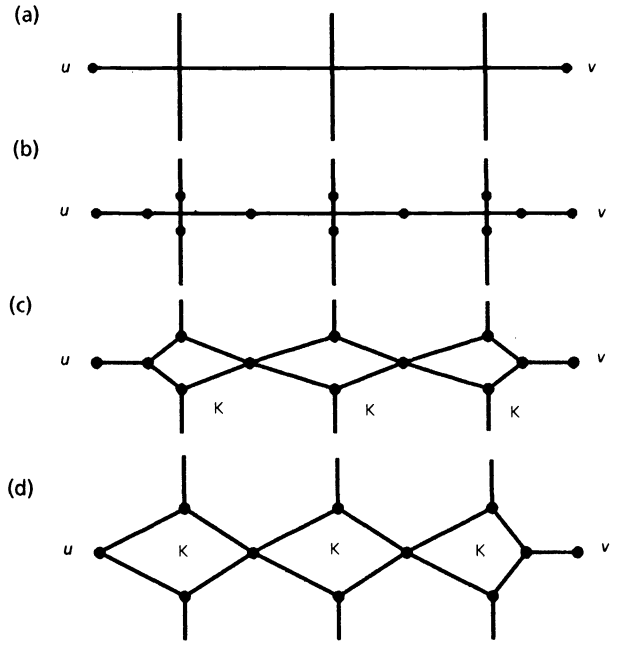


FIG. 4.5

We will always identify α with a point that is on an input line, and (on that line) nearer to the literal. (If both are input lines, just pick any one) (Fig. 4.5(c)).)

(d) For each crossed edge in G_f, if it is an input line choose the literal, if it is an output line choose the output node. Then coalesce the chosen node with the nearest *new* vertex on the line. The original vertex (from V) retains its identity (Fig. 4.5(d)).

With the help of Lemma 4.3, one can easily show that

LEMMA 4.4. *G_f is 3-colorable iff G_f^* is 3-colorable.*

We turn to the "sensitivity" question: Given a new color on an arbitrary node of K, what 3-colorings can there be for the rest of K?

Suppose the 3 colors are {F, T, C}, and the new color is N. We have the following proposition.

PROPOSITION 4.5. *For every $v \in K$, given new color* N, *then*
Either

(a) "*The vertical type:*" *There exist 3-colorings σ_1 and σ_2 for $K-\{v\}$ such that*
a1. $\sigma_1(\beta)=\sigma_1(\beta')=\mathrm{F}$ *and* $\sigma_1(\alpha')\neq\mathrm{F}$ *and* $\sigma_1(\alpha)=\mathrm{F}$ *if* $\alpha\neq v$,
a2. $\sigma_2(\beta)=\sigma_2(\beta')=\mathrm{T}$ *and* $\sigma_2(\alpha')\neq\mathrm{F}$ *and* $\sigma_2(\alpha)=\mathrm{F}$ *if* $\alpha\neq v$,

or

(b)[1] "*The horizontal type:*" *There exist 3-colorings σ_1 and σ_2 for $K-\{v\}$ such that* a1 *and* a2 *hold with α, α' interchanged for β, β', respectively, i.e.*
b1. $\sigma_1(\alpha)=\sigma_1(\alpha')=\mathrm{F}$ *and* $\sigma_1(\beta')\neq\mathrm{F}$ *and* $\sigma_1(\beta)=\mathrm{F}$ *if* $\beta\neq v$,
b2. $\sigma_2(\alpha)=\sigma_2(\alpha')=\mathrm{T}$ *and* $\sigma_2(\beta')\neq\mathrm{F}$ *and* $\sigma_2(\beta)=\mathrm{F}$ *if* $\beta\neq v$.

Intuitively the proposition states that, deleting arbitrarily a node from K, then there is a "direction," vertical or horizontal, such that we may color the rest of K in 3 colors {F, T, C} in such a way that, on that direction, the "in node" is F (unless it

[1] Note: By symmetry of K, if we interchange β with β' in (b), the statement is also valid.

is the deleted node) and the "out node" is not F, *regardless* of the color that is propagated on the other direction.

We will call α the "base node" in case (a), and β in case (b).

The proof of this proposition is deferred to the next section. Meanwhile we will assume its validity and prove:

THEOREM 4.6. *G_f is minimal-3-uncolorable iff G_f^* is minimal-3-uncolorable. Hence* M3UC *for planar graphs is also* D^P*-complete.*

Proof. ($\Rightarrow$) We know that G_f^* is 3-uncolorable.

To see that it is minimally so, let us delete a vertex v_0 from G_f^*, or equivalently give v_0 a new color N. If v_0 is not a vertex of any K, then it must be a node in G_f, (and not coalesced). By assumption, there is a 3-coloring for $G_f - \{v_0\}$. For each edge of G_f that is crossed, start at the coalesced node, propagate the colors across each K, and we obtain a 3-coloring for $G_f^* - \{v_0\}$.

So suppose v_0 is in some crossover K_0.

Case 1. K_0 is a crossover of two input lines, say from literals x and y. According to the proposition, there is a direction—"vertical" or "horizontal." WOLOG, assume this direction is on the x input line. By our construction and the footnote above, we may assume the base node is the crossover of x from 0 or more copies of K. Call it s (Fig. 4.6(a)). The input line of x leads to a particular clause c in which x appears as a literal. If we delete c from f, then there is a satisfying assignment μ for $f - c$. Since μ does not satisfy f, $\mu(x)$ must be F.

Color v_c with C, and v_s with T. Color the literal nodes according to μ as before. Except on the input line of x, we can propagate the colors on each crossover as usual; on the input line of x, we only propagate from x to s. (In case $s = x$, this step is void; if $s = v_0$, we propagate up to s.) (On the y input line, we still may propagate $\mu(y)$ to u to v, etc.) Now regardless of what $\mu(y)$ is, (T or F), according to the proposition, we may color the rest of K_0 with the 3 colors, such that the color of t is not F (Fig. 4.6(b)).

If there are a few more crossover K's on the way from t to the fanout component of x in c, we will just propagate the color of t (which is C or T; unless $t = v_0$, in which case we propagate T instead.)

Now the 3 output of the fanout components of c can be colored C, F, F and the rest are the same as before (Fig. 4.7).

Case 2. K_0 is a crossover of one output line and one input line in a single clause c. Suppose the base node is on the input line, then it is similar to Case 1: a satisfying assignment of $f - c$ must leave all 3 literals of c false, and then we may "alter" on the input line, such that the 3 output nodes of the fanout are C, F and F, respectively (Fig. 4.8).

Suppose the base node is on the output line, then it must be the output node of the fanout, as we constructed. Again, use a satisfying assignment for $f - c$ and alter on the output line and we may color the graph as shown (Fig. 4.9).

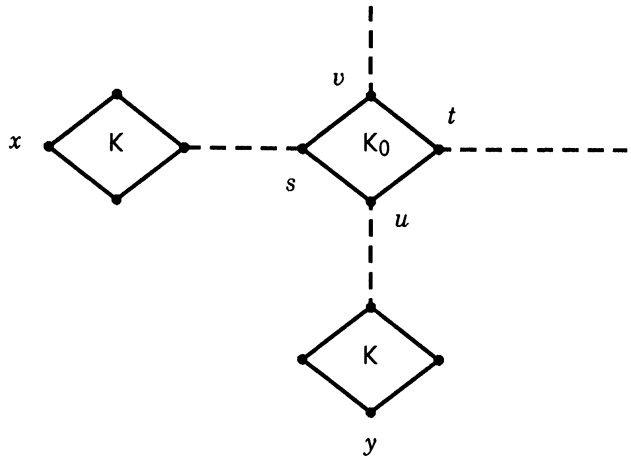
This completes the proof that G_f^* is indeed minimal-3-uncolorable.

($\Leftarrow$) We know that G_f is 3-uncolorable. We only need to show that $f - c$ is satisfiable for any clause c.

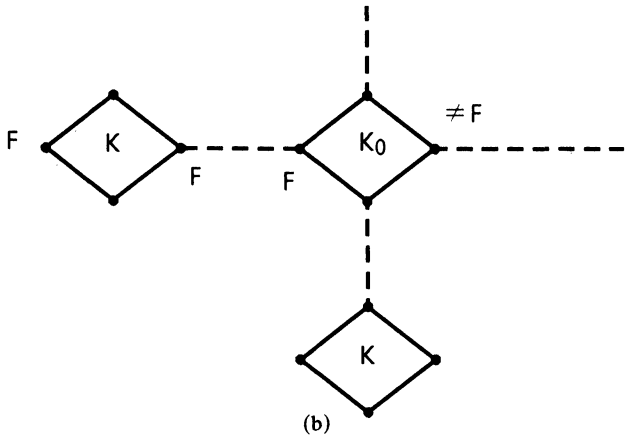
We go to the truth-checking component for c in G_f^*. Delete a node in the triangle, i.e. one of the t's. By assumption $G_f^* - \{v_0\}$ is 3-colorable. Now just as before, since the crossovers must "carry out their duties" faithfully, any 3-coloring gives a truth assignment that satisfies $f - c$. QED

It is also possible to restrict our attention to bounded degree planar graphs as in § 3.

THEOREM 4.7. M3UC *for planar graphs with maximum degree* $\leqq 5$, *is* D^P*-complete.*



(a)



(b)

FIG. 4.6

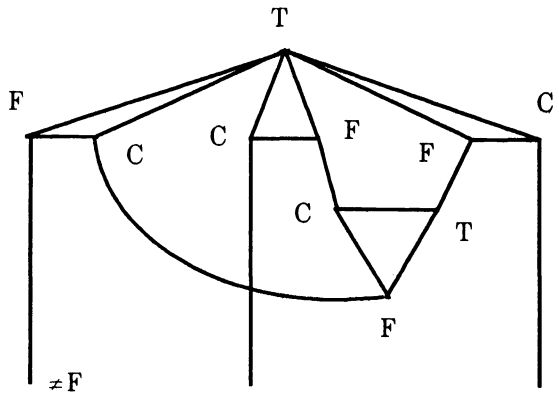


FIG. 4.7

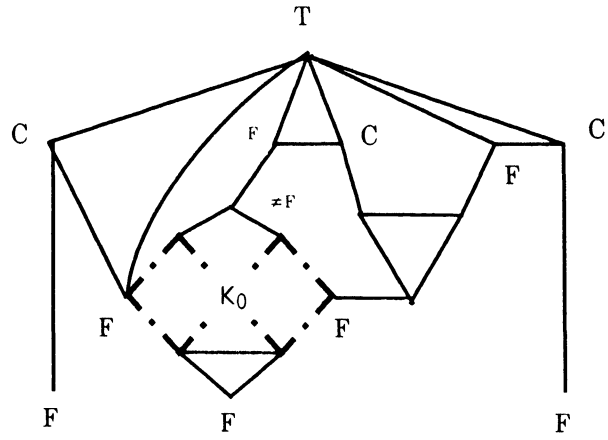


FIG. 4.8

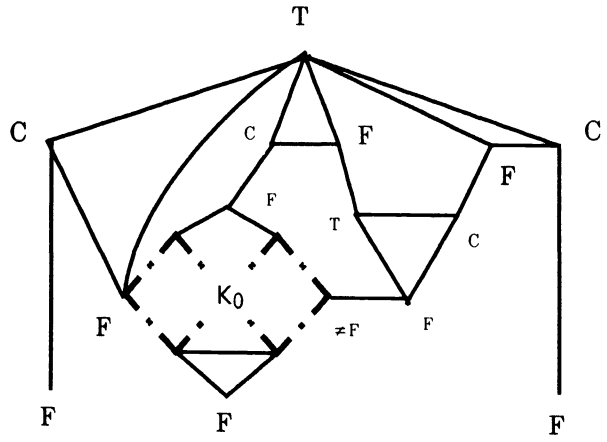


FIG. 4.9

5. Proof of the proposition. In this section, we give the proof of Proposition 4.5. The proof is organized into the following lemmas. We refer the reader to Fig. 4.4.

LEMMA 5.1. *The proposition is true if* $v \in \{\alpha, \alpha', \alpha_1, \alpha_1', \alpha_2, \alpha_2', \beta, \beta', \gamma\}$.

Proof. For example, if α_1' is given the new color, see Fig. 5.1.

Another example: if γ is given the new color, then depending on α, α' are F or T, we have two colorings in Figs. 5.2 and 5.3.

The rest are similar. QED

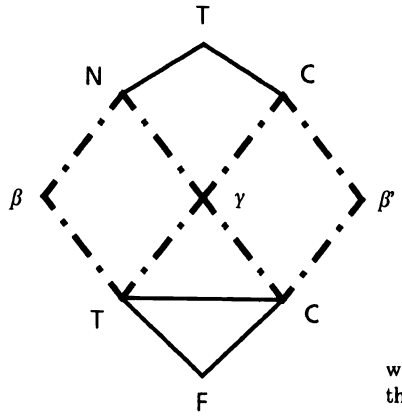
We are left with the cases where the deleted vertex belongs to one of the copies of H and is not an "outlet" of H. (See Fig. 4.1. Rename the outlets as x, y, x', y'.) We say that $H - \{v\}$ satisfies:

Condition 1. If the following two 3-colorings σ and τ are possible (Fig. 5.4).

$$\sigma(x) = \sigma(y) = \sigma(x') = \text{Blue and } \sigma(y') = \text{Red},$$

$$\tau(x) = \tau(y') = \tau(x') = \text{Blue and } \tau(y) = \text{Red}.$$

Similarly, it is said to satisfy Condition 1', if there are colorings in Fig. 5.5.



where β, γ, β' are the same: T or F.

FIG. 5.1

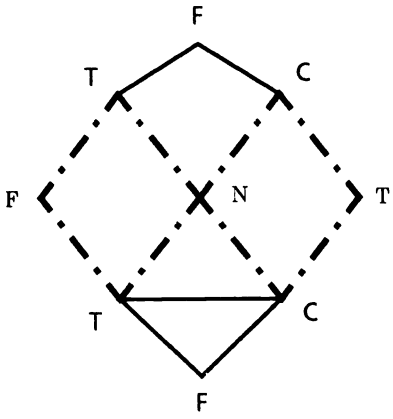


FIG. 5.2

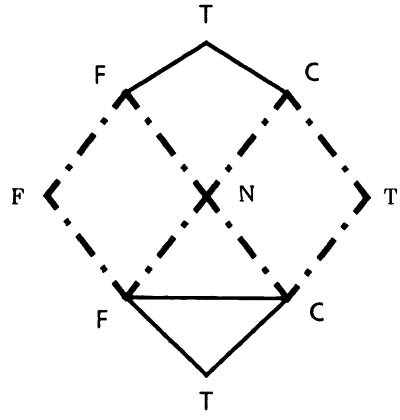


FIG. 5.3

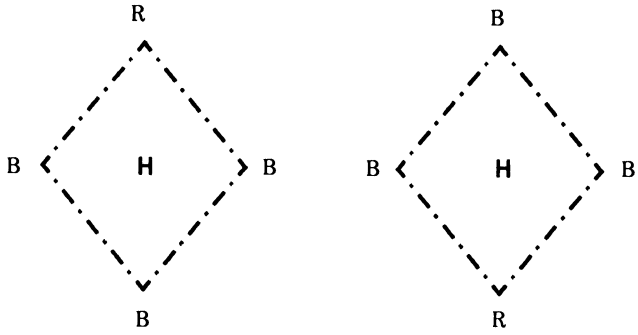


FIG. 5.4. *Condition* 1.

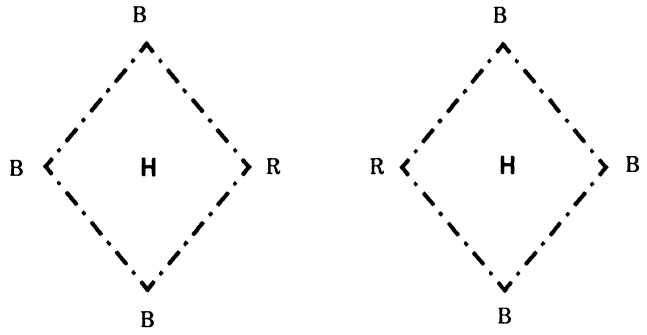


FIG. 5.5. *Condition* 1′.

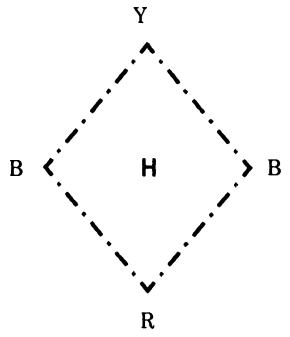


FIG. 5.6. *Condition* 2.

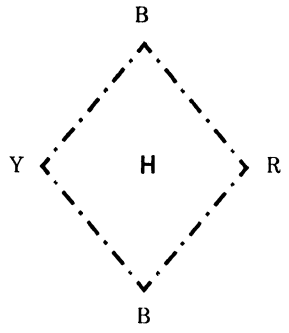


FIG. 5.7. *Condition* 2′.

We say that $H-\{v\}$ satisfies Condition 2, if Fig. 5.6 is possible. Similarly Condition 2′ is Fig. 5.7.

LEMMA 5.2. *For the left or the right copy of H in K, after deleting v, if* $H-\{v\}$ *in K satisfies any one of the above conditions then the proposition is true.*

Proof. We will only verify the case for the left copy of *H*. The other half is the same.

Under Condition 1, we will color as in Fig. 5.8: It is a "vertical" type.

It is similar for Condition 1′, this time a "horizontal" type (Fig. 5.9).

Condition 2. A vertical type (Fig. 5.10).

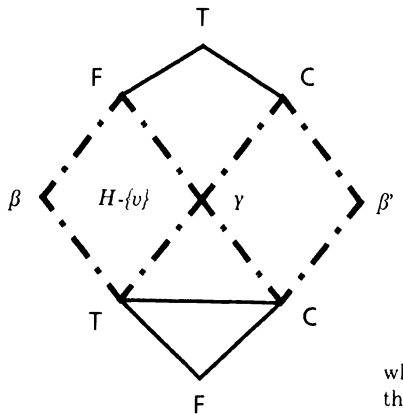


FIG. 5.8

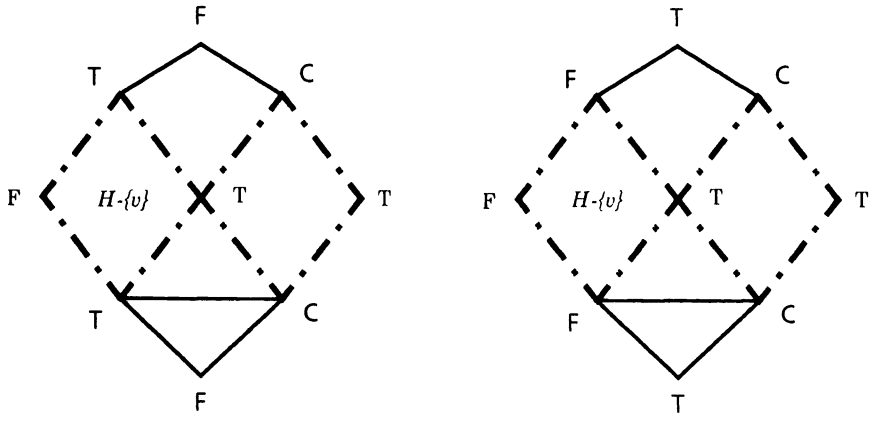


FIG. 5.9

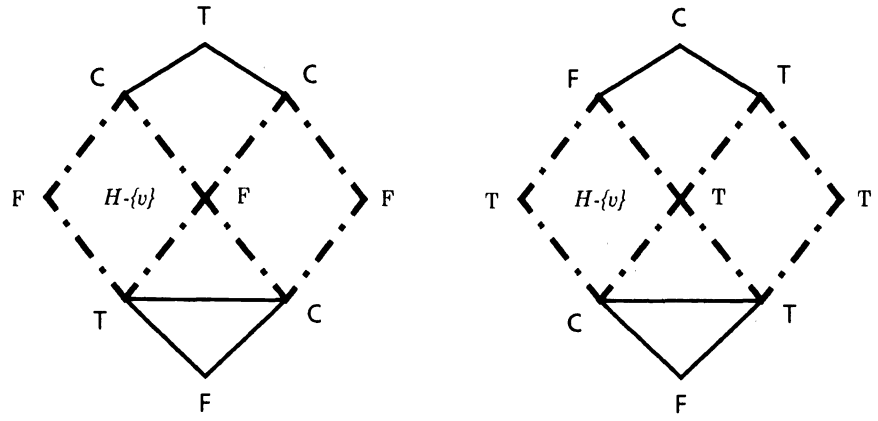


FIG. 5.10

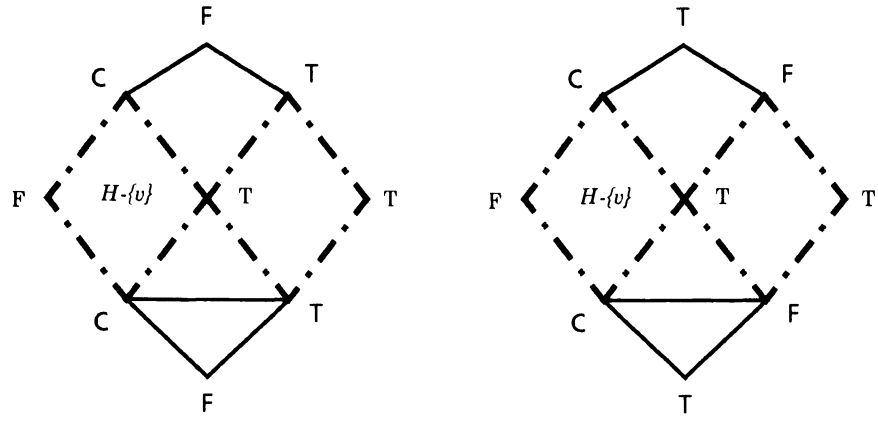


FIG. 5.11

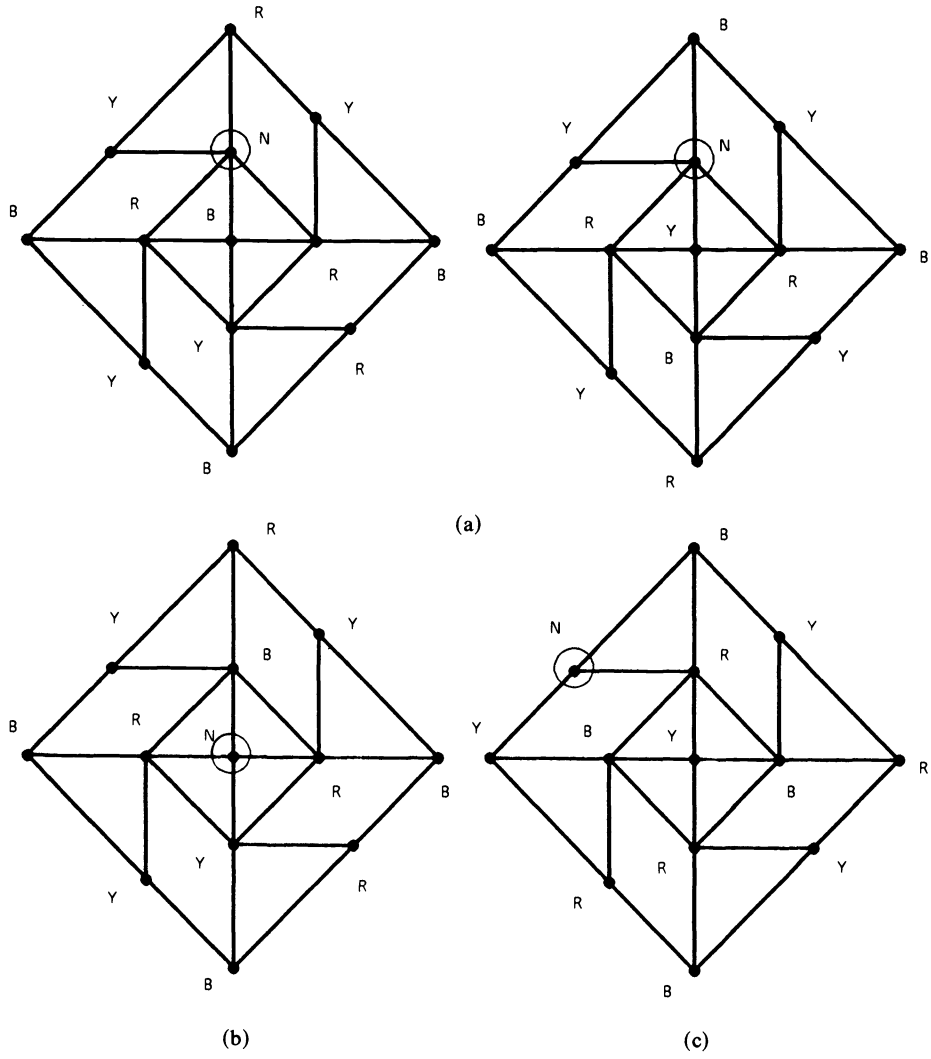


FIG. 5.12

Condition 2'. A horizontal type (Fig. 5.11). QED

In the light of Lemma 5.2, all we are left is to show

LEMMA 5.3. *Deleting any nonoutlet node v from H, $H-\{v\}$ satisfies one of the conditions.*

Proof. By the symmetry of H and the symmetry of the set of conditions (invariance under rotation), we only need to verify for 3 nodes, i.e., Fig. 5.12(a), (b) and (c) are valid 3-colorings. QED

Acknowledgments. We express our deepest gratitude to Professor J. Hartmanis, whose encouragement made this paper possible. We also wish to thank Professor V. Vazarani for helpful discussions.

We are grateful to the referees whose criticisms and suggestions have improved the presentation of the paper.

REFERENCES

[AH] K. APPEL AND W. HAKEN, *Every planar graph is 4-colorable*, Illinois J. Math., 21 (1977), pp. 429–567.

[B] R. BROOKS, *On coloring the nodes of a network*, Proc. Cambridge Philos. Soc., 37 (1941), pp. 194–197.

[BG] A. BLASS AND Y. GUREVICH, *On the unique satisfiability problem*, Inform. and Control, 55 (1982), pp. 80–88.

[CC] T. COLEMAN AND J. CAI, *The cyclic coloring problems and estimation of sparse Hessian matrices*, SIAM J. Algebraic Discrete Methods, 7 (1986), pp. 221–235.

[GJ] M. GAREY AND D. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[GJS] M. GAREY, D. JOHNSON AND L. STOCKMEYER, *Some simplified* NP*-complete graph problems*, Theor. Comput. Sci., 1 (1976), pp. 237–267.

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

[O] O. ORE, *Four Color Conjecture*, Academic Press, New York, 1976.

[PY] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, J. Comput. System Sci., 28 (1982), pp. 244–259.

[S] L. STOCKMEYER, *Planar 3-colorability is* NP*-complete*, SIGACT News, 5:3 (1973), pp. 19–25.

[PW] C. H. PAPADIMITRIOU AND D. WOLFE, *The complexity of facets resolved*, Proc. of the 26th Annual Symposium on Foundations of Computer Science, 1985, pp. 74–78.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
006

THE COMPUTATIONAL COMPLEXITY OF DIVISION IN QUADRATIC EXTENSION FIELDS*

THOMAS LICKTEIG†

Abstract. We show that, for any quadratic field extension $k \subset K$, the division in K requires exactly six essential multiplications and divisions over k. In particular, complex division in real arithmetic requires six multiplications and divisions.

If additions and subtractions are counted as well, exactly nine operations are required iff $k \subset K$ is a radical extension;[1] otherwise ten operations are required.

Key words. arithmetic in quadratic extension fields, straight line algorithms, computational complexity of rational functions, symmetry group of computation problems, adjunction of intermediate results to the ground field

AMS(MOS) subject classifications. 68C20, 68C25

0. Introduction. Let $k \subset K$ be a quadratic field extension, $K = k(\omega)$, where

$$\omega^2 + a\omega + b = 0 \qquad (a, b \in k). \tag{0.1}$$

The division of two "general" elements from K,

$$\frac{X\omega + Y}{U\omega + V} = \frac{XV - YU}{N}\omega + \frac{Y(V - aU) + bXU}{N}, \tag{0.2}$$

where $N = (U\omega + V)(U\bar{\omega} + V) = bU^2 - aUV + V^2$ [$\bar{\omega} = -a - \omega$] denotes the *norm* of $U\omega + V$, leads to the computational task of evaluating the rational functions

$$\frac{XV - YU}{N}, \qquad \frac{Y(V - aU) + bXU}{N}. \tag{0.3}$$

Let L denote the minimum number of nonscalar multiplications or divisions sufficient to compute these rational functions in $k(X, Y, U, V)$ by a straight line algorithm starting from the elements of k and the indeterminates X, Y, U, V (representing the unknown inputs).

The problem originates from the numerical calculation with complex numbers, where it is a basic question of how difficult it is to compute the imaginary and the real part of the quotient of two complex numbers over the reals. The multiplication is known to have complexity three (by a series of results [1]-[3], [7], [11], [13], [18], [22], [23]).

In this article we show that $L = 6$ (Theorem 5.1), the upper bound coming from the "computation sequence"

$$\frac{V}{U}, \quad V \cdot \frac{V}{U}, \quad X \cdot \frac{V}{U}, \quad \frac{X \cdot (V/U) - Y}{bU - aV + V \cdot (V/U)}, \quad Y \cdot \frac{V}{U}, \quad \frac{Y \cdot (V/U) - aY + bX}{bU - aV + V \cdot (V/U)} \tag{0.4}$$

* Received by the editors May 22, 1984; accepted for publication (in revised form) May 5, 1986.

† Mathematisches Institut der Universität Tübingen, Auf der Morgenstelle 10, D 74 Tübingen, West Germany.

[1] Throughout this paper $k \subset K$ is called a radical extension if $K = k(\omega)$ with $\omega^2 \in k$.

(Smith [16]), or from the pure division variant (for aesthetes among the readers)

$$\frac{U}{V},\ \frac{V}{(U/V)},\ \frac{X}{(U/V)},\ \frac{X/(U/V)-Y}{bU-aV+V/(U/V)},\ \frac{Y}{(U/V)},\ \frac{Y/(U/V)-aY+bX}{bU-aV+V/(U/V)}. \tag{0.5}$$

In particular, the complexity of complex division over the reals is six. This proves a conjecture by H. Alt and J. van Leeuwen [2], who gave a lower bound of five (cf. also [10, p. 647]). Although I have no rigorous proof, I think that (0.4) minimizes the number of divisions in an optimal computation sequence for (0.3).

It is worth noting that in the complex case ($a=0$, $b=1$) this computation can be organized in a numerically stable manner by working as in (0.4), if U has a larger absolute value than V, and otherwise performing an exchange according to

$$\begin{pmatrix} X \\ Y \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} X \\ Y \end{pmatrix}, \qquad \begin{pmatrix} U \\ V \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} U \\ V \end{pmatrix} \tag{0.6}$$

beforehand [16]. By a general principle due to V. Strassen [20] the reaction time of such a branching algorithm is at least L on almost all inputs.

A special situation arises if k is a real closed field, where the problem can be stated as a formula in a decidable theory. Then $K=k(i)$ where $i^2=-1$. Since L does only depend on the fields k, K and not on the choice of the generator ω, we may put $\omega=i$, i.e. $a=0$ and $b=1$ [in (0.1)]. Then the "imaginary" and the "real" part (0.3) of the quotient (0.2) are defined over the rationals, and the inequality $L\leqq 5$ can be expressed in the language of the elementary theory of real closed fields (cf. [21]). However, a corresponding existential formula would contain 90 variables, and the polynomials involved (in these 90 variables) would have degrees up to 34. So a machine decision, even with the fastest quantifier elimination procedures (cf. [5], [6], [17]), seems to be impractical. Moreover, it is a lucky fact that there is the uniform answer six for all quadratic extensions $k\subset K$. An application of the generality of the result comes out in [12].

An outline of our general plan to prove the absurdity of the assumption $L\leqq 5$ is as follows: First we study the special division problem where X is replaced by the constant 1 and show that it requires exactly five multiplications or divisions over k (Theorem 3.3). This result, together with a suitable reduction (Conclusion 3.28), gives us the information that multiplications or divisions of a certain type cannot occur in a hypothetic straight line algorithm of length five for the general problem (0.3). Secondly, the symmetry group Γ of the division (a group of linear variable transforms leaving fixed the computation problem (§ 2)) enables us to bring a hypothetic straight line algorithm of length five for the rational functions (0.3) into a form, in which already, after four (counted) multiplication or division steps, $(XV-YU)/N$ is computed, and in which, prior to the fourth step, the algorithm computes V^2/U. The latter is the point where the "crowbar" applies. It turns out that this is enough to determine the results of the first four multiplication or division steps of the algorithm explicitly (Proposition 4.1), closely related to the initial segments of computation sequences (0.4), (0.5). Then, it remains to falsify the assumption that one further multiplication or division step is sufficient to compute $(Y(V-aU)+bXU)/N$, too.

Our essential tools are adjunctions of intermediate results to the ground field (§ 1), a more thorough utilization of Pan's [15] substitution method, and simplification of algorithms by means of transformation via the symmetry group.

While the nonscalar complexity (Ostrowski complexity) does not depend on specific properties of the extension $k \subset K$, a distinction comes in if additions and subtractions are counted as well: Nine operations are required iff $k \subset K$ is a radical extension; otherwise ten are required (Theorem 6.8). The nonscalar complexity does not depend on the underlying k-basis (ω_1, ω_2) of K, so we could choose (in (0.2)) a basis $(\omega, 1)$ for some fixed ω. Counting also additions and subtractions, costs become dependent on the choice of the basis. Minimizing costs of division in basis (ω_1, ω_2) over all k-bases (ω_1, ω_2) of K we get the above values.

Finally we remark that the norm $N = bU^2 - aUV + V^2 \in k[U, V]$ is irreducible, a fact that will be used in many places.

1. Substitution, and adjunction of intermediate results to the ground field. Let k be a field and A be a commutative k-algebra. Following Strassen [19], we computationally consider A as a commutative k-algebra with division by units over some set of generators $B \subset A$, i.e. we consider the partial algebra (A, Ω_B^k) of the type (=set of operation symbols plus arity function)

$$\Omega_B^k := \{+, -, *, /\} \sqcup k \sqcup (B \cup \{0, 1\}) \qquad \text{[disjoint union]}.$$

Thus we have in A the binary operations $+$, $-$, $*$, $/$[Def $(/) = A \times A^*$, $A^* = \{\text{units of } A\}$], the multiplication by scalars from k (unary), and the elements from $B \cup \{0, 1\}$ as constants (o-ary). We suppress explicit distinction between operation symbols and their interpretation in A. If A is a field A is called a k-field [19] over B. We are mainly interested in rational function fields $A = k(X_1, \cdots, X_n)$. In this case we always assume $B = \{X_1, \cdots, X_n\}$ if not stated otherwise.

In the sequel we will study straight line computations in A, and we will use the nonscalar cost function $z = 1_{\{*,/\}}$ on Ω_B^k. z is intimately related to Ω_B^k, so notationally we save explicit mentioning of z. Let $\langle F \rangle$ denote the k-linear span of some subset $F \subset A$. We put

$$\Lambda = \Lambda_B = \langle B \cup \{0, 1\} \rangle.$$

Straight line algorithms in A starting from the elements of $B \cup \{0, 1\}$ (Ω_B^k-computations, in the terminology of [19]) together with the nonscalar cost measure (Ostrowski measure) can be compressed by the definition of computation sequences:

DEFINITION 1.1. (i) A sequence $(p_1, \cdots, p_r) \in A^r$ is called a (A, Ω_B^k)-*computation sequence* (or simply a computation sequence) if for any $\rho \leqq r$ there are $u_\rho, v_\rho \in \langle p_1, \cdots, p_{\rho-1} \rangle + \Lambda$ such that

$$p_\rho = u_\rho * v_\rho \quad \text{or } p_\rho = u_\rho / v_\rho \quad \text{and } v_\rho \in A^*.$$

We call $\langle p_1, \cdots, p_r \rangle + \Lambda$ the *computation hull* of $(p_1, \cdots, p_r)$. The set of all (A, Ω_B^k)-computation sequences (of any length) is denoted by $CS(A, \Omega_B^k)$.

(ii) $(p_1, \cdots, p_r) \in CS(A, \Omega_B^k)$ is called a $((A, \Omega_B^k))$-computation sequence for some subset $F \subset A$ if

$$F \subset \langle p_1, \cdots, p_r \rangle + \Lambda.$$

If the computation sequence has minimal length r, it is called *optimal*, and r is called the *complexity* $L(F)$ of F in (A, Ω_B^k).

(iii) The complexity $L(F \bmod E)$ of F relative to some $E \subset A$ is defined as the complexity of F in $(A, \Omega_{B \cup E}^k)$ [19].

Sometimes we will use notational variants such as $L_A(F)$ in order to stress the structure. By the above definition only multiplications and divisions are counted, while k-linear operations and constants are left free of charge (compare [14], and [4, p. 10]).

Note that the optimality of the computation sequence implies the linear independence of $(p_1, \cdots, p_r)$ modulo Λ.

An important tool for proving lower bounds on the complexity of rational functions is Pan's substitution method [15]. Let A be a k-field over $B \subset A$, let Y be an indeterminate over A, and take $\Omega^k_{B\cup\{Y\}}$ as type for $A(Y)$. We recall the substitution technique: Assume $(p_1, \cdots, p_r) \in CS(A(Y), \Omega^k_{B\cup\{Y\}})$ to be optimal for some $F \subset A(Y)$ (i.e. $r = L_{A(Y)}(F)$), and assume $p_1, \cdots, p_{\rho-1} \in A$ but $p_\rho \notin A (\rho \leqq r)$. By the definition of a computation sequence there are $u_\rho, v_\rho \in \langle p_1, \cdots, p_{\rho-1}\rangle + \Lambda_{B\cup\{Y\}}$ such that

$$p_\rho = u_\rho * v_\rho \quad \text{or } p_\rho = u_\rho / v_\rho \quad (\text{and } v_\rho \in A(Y)^* = A(Y)\backslash\{0\}).$$

Observe that the nonscalar cost measure allows shifting factors $\in k$ from one to the other operand. So, according to whether the second operand involves Y or not, p_ρ can be written as

$$p_\rho = (\cdot\cdot\cdot)*/(Y - f) \quad \text{for some } f \in \langle p_1, \cdots, p_{\rho-1}\rangle + \Lambda_B \subset A \tag{1.2a}$$

or as

$$p_\rho = \frac{Y - f}{g} \quad \text{for some } f, g \in \langle p_1, \cdots, p_{\rho-1}\rangle + \Lambda_B \subset A \tag{1.2b}$$

where in the multiplicative case w.l.o.g. the second operand is assumed not to be Y free. The idea of the substitution method is to eliminate (trivialize) the ρth multiplication or division step by a specialization

$$\sigma : A(Y) \to A, \qquad Y \mapsto \alpha \cdot h + f \quad (\alpha \in k)$$

where $h = 1$ in case of (1.2a), or $h = g$ in case of (1.2b), in order to get a (A, Ω^k_B)-computation sequences $(\sigma(p_1), \cdots, \sigma(p_{\rho-1}), \sigma(p_{\rho+1}), \cdots, \sigma(p_r))$ of length $r-1$ for $\sigma(F)$. Then one has the lower bound $L_{A(Y)}(F) \geqq L_A(\sigma(F)) + 1$.

However, there is the problem that "division by zero" may happen, in particular if σ is not totally defined. Its domain is the local k-algebra $\mathcal{O}_\sigma = \{a/b : a, b \in A[Y],$ $b(\alpha h + f) \neq 0\}$ $(\mathcal{O}^*_\sigma = \{a/b : a(\alpha h + f) \neq 0,\ b(\alpha h + f) \neq 0\})$. Therefore the necessary and sufficient condition for the correctness of the above argument is

$$(p_1, \cdots, p_r) \in CS(\mathcal{O}_\sigma, \Omega^k_{B\cup\{Y\}}).$$

Now, by the finiteness of its length, any $(A(Y), \Omega^k_{B\cup\{Y\}})$—computation sequence is a $(A[Y][d^{-1}], \Omega^k_{B\cup\{Y\}})$—computation sequence for some "denominator" $d \in A[Y]$, and the condition

$$A[Y][d^{-1}] \subset \mathcal{O}_\sigma$$

always can be achieved by a suitable choice of the parameter $\alpha \in k$, provided the base field k is infinite. Of particular importance for the results of this paper is to vary $\alpha \in k$, i.e. exploiting the fact that all but finitely many $\alpha \in k$ are suited. So far this has not been utilized in the literature and our results substantially rely on that fact [see §§ 3–5].

Now, to make things static, let t be a second indeterminate over A, and consider the fields

$$\begin{array}{ccc} & A(Y, t) & \\ \cup & & \cup \\ A(Y) & & A(t). \\ \cup & & \cup \\ & A & \end{array}$$

We choose $\Omega^{k(t)}_B$ as type for $A(t)$ and $\Omega^{k(t)}_{B\cup\{Y\}}$ as type for $A(Y, t)$. So these are viewed as $k(t)$-fields. (In contrast to Y we use a lower case t to indicate the computational

difference between the variables Y and t.) Throughout this paper we will use substitution in the following manner: Consider the computation sequence

$$(p_1, \cdots, p_r) \in CS(A(Y), \Omega^k_{B \sqcup \{Y\}}) \quad \text{for } F \tag{1.3}$$

as a $(A(Y, t), \Omega^{k(t)}_{B \sqcup \{Y\}})$—computation sequence. Then the substitution

$$\sigma : A(Y, t) \to A(t), \qquad Y \mapsto t \cdot h + f, \quad t \mapsto t \tag{1.4}$$

produces a computation sequence

$$(\sigma(p_1), \cdots, \sigma(p_{\rho-1}), \sigma(p_{\rho+1}), \cdots, \sigma(p_r)) \in CS(A(t), \Omega^{k(t)}_B) \quad \text{for } \sigma(F) \tag{1.5}$$

since $A(Y)$ lies in the domain $\mathcal{O}_\sigma = \{a/b : a, b \in A[Y, t], b(th+f, t) \neq 0\}$, by construction of σ. (By the transcendence of $t \cdot h + f$ over A the denominator problem is circumvented (by $0 \neq h \in \{1, g\}$, $f, g \in A$; see (1.2)). Moreover σ restricts to a "birational" mapping $A(Y) \to A(t)$ (inverse: $t \mapsto (Y-f)/h$), so the natural connection of (1.5) to the original computation sequence is not untied (as it is if t is chosen in k). Taking back the structure from $A(t)$ to $A(Y)$, we get the interesting interpretation of a change of structure:

LEMMA 1.6. *Let A be a k-field over some $B \subset A$, Y an indeterminate over A, and let $A(Y)$ be of type $\Omega^k_{B \sqcup \{Y\}}$. Assume $(p_1, \cdots, p_r)$ to be a $(A(Y), \Omega^k_{B \sqcup \{Y\}})$-computation sequence for some $F \subset A(Y)$ with $p_1, \cdots, p_{\rho-1} \in A$ but $p_\rho \notin A (\rho \leqq r)$. Then*

$$p_\rho = (\cdots) * / (Y - f) \quad \text{or} \quad p_\rho = \frac{Y-f}{g}$$

for some $f, g \in \langle p_1, \cdots, p_{\rho-1} \rangle + \Lambda_B$, and

$$(p_1, \cdots, p_{\rho-1}, p_{\rho+1}, \cdots, p_r) \in CS(A(Y), \Omega^{k(t)}_B)$$

is a computation sequence for F of length $r-1$ where $t = Y - f$ or $t = (Y-f)/g$, respectively (in particular $A(Y) = A(t)$).

Remarks. In the sequel we will use the phrase "*adjoining the intermediate result t to the ground field k and deleting the o-ary Y*" to denote the process described in the lemma (a structural expansion followed by a restriction; note that deletion of Y is no sharpening if $f, g \in \Lambda_B$). The main advantage of adjunction (compared with usual substitution) is the fact that the results of the computation sequence are not changed. So a change of the point of view can be used to prove properties of the original computation sequence (see §§ 3–5 for applications). In the individual case also other changes of structure may be favorable. For instance, in the first case it would be sufficient to allow only multiplication, resp. division, by $t = Y - f$ for free. In the second case, $(p_1, \cdots, p_{\rho-1}, p_{\rho+1}, \cdots, p_r) \in CS(A(Y), \Omega^k_{B \sqcup \{t\}})$ if something is known about "access to Y": $p_\nu = u_\nu * / v_\nu$ for some $u_\nu, v_\nu \in \langle p_1, \cdots, p_{\nu-1} \rangle + \Lambda_B$ for $\nu = \rho+1, \cdots, r$. However, we will not make use of these here. In combination with the symmetry group (§ 3) we sometimes will prefer the substitution point of view (if more convenient).

We recall the notion of autarky from [19]: For partial algebras A, A' of arbitrary types Ω, Ω', and $\varphi : \Omega \to \Omega'$ arity preserving, $z' : \Omega' \to \mathbb{R}^+$ some costs function, a φ-homomorphism $f : A \to A'$ is called *autarkical* for some finite $F \subset A$ (see [19] for definitions) if for the pull-back cost function $z' \circ \varphi$

$$L_{z' \circ \varphi}(F) = L_{z'}(f(F));$$

f is called *autarkical* if this holds for any F.

Examples (cf. [19]). Let A be a k-field over $B \subset A$. The embedding

$$(A, \Omega^k_B) \hookrightarrow (A, \Omega^k_{B \cup C}) \quad \text{is autarkical if } C \subset \Lambda_B \tag{1.7}$$

since for all F: $L(F \bmod C) = L(F)$ iff $L(C) = 0$ (cf. [19, Cor. 14]). For a transcendental extension $A(t)$ of A there are two canonical types, namely $\Omega^k_{B\sqcup\{t\}}$ *and* $\Omega^{k(t)}_B$. As is well known, for infinite k,

$$(A, \Omega^k_B) \hookrightarrow (A(t), \Omega^{k(t)}_B) \quad \text{is autarkical} \quad (|k| = \infty) \tag{1.8}$$

(by a specialization argument [19]). (1.7) and (1.8) imply that also

$$(A, \Omega^k_B) \hookrightarrow (A(t), \Omega^k_{B\sqcup\{t\}}) \quad \text{is autarkical} \quad (|k| = \infty).$$

The latter has a strengthening:

LEMMA 1.9. *Let A be a k-field over $B \subset A$, Y an indeterminate over A, k infinite. Then we have: If $(p_1, \cdots, p_r) \in CS(A(Y), \Omega^k_{B\cup\{Y\}})$ is optimal for $F \subset A$ then $(p_1, \cdots, p_r) \in CS(A, \Omega^k_B)$.*

Proof. By the way of contradiction assume that one computational step involves Y; thus $p_\rho \notin A$ but $p_1, \cdots, p_{\rho-1} \in A$ for some $\rho \leqq r$. By Lemma 1.6 there is some intermediate result $t \in A(Y)$ with $A(t) = A(Y)$ such that $(p_1, \cdots, p_{\rho-1}, p_{\rho+1}, \cdots, p_r)$ is a $(A(t), \Omega^{k(t)}_B)$-computation sequence for $F \subset A$. By the autarky statement (1.8) this implies that

$$r - 1 \geqq L_A(F) \geqq L_{A(Y)}(F),$$

contradicting the optimality of the computation sequence. □

2. The symmetry group of the division. From now on we consider rational function fields A. One of the advantages of the Ostrowski complexity measure is the invariance of complexity under regular linear affine variable transforms $\mathfrak{a}$. Let us define the *symmetry group* of $F \subset A$ as

$$\Gamma = \Gamma(F) = \{\mathfrak{a} : L(F \bmod \mathfrak{a}(F)) = 0\}.$$

In our case, let $A = k(X, Y, U, V)$ [type: $\Omega^k_{\{X,Y,U,V\}}$] where X, Y, U, V denote indeterminates over the field k, and let $k \subset K$ be of degree two. Consider for $\zeta, \theta \in K^*$, $\eta \in K$ the identity (in $K(X, Y, U, V)$)

$$\frac{X\omega + Y}{U\omega + V} = (\zeta^{-1}\theta) * \frac{\zeta * (X\omega + Y) + \eta * (U\omega + V)}{\theta * (U\omega + V)} - (\zeta^{-1}\eta), \tag{2.1}$$

algorithmically read as follows: Alter a given division algorithm by changing the input data k-linearly (at zero cost) according to multiplication by ζ, η, θ and adding results (e.g. $U\omega + V$ changes according to

$$\begin{pmatrix} U \\ V \end{pmatrix} \mapsto \begin{pmatrix} f - ae & e \\ -be & f \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix}$$

if $\theta = e\omega + f$), then use the given division algorithm, and finally correct the "wrong" quotient according to multiplying it by $\zeta^{-1}\theta$ and subtracting $\zeta^{-1}\eta$, using k-linear operations only. Thus this identity induces an embedding of the group $\Delta(2, K)$ of regular upper triangular 2×2 matrices

$$\begin{pmatrix} \zeta & \eta \\ 0 & \theta \end{pmatrix}$$

over K into the symmetry group Γ of the rational functions (0.3) (compare also exchange possibility (0.6) as an application of

$$\begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix} \in \Delta(2, \mathbb{C}).$$

Similarly, division of conjugates

$$\frac{X\omega+Y}{U\omega+V}=\left(\frac{\overline{X\bar{\omega}+Y}}{U\bar{\omega}+V}\right) \qquad (\bar{\omega}=-a-\omega)$$

yields an embedding of Aut (K/k) into Γ. As indicated by the above discussion, Γ can be used to transform computation sequences into other ones of the same length (compare also [8], [9] for this technique). This idea will serve to simplify computation sequences; for special manipulations it is useful to distinguish the following subgroups of $\Delta(2, K)$ (mainly used in the sequel):

$$\Gamma_N=\left\{\begin{pmatrix}\zeta & 0\\ 0 & 1\end{pmatrix}:\zeta\in K^*\right\} \qquad (N=\text{numerator}),$$

$$\Gamma_D=\left\{\begin{pmatrix}1 & 0\\ 0 & \theta\end{pmatrix}:\theta\in K^*\right\} \qquad (D=\text{denominator}),$$

$$\Gamma_+=\left\{\begin{pmatrix}1 & \eta\\ 0 & 1\end{pmatrix}:\eta\in K\right\}.$$

It turns out that the transforms mentioned above generate Γ.

PROPOSITION 2.2. *The symmetry group* Γ *of the division in a quadratic extension field* $K\supset k$ *is determined as*

$$\Gamma=\Delta(2, K)\cdot\mathrm{Aut}(K/k).$$

Proof. First of all, observe that $\Delta(2, K)$ and Aut (K/k) (embedded into Γ) commute, so their product is a subgroup of Γ. Let $\alpha\in\Gamma$ be given by $\tilde{X}, \tilde{Y}, \tilde{U}, \tilde{V}\in\Lambda[\Lambda=\langle X, Y, U, V, 1\rangle]$. By definition, $\alpha\in\Gamma$ means

$$\left\langle\frac{\tilde{X}\tilde{V}-\tilde{Y}\tilde{U}}{N(\tilde{U},\tilde{V})}, \frac{\tilde{Y}(\tilde{V}-a\tilde{U})+b\tilde{X}\tilde{U}}{N(\tilde{U},\tilde{V})}\right\rangle+\Lambda=\left\langle\frac{XV-YU}{N(U,V)}, \frac{Y(V-aU)+bXU}{N(U,V)}\right\rangle+\Lambda. \tag{2.3}$$

By the irreducibility of $N\in k[U, V]$, a comparison of denominators gives

$$N(\tilde{U},\tilde{V})=\lambda\cdot N(U,V) \quad \text{for some } \lambda\in k^*.$$

N factors linearly over the extended coefficient field K. So we get

$$(\tilde{U}\omega+\tilde{V})(\tilde{U}\bar{\omega}+\tilde{V})=\lambda\cdot(U\omega+V)(U\bar{\omega}+V) \qquad (\bar{\omega}=-a-\omega).$$

Now use unique factorization in $K[X, Y, U, V]$, and change α by means of Aut (K/k) in order to obtain

$$\tilde{U}\omega+\tilde{V}\in K\cdot(U\omega+V).$$

So by the action of Γ_D we may assume that

$$\tilde{U}=U, \qquad \tilde{V}=V. \tag{2.4}$$

It remains to prove a relation

$$\tilde{X}\omega+\tilde{Y}\in K\cdot(X\omega+Y)+K\cdot(U\omega+V)$$

in $K[X, Y, U, V]$. To this end, we extend the coefficient field k in (2.3) by ω, multiply by the linear $U\omega+V\in K[U, V]$ and intersect by $K[X, Y, U, V]$. By (2.4) (using (0.2)) this yields

$$K\cdot(\langle\tilde{X}\omega+\tilde{Y}\rangle+(U\omega+V)\Lambda)=K\cdot(\langle X\omega+Y\rangle+(U\omega+V)\Lambda),$$

implying the desired relation (here the prefix $K\cdot$ indicates that these linear spans are to be taken over the extended coefficient field K). □

3. Special divisions. We begin with the remark that, proving a lower bound, we may assume the base field k to be infinite, since we can adjoin one and the same transcendental element to K and to k (by a linear disjointness argument we again get a quadratic extension). Therefore infinity of k will be tacitly assumed in the sequel.

We introduce some notation used from now on: If p is a polynomial then $p^{(j)}$ denotes the homogeneous part of degree j of p; for a set M of polynomials we let $M^{(j)}=\{p^{(j)}: p\in M\}$. We write $p|q$ if p divides q, $p\nmid q$ if not.

Let us first consider the division of an element of k ($X=0$) by an element of K, especially computing the reciprocal of an element of K ($X=0$, $Y=1$).

PROPOSITION 3.1. *For any quadratic field extension $k\subset K$ the complexity of division of elements from k by elements from K and the complexity of inversion of elements from K over k are both four.*

Proof. The proof is similar as in [2]. Since the upper bounds are clear from the computation sequence

$$\frac{V}{U},\quad V\cdot\frac{V}{U},\quad \frac{-Y}{bU-aV+V\cdot(V/U)},\quad \frac{-Y}{bU-aV+V\cdot(V/U)}\cdot\left(a-\frac{V}{U}\right),$$

it suffices to show the lower bounds. (Reciprocal.) Assume $(p_1,\cdots,p_r)$ to be an optimal computation sequence for $\{-U/N,(V-aU)/N\}$, so

$$\left\langle\frac{-U}{N},\frac{V-aU}{N}\right\rangle\subset\langle p_1,\cdots,p_r\rangle+\Lambda.$$

Counting dimensions, this subspace and $\langle p_1,\cdots,p_{r-1}\rangle+\Lambda$ intersect nontrivially. Since any nonzero element from the left-hand side is contained in the orbit $\Gamma_D(U/N)$ (cf. (2.1)), we may assume that

$$\frac{U}{N}\in\langle p_1,\cdots,p_{r-1}\rangle+\Lambda,$$

whence $r-1\geqq L(U/N)\geqq 3$, by Lemma 3.2 (ii) below and the irreducibility of $N\in k[U,V]^{(2)}$. Thus $r\geqq 4$. We reduce the remaining statement to the first proved one:

$$L\left(\frac{-YU}{N},\frac{Y(V-aU)}{N}\right)\geqq L\left(\frac{-U}{N},\frac{V-aU}{N}\right)\geqq 4.$$

To see this, enrich computation theoretic structure by adjoining Y to k. Then replace U, V by YU, YV (a use of Γ_D) to make things Y free. By autarky (1.8) this implies the above inequality. □

Next we list some auxiliary facts:

LEMMA 3.2. *Let $p,q\in k[X_1,\cdots,X_n]$ be relatively prime, k being any field.*

(i) *If $L(p/q)\leqq 1$ then $p^{(\deg p)}$ is a product of linear forms and $q^{(\deg q)}|p^{(2)}$.*

(ii) *If $L(p/q)\leqq 2$ then $q^{(\deg q)}$ is a product of linear forms.*

Proof. In the first case p/q has the form $l_1l_2+l_3$ or $l_1/l_2+l_3=(l_1+l_2l_3)/l_2$ for suitable l_1, l_2, $l_3\in\Lambda$; in the second case, for instance if two divisions happen,

$$\frac{p}{q}=\frac{\alpha l_1/l_2+l_3}{\beta l_1/l_2+l_4}+\gamma\frac{l_1}{l_2}+l_5=\frac{(\alpha l_1+l_2l_3)l_2+\gamma(\beta l_1+l_2l_4)l_1}{(\beta l_1+l_2l_4)l_2}+l_5$$

for suitable α, β, $\gamma\in k$ and $l_1,\cdots,l_5\in\Lambda$. From these representations the statements are read off. The other cases are similar and left to the reader. (Similar arguments are also used in [2].) □

Inversion can be viewed as "zero-dimensional affine numerator restriction". Next we consider a "one-dimensional affine numerator restriction." Let us call $e\omega+f\in K$ *normalized* if $e=1$.

THEOREM 3.3. *For any quadratic field extension $k\subset K=k(\omega)$ the complexity of the division*

$$\frac{\omega+Y}{U\omega+V}$$

of normalized elements from K by arbitrary elements from K over k is five.

The proof of this theorem, being more involved than the previous one (3.1), is based on a uniqueness result for optimal computation sequences for $(V-YU)/N$ (Proposition 3.16 below). We start with following purely preparative

LEMMA 3.4. *For any ground field k we have:*

(i) $L(V^2/U)=2$;

(ii) *Any optimal computation sequence (p_1, p_2) for V^2/U satisfies one of the following statements:*

1) $\langle p_1\rangle+\Lambda_{\{U,V\}}=\left\langle\frac{V}{U}\right\rangle+\Lambda_{\{U,V\}}$,

2) $\langle p_1\rangle+\Lambda_{\{U,V\}}=\left\langle\frac{U}{\sigma U+V}\right\rangle+\Lambda_{\{U,V\}}$ *for some* $\sigma\in k$,

3) $\langle p_1\rangle+\Lambda_{\{U,V\}}=\langle(\sigma U+V)(\tau U+V)\rangle+\Lambda_{\{U,V\}}$ *for some* $\sigma, \tau\in k$.

Proof. Since the first part is trivial (use Lemma 3.2 (i)), we treat the second one and look at the relation

$$\frac{V^2}{U}\in\langle p_1, p_2\rangle+\Lambda_{\{U,V\}}\subset k(U, V)\qquad\text{(by Lemma 1.9).}\tag{3.5}$$

Clearly, we have at least one division since V^2/U is not a polynomial. So we have three cases:

Case $(/, *)$. $p_1=l_1/l_2$ and

$$p_2=(\alpha p_1+l_3)(\beta p_1+l_4)=\frac{(\alpha l_1+l_2l_3)(\beta l_1+l_2l_4)}{l_2^2}=\frac{A}{l_2^2}$$

(for short) where $\alpha, \beta\in k$, $l_1, l_2\in\Lambda_{\{U,V\}}$ relatively prime, and w.l.o.g.

$$l_3, l_4\in\Lambda^{(1)}_{\{U,V\}}\tag{3.6}$$

since constant terms in l_3, l_4 do not change the computation hull. After multiplying by Ul_2^2 (3.5) becomes

$$l_2^2V^2\in\langle Ul_2l_1, UA\rangle+Ul_2^2\Lambda_{\{U,V\}}.\tag{3.7}$$

Thus $\langle l_2\rangle=\langle U\rangle$ since the left-hand side must be divisible by U. (This can be seen directly from the poles in (3.5), but for reasons of a uniform presentation we prefer the polynomial argumentation.) We show that l_1 has no constant term, proving Lemma 3.4 (ii) 1 in Case $(/, *)$ since $l_2\nmid l_1$. Since UA must participate in any linear combination behind (3.7) (i.e., with a nonzero coefficient) we get $U^2|UA$, whence $\beta=0$ (say) for $l_2\nmid l_1$. Comparing the homogeneous parts of degree two in (3.7) (by (3.6)) we immediately obtain $l_1^{(0)}=0$, or otherwise Ul_2l_1 does not participate in any linear combination behind (3.7). But then $\alpha\neq 0$ since the left-hand side in (3.7) is not divisible by U^3. By the same reasoning $l_4\notin\langle U\rangle$, and therefore the nonzero l_4 is not V free (by (3.6)). Comparing the homogeneous parts of degree three likewise, we conclude that $l_1^{(0)}=0$.

Case (/, /). $p_1 = l_1/l_2$ and

$$p_2 = \frac{\alpha p_1 + l_3}{\beta p_1 + l_4} = \frac{\alpha l_1 + l_2 l_3}{\beta l_1 + l_2 l_4} = \frac{A}{B}$$

(for short) where $\alpha, \beta \in k$ and $l_1, \cdots, l_4 \in \Lambda_{\{U,V\}}$. Multiplying (3.5) by Ul_2B, we get

$$l_2 B V^2 \in \langle UBl_1, Ul_2A \rangle + Ul_2B\Lambda_{\{U,V\}}. \tag{3.8}$$

The homogeneous parts of degree five tell us that deg $B \leqq 1$. So $l_4 \in k$, and then $\beta \neq 0$, by the optimality of the computation sequence. We do not change $\langle p_1 \rangle + \Lambda$ by adding a multiple of l_2 to l_1. Therefore we may assume that $l_4 = 0$, whence $\langle l_1 \rangle = \langle B \rangle$. (3.8) implies that $U | l_2 B$. In case of $\langle l_2 \rangle = \langle U \rangle$, adjunction of U to k and deletion of the o-ary U would imply that V^2 could be computed with one division over the extended coefficient field $k(U)$, this being impossible. Therefore

$$\langle l_1 \rangle = \langle B \rangle = \langle U \rangle \text{ and } U \nmid l_2. \tag{3.9}$$

We show that l_2 has no constant term, proving Lemma 3.4 (ii) 2 in Case (/, /). Assume contrary that $l_2^{(0)} \neq 0$. Then the homogeneous parts of degree one in (3.8) tell us that $l_3^{(0)} = 0$ since Ul_2A must participate in any linear combination behind (3.8). But then the homogeneous parts of degree two tell us that $l_3 \in \langle U \rangle$, and (3.8) would imply that $U | l_2$, contradicting (3.9).

Case (*, /). $p_1 = l_1 l_2$ and $p_2 = (\alpha p_1 + l_3)/(\beta p_1 + l_4)$ where $\alpha, \beta \in k$, w.l.o.g. $l_1, l_2 \in \Lambda^{(1)}_{\{U,V\}}$, and $l_3, l_4 \in \Lambda_{\{U,V\}}$. If p_1 has not the stated form, then $\langle l_1 \rangle = \langle U \rangle$, say. But then adjunction of U to k and deletion of the o-ary U would imply that V^2 could be computed with one division, being impossible. This proves Lemma 3.4 (ii) 3 in Case (*, /). □

LEMMA 3.10. *For any ground field k we have:*

(i) $L(H - (U/N)Y) \geqq 3$ *for every rational function* $H - (U/N)Y \in k(U, V, Y)$ *with* $H \in k(U, V)$ *and irreducible* $N \in k[U, V]^{(2)}$;

(ii) *If* $L(H - (U/N)Y) = 3$, *and* (p_1, p_2, p_3) *is an optimal computation sequence for* $H - (U/N)Y$, *then*

1) (p_1, p_2) *is an optimal computation sequence for* V^2/U;

2) p_3 *is linear in* Y, $\deg_Y p_3 = 1$, *and* p_3 *must result from a division by a multiple of* N/U *so that* $\langle p_3 \rangle = \langle (Y - F)/(N/U) \rangle$ *for some* $F \in \langle p_1, p_2 \rangle + \Lambda_{\{U,V\}} \subset k(U, V)$.

Proof. (i) The coefficient of Y shows that the denominator of a relatively prime quotient representation of $H - (U/N)Y$ has the irreducible $N \in k[U, V]^{(2)}$ as a divisor. So $L(H - (U/N)Y) \leqq 2$ is impossible by Lemma 3.2 (ii).

(ii) Now, assume (p_1, p_2, p_3) to be an optimal computation sequence for $H - (U/N)Y$, so

$$H - \frac{U}{N} Y \in \langle p_1, p_2, p_3 \rangle + \Lambda, \tag{3.11}$$

and let $p_\rho \notin k(U, V)$ by $p_1, \cdots, p_{\rho-1} \in k(U, V)$ ($\rho \leqq 3$). p_ρ can be written as

$$p_\rho = (\cdots)*/(Y - F), \qquad F \in \langle p_1, \cdots, p_{\rho-1} \rangle + \Lambda_{\{U,V\}} \tag{3.12a}$$

or as

$$p_\rho = \frac{Y - F}{G}, \qquad F, G \in \langle p_1, \cdots, p_{\rho-1} \rangle + \Lambda_{\{U,V\}}. \tag{3.12b}$$

For the moment, we adjoin to k the intermediate result

$$t = \frac{Y - F}{R} \tag{3.13}$$

where $R=1$ in case of (3.12a), or $R=G$ in case of (3.12b), and delete the o-ary Y. So we have the computation sequence of length two

$$(p_1, \cdots, p_{\rho-1}, p_{\rho+1}, \cdots, p_3) \in CS(k(U, V, Y), \Omega^{k(t)}_{\{U,V\}}) \quad \text{for } H - \frac{U}{N}Y.$$

Although having changed computation theoretic interpretation, U, V, t remain algebraically independent generators of $k(U, V, Y)$. So we may expand in powers of the intermediate result t:

$$H - \frac{U}{N}Y = \left(H - \frac{U}{N}F\right) - R\frac{U}{N} \cdot t \qquad \text{(by (3.13))}.$$

If $R=1$ (case (3.12a)), then the coefficient of t shows that the denominator q in a relatively prime quotient representation

$$H - \frac{U}{N}Y = \frac{p}{q}, \qquad p, q \in k[U, V, t]$$

has the irreducible $N \in k[U, N]^{(2)}$ as a divisor. Adjoining t to k this property remains valid (i.e. with $p, q \in k(t)[U, V]$), contradicting Lemma 3.2 (ii). So p_ρ must result from a division of the type (3.12b). Write $R = G = P/Q$ with $P, Q \in k[U, V]$ relatively prime. By the very same argument as above we obtain from Lemma 3.2 (ii) that $N|P$. With this insight in mind we return to the original computation theoretic structure and obtain from Lemma 3.2 (i) $L(G) \geqq 2$ since $G \neq 0$. On the other hand, $L(G) \leqq \rho - 1 < 3$ (by (3.12b)); thus

$$L(G) = 2 \quad \text{and} \quad \rho = 3. \tag{3.14}$$

Therefore relation (3.11) is linear in Y (see (3.12b)). (3.11) implies, if one compares the coefficients of Y, that

$$\frac{U}{N} \in \left\langle \frac{1}{G} \right\rangle + \Lambda^{(0)}$$

($\Lambda^{(0)} = k$, but for the sake of easier tracing we often prefer to write $\Lambda^{(0)}$). Thus, since $U/N \notin \Lambda^{(0)}$, there is a $\sigma \in k$ such that

$$\langle G \rangle = \left\langle \frac{N}{U + \sigma N} \right\rangle.$$

From (3.14) and Lemma 3.2 (ii) we get $\sigma = 0$. Together with (3.12b) and (3.14) this completes the proof of (ii) since $\langle N/U \rangle + \Lambda = \langle V^2/U \rangle + \Lambda$ (by the irreducibility of N). □

Remark. As exemplified in the above proof, the irreducibility of $N = N^{(2)}$ plays a decisive role. In order to preserve this property, our strategy therefore will never adjoin some intermediate result $\in k(U, V)$. For illustration, consider the computation sequence for division

$$\frac{U}{V - aU'}, \quad \frac{X}{V - aU'}, \quad \frac{U}{V - aU} \cdot bU, \quad \frac{X}{V - aU} \cdot bU, \quad \frac{Y + \dfrac{X}{V - aU} \cdot bU}{V + \dfrac{U}{V - aU} \cdot bU},$$

(3.15)

$$\frac{X}{V - aU} - \frac{Y + \dfrac{X}{V - aU} \cdot bU}{V + \dfrac{U}{V - aU} \cdot bU} \cdot \frac{U}{V - aU},$$

cooked up from Gaussian elimination applied to the linear system

$$\begin{pmatrix} V-aU & U \\ -bU & V \end{pmatrix}\begin{pmatrix} A \\ B \end{pmatrix}=\begin{pmatrix} X \\ Y \end{pmatrix}.$$

This shows that adjunction of $V-aU$, or of any $l\in\Lambda_{\{U,V\}}$ (by the action of Γ_D), will result in a collapse of the complexity ($\leqq 4$). So these adjunctions must be avoided.

From now on k denotes the base field of a quadratic extension $k\subset K=k(\omega)$, and N denotes the norm of the denominator element in (0.2).

PROPOSITION 3.16. *The computation hull of any optimal computation sequence* (p_1, p_2, p_3) *for* $(V-YU)/N$ *is determined as*

$$\langle p_1, p_2, p_3\rangle+\Lambda=\left\langle \frac{V}{U}, \frac{V^2}{U}, \frac{V-YU}{N}\right\rangle+\Lambda.$$

Proof. By Lemma 3.10 (i) and (ii) 1, the relation

$$\frac{V-YU}{N}\in\langle p_1, p_2, p_3\rangle+\Lambda \tag{3.17}$$

implies by Steinitz exchange

$$\langle p_1, p_2, p_3\rangle+\Lambda=\left\langle p_1, \frac{V^2}{U}, p_3\right\rangle+\Lambda=\left\langle p_1, \frac{V^2}{U}, \frac{V-YU}{N}\right\rangle+\Lambda$$

since $V^2/U\notin\langle p_1\rangle+\Lambda$ and $(V-YU)/N\notin\langle p_1, p_2\rangle+\Lambda$. It remains to show that p_1 satisfies Lemma 3.4 (ii) 1. By Lemma 3.10 (ii) 2, relation (3.17) is linear in Y, and a comparison of the Y free terms in (3.17) implies (by Lemma 3.10 (ii) 2)

$$\frac{V}{N}\in\langle p_1, p_2\rangle+\Lambda+\frac{U}{N}\langle p_1, p_2\rangle+\frac{U}{N}\Lambda=\left\langle p_1, \frac{V^2}{U}\right\rangle+\Lambda+\frac{U}{N}\left\langle p_1, \frac{V^2}{U}\right\rangle+\frac{U}{N}\Lambda. \tag{3.18}$$

p_1 cannot result from a multiplication, for (3.18) multiplied by NU would yield

$$UV\in(N\langle Up_1, V^2\rangle+NU\Lambda+U\langle Up_1, V^2\rangle+U^2\Lambda)^{(2)}=\langle U^2\rangle.$$

If $p_1=l_1/l_2$, where $l_1, l_2\in\Lambda^{(1)}_{\{U,V\}}$ as in Lemma 3.4 (ii) 1 or 2, (3.18) multiplied by NUl_2 gives

$$Ul_2V\in(N\langle Ul_1, l_2V^2\rangle+NUl_2\Lambda+U\langle Ul_1, l_2V^2\rangle+U^2l_2\Lambda)^{(3)}=\langle U^2l_1\rangle+U^2l_1\Lambda^{(0)},$$

whence $U|l_2$. So p_1 has the form Lemma 3.4 (ii) 1, since Lemma 3.4 (ii) 2 is excluded. □

Proof of Theorem 3.3. The upper bound follows from the computation sequence (0.4) with X replaced by one. It remains to prove the lower bound. First of all, we have

$$L\left(\frac{V-YU}{N}, \frac{Y(V-aU)+bU}{N}\right)\geqq L\left(\frac{-U}{N}, \frac{V-aU}{N}\right)\geqq 4$$

(adjoin Y to k, apply

$$\begin{pmatrix} 1 & 0 \\ 0 & \omega+Y \end{pmatrix}\in\Gamma_D,$$

use autarky (1.8) and Proposition 3.1). By the way of contradiction we assume $(p_1, \cdots, p_4)$ to be an optimal computation sequence for

$$\left\{\frac{V-YU}{N}, \frac{Y(V-aU)+bU}{N}\right\},$$

so

$$\left\langle \frac{V-YU}{N}, \frac{Y(V-aU)+bU}{N}\right\rangle\subset\langle p_1, p_2, p_3, p_4\rangle+\Lambda. \tag{3.19}$$

Counting dimensions this subspace and $\langle p_1, p_2, p_3\rangle + \Lambda$ intersect nontrivially. Because any nonzero element from the left-hand side is contained in the orbit $\Gamma_D((V - YU)/N)$ we may assume w.l.o.g.

$$\frac{V - YU}{N} \in \langle p_1, p_2, p_3\rangle + \Lambda = \left\langle \frac{V}{U}, \frac{V^2}{U}, \frac{V - YU}{N} \right\rangle + \Lambda, \tag{3.20}$$

the latter coming from Proposition 3.16. Thus by (3.19)

$$\frac{Y(V - aU) + bU}{N} \in \left\langle \frac{V}{U}, \frac{V^2}{U}, \frac{V - YU}{N}, p_4 \right\rangle + \Lambda, \tag{3.21}$$

and by Steinitz exchange (use the optimality of the computation sequence)

$$p_4 \in \left\langle \frac{V}{U}, \frac{V^2}{U}, \frac{V - YU}{N}, \frac{Y(V - aU) + bU}{N} \right\rangle + \Lambda. \tag{3.22}$$

Now, the latter and the coefficients of Y in (3.21) show two things, namely

$$p_4 \in k(U, V)[Y], \qquad \deg_Y p_4 = 1. \tag{3.23}$$

We distinguish the cases whether p_4 is the result of a multiplication or a division. In the multiplicative case one operand must be Y free (by (3.23)), so by (3.20) and (3.23) (use $\langle (V - YU)/N, Y\rangle \cap k(U, V) = \langle 0\rangle$)

$$p_4 = \left(\alpha_1 \frac{V}{U} + \alpha_2 \frac{V^2}{U} + \alpha_3 \frac{V - YU}{N} + \alpha_4 Y + l_1\right)\left(\beta_1 \frac{V}{U} + \beta_2 \frac{V^2}{U} + l_2\right) \tag{3.24}$$

where $\alpha_1, \cdots, \alpha_4, \beta_1, \beta_2 \in k$, and w.l.o.g. $l_1, l_2 \in \Lambda^{(1)}_{\{U,V\}}$. Comparing the Y free terms in (3.21) and multiplying them by NU^2 we get, in the case of (3.24),

$$U^3 \in \langle NUV, NUV^2, U^2V, (\alpha_1 NV + \alpha_2 NV^2 + \alpha_3 UV + NUl_1)(\beta_1 V + \beta_2 V^2 + Ul_2)\rangle + NU^2\Lambda$$

since $b \neq 0$. Comparing the homogeneous parts of degree three, we get the contradiction

$$U^3 \in \langle U^2V, UV^2\rangle.$$

It remains to discuss the case of a division, so

$$p_4 = \frac{\alpha_1 \frac{V}{U} + \alpha_2 \frac{V^2}{U} + \alpha_3 \frac{V - YU}{N} + \alpha_4 Y + l_1}{\beta_1 \frac{V}{U} + \beta_2 \frac{V^2}{U} + l_2} \tag{3.25}$$

for suitable $\alpha_1, \cdots, \alpha_4, \beta_1, \beta_2 \in k$ and $l_1, l_2 \in \Lambda_{\{U,V\}}$ (by (3.20) and (3.23)). Comparing the coefficients of Y in (3.22), we get from (3.25) and (3.23)

$$0 \neq \frac{-\alpha_3 \frac{U}{N} + \alpha_4}{\beta_1 \frac{V}{U} + \beta_2 \frac{V^2}{U} + l_2} \in \left\langle \frac{-U}{N}, \frac{V - aU}{N} \right\rangle + \Lambda^{(0)}. \tag{3.26}$$

Let $\beta_1 V + \beta_2 V^2 + Ul_2 = B$, for short. Multiplying (3.26) by BN, we get

$$0 \neq (-\alpha_3 U + \alpha_4 N)U \in \langle -BU, B(V - aU)\rangle + BN\Lambda^{(0)}. \tag{3.27}$$

Analogously, from (3.21),

$$B(V - aU) \in \langle -BU, (-\alpha_3 U + \alpha_4 N)U\rangle + BN\Lambda^{(0)}.$$

The latter shows that $U|B(V-aU+\sigma N)$ for some $\sigma \in k$; thus $U|B$ ($U \nmid (V-aU)$, $U \nmid (V-aU+\sigma N)^{(2)} = \sigma N$ if $\sigma \neq 0$, for N is irreducible), whence

$$\beta_1 = \beta_2 = 0 \quad \text{and then } \langle -\alpha_3 U + \alpha_4 N \rangle = \langle U \rangle = \langle l_2 \rangle$$

since $B = Ul_2$ must divide the nonzero left-hand side in (3.27). A comparison of the homogeneous parts of degree two in (3.27) now gives [$\langle B \rangle = \langle U^2 \rangle$],

$$U^2 \in \langle 0 \rangle,$$

being absurd. This finishes the proof of Theorem 3.3. □

We remark that Theorem 3.3 is included in the proof of the lower bound for the general division. We note the following important consequences:

CONCLUSION 3.28. *Let*

$$F = \left\{ \frac{XU - YU}{N}, \frac{Y(V-aU)+bXU}{N} \right\}.$$

$L(F) \leqq 5$ *implies*:

(i) $L(F) = 5$, *and adjunction of any* $l \in \Lambda \backslash \Lambda_{\{U,V\}}$ *to the ground field* k *is autarkical for* F.

(ii) *If* $(p_1, \cdots, p_5)$ *is an optimal computation sequence for* F *with* $p_1 \notin k(U, V)$, *then* $X/l \in \Gamma(p_1)$ *for some* $l \in \Lambda^{(1)}_{\{U,V\}}$. (*This could happen. Consider computation sequence* (3.15).)

Proof. (i) Let $l = l_1 + l_2$ be given with nonzero $l_1 \in \Lambda^{(1)}_{\{X,Y\}}$ and $l_2 \in \Lambda_{\{U,V\}}$. We show that F has complexity five in $(k(X, Y, U, V), \Omega^{k(l)}_{\{X,Y,U,V\}})$. We transform the new coefficient field via Γ such that $k(l) = k(X)$: By the action of Γ_N and Γ_D, l_1, l_2 may be changed such that

$$l = X + \beta U + \alpha \qquad (\alpha, \beta \in k)$$

(by $l_1 \neq 0$), and by the action of Γ_+ w.l.o.g. $\beta = 0$. Next replace Y, U, V by XY, XU, XV in order to make things X free. By autarky (1.7) we may restrict ourselves to $(k(X, Y, U, V), \Omega^{k(X)}_{\{Y,U,V\}})$, and by (1.8) to $(k(Y, U, V), \Omega^{k}_{\{Y,U,V\}})$, and then Theorem 3.3 yields the desired statement.

(ii) Here we prefer the substitution point of view (1.4). As follows from (i), p_1 cannot result from a multiplication or a division by some $l \notin \Lambda_{\{U,V\}}$. Thus, if $p_1 \notin k(U, V)$,

$$p_1 = \frac{l_1}{l}$$

for some $l_1 \in \Lambda \backslash \Lambda_{\{U,V\}}$, $l \in \Lambda_{\{U,V\}}$. By the actions of Γ_N, Γ_D, Γ_+ w.l.o.g.

$$p_1 = \frac{X - \alpha}{l} \qquad (\alpha \in k, l \in \Lambda_{\{U,V\}}).$$

It remains to show that

$$l^{(0)} = \alpha = 0. \tag{3.29}$$

Let t be an additional indeterminate. We consider all in $(k(X, Y, U, V, t), \Omega^{k(t)}_{\{X,Y,U,V\}})$. The substitution $X \mapsto t \cdot l + \alpha$, trivializing p_1, reduces the complexity of F. If (3.29) is not satisfied we get

$$t \cdot l^{(0)} + \alpha \neq 0,$$

by the transcendence of t over k. But then, analogously as above but using the symmetry group of the division in $K(t) \supset k(t)$ (!) we may transform the computation sequence such that

$$p_1 = \frac{(t \cdot l^{(0)} + \alpha)X + t \cdot l^{(1)} - \alpha}{l},$$

and thus the substitution $X \mapsto 1$ also reduces the complexity of F (observe that the transformed computation sequence is in the domain of $X \mapsto 1$). But by Theorem 3.3 this is impossible, and therefore (3.29) must hold. □

Remark. Following the lines of this paragraph, one can show that the division of elements from k by element from K admits only two different types of optimal computation seuqences $(p_1, \cdots, p_4)$: Either p_1, p_3 must result from divisions, and p_2, p_4 must result from multiplications, or all must result from divisions. After a suitable application of Γ_D the computation hulls are determined as

$$\left\langle \frac{V}{U}, \frac{V^2}{U}, \frac{YU}{N}, \frac{Y(V-aU)}{N} \right\rangle + \Lambda,$$

or as

$$\left\langle \frac{U}{\sigma U + V}, \frac{V^2}{U}, \frac{YU}{N}, \frac{Y(V-aU)}{N} \right\rangle + \Lambda \quad \text{for some } \sigma \in k.$$

We omit a proof.

4. Optimal computation sequences for $(XV - YU)/N$. In this section we prove some results on optimal computation sequences for $(XV - YU)/N$ which will be used to prove the lower bound of six for the general division (§ 5). Keeping in mind a computation sequence for $(XV - YU)/N$ to be a part of a computation sequence for division (0.3), we restrict ourselves to $\Gamma((XV - YU)/N) \cap \Gamma$ for simplifying computation sequences. For our purpose it will be sufficient to work with the part

$$\Sigma := \left\{ \begin{pmatrix} \zeta & \eta \\ 0 & \zeta \end{pmatrix} \in \Delta(2, K) : \zeta \in K^*, \eta \in K \right\} \subset \Gamma\left(\frac{XV - YU}{N}\right) \cap \Gamma,$$

stabilizing pointwise

$$\left\langle \frac{XY - YU}{N}, \frac{Y(V-aU) + bXU}{N} \right\rangle + \Lambda \Big/ \Lambda.$$

PROPOSITION 4.1.

(i) $L((XV - YU)/N) = 4$.

(ii) *After a suitable application of* Σ *an optimal computation sequence* $(p_1, \cdots, p_4)$ *for* $(XV - YU)/N$ *satisfies the following conditions, where* 1-4 *already are achieved by some diagonal*

$$\begin{pmatrix} \zeta & 0 \\ 0 & \zeta \end{pmatrix} \in \Sigma:$$

1) $\min\{\rho : p_\rho \notin k(U, V)\} \leqq 3$ *and* $p_1, p_2, p_3 \in k(X, U, V)$,

2) p_4 *must result from a division by a multiple of* N/U *so that* $\langle p_4 \rangle = \langle (Y - F)/(N/U) \rangle$ *for some* $F \in \langle p_1, p_2, p_3 \rangle + \Lambda_{\{X,U,V\}}$,

3) (p_1, p_2, p_3) *is a computation sequence for* V^2/U,

4) $X(V/U) \in \langle p_1, p_2, p_3 \rangle + \Lambda_{\{X,U,V\}} + (N/U)\langle p_1, p_2, p_3 \rangle + (N/U)\Lambda_{\{X,U,V\}}$,

5) *If* $p_1 \in k(U, V)$, *then the possible computation hulls are determined as*

$$\left\langle \frac{V}{U}, \frac{V^2}{U}, \left(X+\sigma\frac{N}{U}\right)\left(\frac{V}{U}+\tau\frac{N}{U}\right), \frac{XV-YU}{N} \right\rangle + \Lambda \quad \textit{for some } \sigma, \tau \in k,$$

or as

$$\left\langle \frac{U}{\sigma U+V}, \frac{V^2}{U}, X\frac{V}{U}, \frac{XV-YU}{N} \right\rangle + \Lambda \quad \textit{for some } \sigma \in k.$$

Proof. Let $(p_1, \cdots, p_r)$ be an optimal computation sequence for $(XY-YU)/N$, $r \leqq 4$ (by the initial part of (0.4)), and let $\nu = \min\{\rho : p_\rho \notin k(U, V)\}$. We apply some diagonal

$$\begin{pmatrix} \zeta & 0 \\ 0 & \zeta \end{pmatrix} \in \Sigma$$

to get p_ν, being computed by a multiplication or a division, into one of the following forms (making one operand Y free):

(4.2) $\quad p_\nu = (\alpha X + \beta Y - S)*/(X-T), \quad \alpha, \beta \in k \quad \text{and} \quad S, T \in \langle p_1, \cdots, p_{\nu-1}\rangle + \Lambda_{\{U,V\}},$

or

(4.3) $$p_\nu = \frac{X-T}{S}, \qquad S, T \in \langle p_1, \cdots, p_{\nu-1}\rangle + \Lambda_{\{U,V\}}.$$

Adjunction of the intermediate result

(4.4) $$t = \frac{X-T}{R}, \qquad R, T \in \langle p_1, \cdots, p_{\nu-1}\rangle + \Lambda_{\{U,V\}} \subset k(U, V)$$

to k where $R = 1$ in case of (4.2), or $R = S$ in case of (4.3), and deletion of the o-ary X produces a computation sequence of length $r-1 \leqq 3$ (over the extended coefficient field $k(t)$)

$$(p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_r) \in CS(k(X, Y, U, V), \Omega^{k(t)}_{\{Y,U,V\}})$$

for $$\frac{XV-YU}{N} = \frac{(tR+T)V}{N} - \frac{U}{N}Y.$$

By Lemma 3.10 (i) this computation sequence has length 3; thus $r = 4$, proving (i). Moreover, by Lemma 3.10 (ii) 2, its latest member is not Y free, so

(4.5) $$\nu \leqq 3;$$

therefore the latest member is p_4 [$r = 4$]. By Lemma 3.10 (ii) 1, we have the *optimal* computation sequence

(4.6) $\quad (p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_3) \in CS(k(X, Y, U, V), \Omega^{k(t)}_{\{Y,U,V\}}) \quad \text{for} \dfrac{V^2}{U};$

thus (by Lemma 1.9)

(4.7) $$p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_3 \in k(t)(U, V) = k(X, U, V).$$

Furthermore, Lemma 3.10 (ii) 2 gives information about p_4:

(4.8a) $\quad p_4 \in k(t, U, V)[Y] = k(X, U, V)[Y], \quad \deg_Y p_4 = 1, \quad$ and
p_4 must result from a division;

(4.8b) $$k(t) \cdot \langle p_4 \rangle = k(t) \cdot \left\langle \frac{Y-\tilde{F}}{(N/U)} \right\rangle \quad \text{for some } \tilde{F} \in k(t, U, V) = k(X, U, V).$$

(4.5) and (4.7) would prove Proposition 4.1 (ii) 1 if additionally

$$p_\nu \in k(X, U, V). \tag{4.9}$$

We postpone its proof and first complete the proofs of 4.1 (ii) 1-4 under this hypothesis.

We now return to the original computation theoretic structure. Assuming (4.9) the relation

$$\frac{XV - YU}{N} \in \langle p_1, p_2, p_3, p_4\rangle + \Lambda \tag{4.10}$$

is linear in Y (by (4.7), (4.8a)), p_4 having the form

$$p_4 = \frac{Y - F}{G} \quad \text{for some } F, G \in \langle p_1, p_2, p_3\rangle + \Lambda_{\{X,U,V\}} \tag{4.11}$$

(by (4.8a), shifting a nonzero scalar factor to the second operand G). We compare the coefficients of Y in (4.10) and get by (4.11) (and (4.7), (4.9))

$$\frac{U}{N} \in \left\langle \frac{1}{G} \right\rangle + \Lambda^{(0)}.$$

Whence, for some $\sigma \in k$

$$\left\langle \frac{N}{U + \sigma N} \right\rangle = \langle G\rangle \subset \langle p_1, p_2, p_3\rangle + \Lambda, \tag{4.12}$$

the latter inclusion coming from (4.11). Now, Lemma 3.2 (ii), adjunction of t (see (4.4)) to the ground field and deletion of the o-ary X, killing p_ν ($\nu \leqq 3$), show that $\sigma = 0$. Together with (4.8a), (4.11), (4.12) this shows 4.1 (ii) 2 and 3 (replacing N/U by the simpler V^2/U). Multiplying inclusion (4.10) by N/U and comparing the Y free terms, we obtain 4.1 (ii) 4 (by (4.11), (4.12) with $\sigma = 0$).

We still owe a proof of assumption (4.9), being clear in case of (4.3). By the way of contradiction we assume $\beta \neq 0$ in (4.2), w.l.o.g. $\beta = 1$. Then

$$p_\nu = (\alpha X + Y - S)A \quad \text{where } A = (X - T)^{\pm 1} \qquad (\text{with } T \in k(U, V)). \tag{4.13}$$

First of all, we may assume that

$$\nu = 3 \qquad (\text{if } p_\nu \text{ is not } Y \text{ free}).$$

This is achieved by showing that

$$(p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_3, p_\nu) \in CS(k(X, Y, U, V), \Omega^k_{\{X,Y,U,V\}}). \tag{4.14}$$

To see this, we work with a second change of structure, also killing p_ν. Adjoining

$$\tilde{t} = \begin{cases} (\alpha X + Y - S), & \text{in the multiplicative case of (4.2),} \\ (\alpha X + Y - S)(X - T)^{-1} & \text{in the division case of (4.2),} \end{cases}$$

to the ground field k and deleting the o-ary Y, we get the computation sequence (over the extended coefficient field $k(\tilde{t})$)

$$(p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_3) \in CS(k(X, Y, U, V), \Omega^{k(\tilde{t})}_{\{X,U,V\}}).$$

Now, $\tilde{t}$ (containing Y) is transcendental over $k(X, U, V)$, and the latter contains all the results, by the already proved (4.7). So by transcendence of $\tilde{t}$ over $k(X, U, V)$ (specialize it to conclude)

$$(p_1, \cdots, p_{\nu-1}, p_{\nu+1}, \cdots, p_3) \in CS(k(X, U, V), \Omega^k_{\{X,U,V\}}),$$

implying (4.14). Now, we return to the original structure and assume w.l.o.g. that $\nu = 3$. Then, by (4.8a) and (4.13) (with $\nu = 3$),

$$p_4 = \frac{\alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 + \alpha_4 Y + l_1}{\beta_1 p_1 + \beta_2 p_2 + l_2} \tag{4.15}$$

for suitable $\alpha_1, \cdots, \alpha_4$, β_1, $\beta_2 \in k$ and l_1, $l_2 \in \Lambda_{\{X,U,V\}}$ (use $\deg_Y p_4 = 1$ and $\langle p_3, Y\rangle \cap k(X, U, V) = \langle 0\rangle$). Over the extended coefficient field $k(t)$ we have the so far unused relation (4.8b). A comparison of the coefficients of Y in (4.8b) shows for the formally written denominator in (4.15)

$$k(t) \cdot \langle \beta_1 p_1 + \beta_2 p_2 + l_2\rangle = k(t) \cdot \left\langle \frac{N}{U} \right\rangle \tag{4.16}$$

where we remember that $t = X - T = A^{\pm 1}$ for some $T \in k(U, V)$ (by (4.2), (4.4) with $R = 1$, (4.13)). If this formally written denominator in (4.15) is contained in $k(U, V)$, (4.16) implies also proportionality over k, since t is transcendental over $k(U, V)$. Then w.l.o.g.

$$\beta_1 p_1 + \beta_2 p_2 + l_2 = \frac{N}{U}. \tag{4.17}$$

Otherwise this denominator is linear in X, i.e. $l_2 \notin \Lambda_{\{U,V\}}$ (by $\nu = 3$). By (4.16) there are P, $Q \in k[t]$ such that

$$P \cdot (\beta_1 p_1 + \beta_2 p_2 + l_2) = Q \cdot \frac{N}{U}.$$

Reading this as an identity in $k(U, V)[X]$ (by $t = X - T$) and comparing leading coefficients with respect to X, we get

$$\frac{N}{U} \in k,$$

that being absurd. Thus $l_2 \in \Lambda_{\{U,V\}}$ and (4.17) can be assumed. Now, a comparison of the coefficients of Y in (4.10) gives (by (4.13) with $\nu = 3$, (4.15), (4.17))

$$\frac{-U}{N} \in \left\langle A, \frac{\alpha_3 A + \alpha_4}{(N/U)} \right\rangle + \Lambda^{(0)}. \tag{4.18}$$

Observe that $A = t^{\pm 1}$, U, V are algebraically independent over k. So, looking at the coefficients of A, we conclude that A does *not* participate in any linear combination behind (4.18), and then also

$$\alpha_3 = 0. \tag{4.19}$$

Now, observe that the transition (4.10) $\to$ (4.18) is a k-linear mapping. Thus, since A cannot participate in (4.18), (4.10) reduces to

$$\frac{XV - YU}{N} \in \langle p_1, p_2, p_4\rangle + \Lambda,$$

and by (4.15) and (4.19) this relation comes from a computation sequence of length three, contradicting the already proved (i). This proves the proviso (4.9).

It remains to prove 4.1 (ii) 5. Skipping over the straightforward verification that the given spans in 4.1 (ii) 5 come from computation sequences, we show that these cover all possible computation hulls. By 4.1 (ii) 3, we have the computation sequence

(p_1, p_2, p_3) for V^2/U. The premise $p_1 \in k(U, V)$ will allow us to rearrange the computation sequence without changing the computation hull, so that already

$$\frac{V^2}{U} \in \langle p_1, p_2 \rangle + \Lambda_{\{U,V\}} \subset k(U, V). \tag{4.20}$$

Then Lemma 3.4 applies, and, as we will see the only possible beginnings of the computation sequence are as in Lemma 3.4 (ii) 1 or 2, leading to the first or to the second variant in Proposition 4.1 (ii) 5, where p_3 must result from a multiplication or a division, respectively.

We now recall that $\nu = \min\{\rho : p_\rho \notin k(U, V)\}$, so by our premise $p_1 \in k(U, V)$ and the already proved 4.1 (ii) 1, $2 \leqq \nu \leqq 3$. We adjoin the intermediate result t to k (see (4.4)) and delete the o-ary X. Then the *optimal* computation sequence (4.6) gives (over the extended coefficient field $k(t)$)

$$L\left(\frac{V^2}{U} \bmod p_1, p_2\right) = 0 \quad \text{in the case of } \nu = 3,$$

$$L\left(\frac{V^2}{U} \bmod p_1\right) = 1 \quad \text{in the case of } \nu = 2.$$

Returning to the original structure, these remain valid, by autarky (1.8). So we have (4.20) in case of $\nu = 3$. If $\nu = 2$ we choose some $p \in k(U, V)$ such that

$$(p_1, p, p_2) \in CS(k(X, U, V), \Omega^k_{\{X,U,V\}})$$

satisfying

$$\frac{V^2}{U} \in \langle p_1, p \rangle + \Lambda_{\{U,V\}} \subset k(U, V).$$

Then, by $V^2/U \notin \langle p_1 \rangle + \Lambda$,

$$\langle p_1, p, p_2 \rangle + \Lambda = \left\langle p_1, \frac{V^2}{U}, p_2 \right\rangle + \Lambda = \langle p_1, p_2, p_3 \rangle + \Lambda,$$

the latter equality coming from the already proved 4.1 (ii) 3, together with the fact that $V^2/U \notin \langle p_1, p_2 \rangle + \Lambda$ (by $p_2 = p_\nu \notin k(U, V)$ and Lemma 1.9). Therefore we may assume (4.20) and $\nu = 3$ for the rest of the proof.

In the sequel we discuss the possible beginnings of the computation sequence according to Lemma 3.4 (ii), thereby fixing the form of p_3. First of all, we look at the already proved relation 4.1 (ii) 4. The inclusion there cannot reduce to

$$X\frac{V}{U} \in \langle p_1, p_2 \rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\langle p_1, p_2 \rangle + \frac{N}{U}\Lambda_{\{X,U,V\}},$$

for a comparison of the coefficients of X would give

$$\frac{V}{U} \in \Lambda^{(0)} + \frac{N}{U}\Lambda^{(0)} \qquad \text{(by (4.20))},$$

that being impossible. Thus 4.1 (ii) 4 implies the important property of *linearity of* p_3 *with respect to* X,

$$p_3 \in k(U, V)[X], \qquad \deg_X p_3 = 1 \tag{4.21}$$

since $p_3 (= p_\nu)$ is not X free.

Now the discussion of the beginnings according to Lemma 3.4 (ii):
Case 1.

$$\langle p_1, p_2\rangle + \Lambda_{\{U,V\}} = \left\langle \frac{V}{U}, \frac{V^2}{U} \right\rangle + \Lambda_{\{U,V\}}.$$

Here, relation 4.1 (ii) 4 can be rewritten as

$$X\frac{V}{U} \in \left\langle \frac{V}{U}, \frac{V^2}{U}, p_3 \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\left\langle \frac{V}{U}, \frac{V^2}{U}, p_3 \right\rangle + \frac{N}{U}\Lambda_{\{X,U,V\}}. \tag{4.22}$$

Case (1*). If p_3 results from a multiplication, then by (4.21) one operand must be X free, so

$$p_3 = \left(\alpha_1\frac{V}{U} + \alpha_2\frac{V^2}{U} + X + l_1\right)\left(\beta_1\frac{V}{U} + \beta_2\frac{V^2}{U} + l_2\right)$$

where $\alpha_1, \alpha_2, \beta_1, \beta_2 \in k$, and w.l.o.g. $l_1, l_2 \in \Lambda^{(1)}_{\{U,V\}}$ (here and in the sequel the coefficient of X can be assumed to be one, by shifting a nonzero scalar factor to the second operand). Clearly, (in any case) $(XV - YU)/N$ can be exchanged for the last p_4 (by the optimality of the computation sequence), so it suffices to show that

$$\beta_1 \neq 0, \quad \beta_2\frac{V^2}{U} + l_2 \in \left\langle \frac{N}{U} \right\rangle, \quad \alpha_1 = 0 \tag{4.23}$$

(since then an additional use of $\Gamma_+ \subset \Sigma$ provides that

$$\alpha_2\frac{V^2}{U} + l_1 \in \left\langle \frac{N}{U} \right\rangle$$

is fulfilled too). To prove (4.23), we compare the coefficients of X in (4.22) and multiply them by U^2, so

$$UV \in U\langle \beta_1 V + \beta_2 V^2 + Ul_2\rangle + U^2\Lambda^{(0)} + N\langle \beta_1 V + \beta_2 V^2 + Ul_2\rangle + UN\Lambda^{(0)}. \tag{4.24}$$

The homogeneous parts of degree two tell us that $\beta_1 \neq 0$ (the first assertion in (4.23)), and that $U(\beta_1 V + \beta_2 V^2 + Ul_2)$ must participate in any linear combination behind (4.24). Thus there are $\sigma, \tau \in k$, $\sigma \neq 0$ such that

$$\mathrm{N}|(UV + \sigma U(\beta_1 V + \beta_2 V^2 + Ul_2) + \tau U^2)^{(3)} = \sigma U(\beta_2 V^2 + Ul_2),$$

proving the second part of (4.23). By $\beta_1 \neq 0$, $N(\beta_1 V + \beta_2 V^2 + Ul_2)$ *cannot* participate in any linear combination behind (4.24) since it is not divisible by U. Because the transition (4.22) → (4.24) is a k-linear mapping, a backward analysis shows that $(N/U)p_3$ does not participate in (4.22), but p_3 *must* participate (compare coefficients of X). Thus, after Steinitz exchange in (4.22)

$$p_3 \in \left\langle \frac{V}{U}, \frac{V^2}{U}, X\frac{V}{U} \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\left\langle \frac{V}{U}, \frac{V^2}{U} \right\rangle + \frac{N}{U}\Lambda_{\{X,U,V\}}.$$

Multiplying by U^2, we get for the X free terms

$$(\alpha_1 V + \alpha_2 V^2 + Ul_1)(\beta_1 V + \beta_2 V^2 + Ul_2) \in \langle UV, UV^2\rangle + U^2\Lambda_{\{U,V\}} + N\langle U, V^2\rangle + UN\Lambda_{\{U,V\}}.$$

Comparing the homogeneous parts of degree two, we get $\alpha_1\beta_1 = 0$, thus $\alpha_1 = 0$, proving the remaining part of (4.23).

Case (1/). p_3 cannot result from a division. Assume otherwise. By (4.21)

$$p_3 = \frac{\alpha_1 \frac{V}{U} + \alpha_2 \frac{V^2}{U} + X + l_1}{\beta_1 + \frac{V}{U} + \beta_2 \frac{V^2}{U} + l_2} \tag{4.25}$$

for suitable α_1, α_2, $\beta_1\beta_2 \in k$ and l_1, $l_2 \in \Lambda_{\{U,V\}}$. Let $\beta_1 V + \beta_2 V^2 + Ul_2 = B$, for short. We compare the coefficients of X in (4.22)

$$\frac{V}{U} \in \left\langle \frac{U}{B} \right\rangle + \Lambda^{(0)} + \frac{N}{U}\left\langle \frac{U}{B} \right\rangle + \frac{N}{U}\Lambda^{(0)}.$$

Multiplying by UB, we get

$$BV \in \langle U^2 \rangle + UB\Lambda^{(0)} + N\langle U \rangle + BN\Lambda^{(0)}, \tag{4.26}$$

and therefore $U|(BV + \sigma BN) = B(V + \sigma N)$ for some $\sigma \in k$. Thus, $U|B$ (use the irreducibility of N), i.e.

$$\beta_1 = \beta_2 = 0, \quad \text{and then } l_2 \notin k,$$

the latter coming from the optimality of the computation sequence (see (4.25)). Comparing the homogeneous parts of degree three in (4.26), we get

$$0 \neq Ul_2^{(1)} V \in \langle U^2 l_2^{(1)} \rangle + \langle NU \rangle,$$

that being impossible (discuss participation of NU, and use the irreducibility of N). So p_3 does not result from a division.

Case 2.

$$\langle p_1, p_2 \rangle + \Lambda_{\{U,V\}} = \left\langle \frac{U}{l}, \frac{V^2}{U} \right\rangle + \Lambda_{\{U,V\}}$$

where $l = \sigma U + V$ for some $\sigma \in k$. Here, Relation 4.1 (ii) 4 can be rewritten as

$$X\frac{V}{U} \in \left\langle \frac{U}{l}, \frac{V^2}{U}, p_3 \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\left\langle \frac{U}{l}, \frac{V^2}{U}, p_3 \right\rangle + \frac{N}{U}\Lambda_{\{X,U,V\}}. \tag{4.27}$$

Case (2*). If p_3 is the result of a multiplication, then by (4.21) one operand is X free, so

$$p_3 = \left(\alpha_1 \frac{U}{l} + \alpha_2 \frac{V^2}{U} + X + l_1\right)\left(\beta_1 \frac{U}{l} + \beta_2 \frac{V^2}{U} + l_2\right)$$

for suitable α_1, α_2, β_1, $\beta_2 \in k$, and w.l.o.g. l_1, $l_2 \in \Lambda^{(1)}_{\{U,V\}}$. Comparing the coefficients of X in (4.27), we get

$$\frac{V}{U} \in \left\langle \beta_1 \frac{U}{l} + \beta_2 \frac{V^2}{U} + l_2 \right\rangle + \Lambda^{(0)} + \frac{N}{U}\left\langle \beta_1 \frac{U}{l} + \beta_2 \frac{V^2}{U} + l_2 \right\rangle + \frac{N}{U}\Lambda^{(0)},$$

and multiplication by $U^2 l$ yields

$$UlV \in \langle \beta_1 U^3 + \beta_2 UlV^2 + U^2 l l_2 \rangle + U^2 l\Lambda^{(0)} + N\langle \beta_1 U^2 + \beta_2 lV^2 + Ull_2 \rangle + UlN\Lambda^{(0)}.$$

The homogeneous parts of degree three tell us that

$$UlV \in \langle \beta_1 U^3 \rangle + U^2 l\Lambda^{(0)}.$$

Thus $U^2|UlV$, thus $U|l = \sigma U + V$, a contradiction.

Case (2/). So p_3 must result from a division. Then, by (4.21),

$$p_3 = \frac{\alpha_1 \frac{U}{l} + \alpha_2 \frac{V^2}{U} + X + l_1}{\beta_1 \frac{U}{l} + \beta_2 \frac{V^2}{U} + l_2} \tag{4.28}$$

$$= \frac{\alpha_1 U^2 + \alpha_2 lV^2 + lUX + lUl_1}{\beta_1 U^2 + \beta_2 lV^2 + lUl_2} = \frac{A}{B} \quad \text{(for short)}$$

where $\alpha_1, \alpha_2, \beta_1, \beta_2 \in k$ and $l_1, l_2 \in \Lambda_{\{U,V\}}$. It suffices to show that

$$\beta_2 = l_2 = 0, \quad \beta_1 \neq 0, \quad \alpha_2 = l_1^{(0)} = 0 \tag{4.29}$$

(since then, using $\Gamma_+ \subset \Sigma$, one can achieve that $l_1 = 0$, and then $\langle p_3 \rangle + \Lambda = \langle X(V/U) \rangle + \Lambda$). To prove (4.29), we compare the coefficients of X in (4.27)

$$\frac{V}{U} \in \left\langle \frac{lU}{B} \right\rangle + \Lambda^{(0)} + \frac{N}{U} \left\langle \frac{lU}{B} \right\rangle + \frac{N}{U} \Lambda^{(0)}.$$

Multiplication by UB yields

$$BV \in U\langle lU \rangle + UB\Lambda^{(0)} + N\langle lU \rangle + BN\Lambda^{(0)}; \tag{4.30}$$

thus for a suitable $\rho \in k$,

$$U | (BV + \rho BN) = B(V + \rho N).$$

Therefore $U|B$ (use the irreducibility of N), i.e.

$$\beta_2 = 0 \tag{4.31}$$

since $U \nmid l = \sigma U + V$. Relation (4.30) also implies that for suitable $\rho, \tau \in k$

$$N | (BV + \rho lU^2 + \tau UB)^{(4)} = lUl_2^{(1)}(V + \tau U) \quad \text{(by (4.31))};$$

thus by the irreducibility of N

$$l_2^{(1)} = 0. \tag{4.32}$$

Next we consider the homogeneous parts of degree three in (4.30)

$$(\beta_1 U^2 + lUl_2^{(0)}) V \in \langle lU^2 \rangle + U(\beta_1 U^2 + lUl_2^{(0)}) \Lambda^{(0)}.$$

Since U^2 must divide the left-hand side of this inclusion, we get $l_2^{(0)} = 0$ ($l = \sigma U + V$), and together with (4.32) and (4.31)

$$l_2 = 0, \quad B = \beta_1 U^2 \quad \text{and then } \beta_1 \neq 0 \tag{4.33}$$

(see (4.28)), proving the first two assertions of (4.29). (4.33) shows that *NlU cannot* participate in any linear combination behind (4.30) since it is not divisible by $U^2 (l = \sigma U + V)$. Since the transition (4.27) → (4.30) is a k-linear mapping, a backward analysis shows that $(N/U)p_3$ does not participate in (4.27), but p_3 *must* participate (compare coefficients of X). Thus, after Steinitz-exchange in (4.27) we get

$$p_3 \in \left\langle X \frac{V}{U} \right\rangle + \left\langle \frac{U}{l}, \frac{V^2}{U} \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U} \left\langle \frac{U}{l}, \frac{V^2}{U} \right\rangle + \frac{N}{U} \Lambda_{\{X,U,V\}}.$$

Multiplication by $U^2 l$ yields (by (4.28), (4.33))

$$lA \in \langle UlXV \rangle + \langle U^3, UlV^2 \rangle + U^2 l \Lambda_{\{X,U,V\}} + N\langle U^2, lV^2 \rangle + UlN\Lambda_{\{X,U,V\}}. \tag{4.34}$$

This implies that

$$U|(lA+\tau NlV^2) \quad \text{for some } \tau \in k.$$

The homogeneous part of degree five tells us that $\tau = 0$ (by $\deg A \leqq 3$, $l = \sigma U + V$). Thus $U|A$, thus

$$\alpha_2 = 0. \tag{4.35}$$

(by $l = \sigma U + V$). Then the homogeneous parts of degree three of the X free terms in (4.34) give

$$l(\alpha_1 U^2 + lUl_1^{(0)}) \in \langle U^3 \rangle + U^2 l\Lambda^{(0)},$$

and therefore

$$l_1^{(0)} = 0$$

since the left-hand side must be divisible by U^2 (by $l = \sigma U + V$). Together with (4.35) this proves the remaining part of (4.29).

We now show that a multiplicative beginning of the computation sequence according to Lemma 3.4 (ii) 3 is impossible, completing the proof of Proposition 4.1 (ii) 5 by the just-treated Cases 1 and 2.

Case 3.

$$\langle p_1, p_2 \rangle + \Lambda_{\{U,V\}} = \left\langle ll_0, \frac{V^2}{U} \right\rangle + \Lambda_{\{U,V\}}$$

where $l = \sigma U + V$, $l_0 = \tau U + V$ for some $\sigma, \tau \in k$. Here, relation 4.1 (ii) 4 can be rewritten as

$$X\frac{V}{U} \in \left\langle ll_0, \frac{V^2}{U}, p_3 \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\left\langle ll_0, \frac{V^2}{U}, p_3 \right\rangle + \frac{N}{U}\Lambda_{\{X,U,V\}}. \tag{4.36}$$

Case (3∗). If p_3 results from a multiplication, then by (4.21)

$$p_3 = \left(\alpha_1 ll_0 + \alpha_2 \frac{V^2}{U} + X + l_1\right)\left(\beta_1 ll_0 + \beta_2 \frac{V^2}{U} + l_2\right)$$

for suitable $\alpha_1, \alpha_2, \beta_1, \beta_2 \in k$, and w.l.o.g. $l_1, l_2 \in \Lambda^{(1)}_{\{U,V\}}$ (one operand must be X free). Comparing the coefficients of X in (4.36), we get, after multiplying by U^2,

$$UV \in U\langle \beta_1 Ull_0 + \beta_2 V^2 + Ul_2 \rangle + U^2\Lambda^{(0)} + N\langle \beta_1 Ull_0 + \beta_2 V^2 + Ul_2 \rangle + UN\Lambda^{(0)}.$$

This implies for the homogeneous parts of degree two

$$UV \in U^2\Lambda^{(0)},$$

that being absurd. So p_3 is not the result of a multiplication assuming a multiplicative beginning of the computation sequence.

Case (3/). In case of a division p_3 has the form

$$p_3 = \frac{\alpha_1 ll_0 + \alpha_2 \dfrac{V^2}{U} + X + l_1}{\beta_1 ll_0 + \beta_2 \dfrac{V^2}{U} + l_2} \tag{4.37}$$

for suitable $\alpha_1, \alpha_2, \beta_1, \beta_2 \in k$ and $l_1, l_2 \in \Lambda_{\{U,V\}}$ (by (4.21)). Let $\beta_1 Ull_0 + \beta_2 V^2 + Ul_2 = B$, for short. Comparing the coefficients of X in (4.36), we get

$$\frac{V}{U} \in \left\langle \frac{U}{B} \right\rangle + \Lambda^{(0)} + \frac{N}{U}\left\langle \frac{U}{B} \right\rangle + \frac{N}{U}\Lambda^{(0)};$$

thus, after multiplying by UB,

$$BV \in \langle U^2 \rangle + UB\Lambda^{(0)} + N\langle U \rangle + BN\Lambda^{(0)}. \tag{4.38}$$

Whence there is a $\rho \in k$ such that $U|(BV + \rho BN) = B(V + \rho N)$, thus $U|B$ (use the irreducibility of N), i.e.

$$\beta_2 = 0. \tag{4.39}$$

Relation (4.38) also implies that, for suitable $\delta, \varepsilon \in k$,

$$N|(BV + \delta U^2 + \varepsilon UB), \tag{4.40}$$

thus $N|(BV + \delta U^2 + \varepsilon UB)^{(4)} = \beta_1 Ull_0(V + \varepsilon U)$, whence

$$\beta_1 = 0 \tag{4.41}$$

(by the irreducibility of N). Thus, together with (4.39), $B = Ul_2$. But then, (4.40) gives for the irreducible N

$$N|(BV + \delta U^2 + \varepsilon UB)^{(3)} = Ul_2^{(1)}(V + \varepsilon U).$$

Whence $l_2^{(1)} = 0$, and p_3 is the result of a division by the constant $l_2^{(0)}$ (by (4.37), (4.39), (4.41)), contradicting the optimality of the computation sequence. So a multiplicative beginning of the computation sequence is impossible.

This finishes the proof of Proposition 4.1. □

5. The general division. In this section we prove the main result of this paper:

THEOREM 5.1. *For any quadratic field extension $k \subset K$ the complexity of the division of two elements from K over k is six.*

An inspection of computation sequence (0.4), for $a = 0$ and $b = 1$, gives us immediately the following consequence, even if scalar multiplication are counted as well.

COROLLARY. *The total number of multiplications or divisions necessary and sufficient to compute the quotient of two complex numbers over the reals is six.*

Remark. For comparison and completeness we note the following fact on division of polynomials modulo a reducible quadratic polynomial: The complexity of the division in the k-algebra $Q = k[T]/((T - \omega_1)(T - \omega_2))$ is two or three, according to whether $\omega_1 \neq \omega_2$ or not. For, by the Chinese Remainder Theorem, $Q \cong k \times k$ or $Q \cong k[T]/(T^2)$. In the first case the statement follows immediately. In the second case we have

$$(XT + Y)(UT + V)^{-1} \equiv \left(\frac{X - (Y/V) \cdot U}{V}\right)T + \frac{Y}{V} \bmod T^2,$$

implying the upper bound. For the lower bound one shows $L((XV - YU)/V^2 \bmod Y/V) > 1$. This is left to the reader as an exercise.

As suggested by Proposition 4.1 (ii) 5 the proof of Theorem 5.1 starts by showing the following

LEMMA 5.2. (i) *For any $\sigma, \tau \in k$*

$$L\left(\frac{Y(V - aU) + bXU}{N} \bmod \frac{V}{U}, \frac{V^2}{U}, \left(X + \sigma\frac{N}{U}\right)\left(\frac{V}{U} + \tau\frac{N}{U}\right), \frac{XV - YU}{N}\right) = 2.$$

(ii) *For any* $\sigma \in k$

$$L\left(\frac{Y(V-aU)+bXU}{N} \bmod \frac{U}{\sigma U+V}, \frac{V^2}{U}, X\frac{V}{U}, \frac{XV-YU}{N}\right)=2.$$

Proof. The upper bounds being clear (compare (0.4), (0.5)), we prove the lower bounds. Let

$$\tilde{\Lambda}=\begin{cases}\left\langle \frac{V}{U}, \frac{V^2}{U}, \left(X+\sigma\frac{N}{U}\right)\left(\frac{V}{U}+\tau\frac{N}{U}\right), \frac{XV-YU}{N}\right\rangle+\Lambda & \text{in the case (i),}\\ \left\langle \frac{U}{\sigma U+V}, \frac{V^2}{U}, X\frac{V}{U}, \frac{XV-YU}{N}\right\rangle+\Lambda & \text{in the case (ii).}\end{cases} \tag{5.3}$$

Contrary to the assertion, we assume in both cases that there is a product or a quotient p of two elements from $\tilde{\Lambda}$ such that

$$\frac{Y(V-aU)+bXU}{N} \in \langle p\rangle+\tilde{\Lambda}. \tag{5.4}$$

By Conclusion 3.28 (i) p is not superfluous (and also $p \notin \tilde{\Lambda}$), so by Steinitz exchange

$$p \in \left\langle \frac{Y(V-aU)+bXU}{N}\right\rangle+\tilde{\Lambda}.$$

These imply in both cases *linearity of p with respect to X, Y,*

$$p \in k(U, V)[X, Y], \quad \deg_{X,Y} p=1. \tag{5.5}$$

In the sequel we utilize this special form of p. Preparatively we remark first that

$$\tilde{\Lambda} \cap k(U, V)=\begin{cases}\left\langle \frac{V}{U}, \frac{V^2}{U}\right\rangle+\Lambda_{\{U,V\}} & \text{in the case (i),}\\ \left\langle \frac{U}{\sigma U+V}, \frac{V^2}{U}\right\rangle+\Lambda_{\{U,V\}} & \text{in the case (ii)}\end{cases} \tag{5.6}$$

(compare coefficients of X and Y).

Case (i*). If p results from a multiplication, then by (5.5) one operand must be X and Y free, so by (5.3) and (5.6)

$$p=\left(\alpha_1\frac{V}{U}+\alpha_2\frac{V^2}{U}+\alpha_3\left(X+\sigma\frac{N}{U}\right)\left(\frac{V}{U}+\tau\frac{N}{U}\right)+\alpha_4\frac{XV-YU}{N}+\alpha_5 X+\alpha_6 Y+l_1\right) \times\left(\beta_1\frac{V}{U}+\beta_2\frac{V^2}{U}+l_2\right) \tag{5.7}$$

where $\alpha_1, \cdots, \alpha_6, \beta_1, \beta_2 \in k$, and w.l.o.g. $l_1, l_2 \in \Lambda^{(1)}_{\{U,V\}}$. Comparing the coefficients of X in (5.4), we get from (5.7) and (5.3)

$$\frac{U}{N} \in \left\langle\left(\alpha_3\left(\frac{V}{U}+\tau\frac{N}{U}\right)+\alpha_4\frac{V}{N}+\alpha_5\right)\left(\beta_1\frac{V}{U}+\beta_2\frac{V^2}{U}+l_2\right)\right\rangle+\left\langle\left(\frac{V}{U}+\tau\frac{N}{U}\right), \frac{V}{N}\right\rangle+\Lambda^{(0)}$$

since $b \neq 0$. Multiplication by U^2N yields

$$U^3 \in \langle \alpha_3 N(V+\tau N)+\alpha_4 UV+\alpha_5 UN)(\beta_1 V+\beta_2 V^2+Ul_2)\rangle+\langle UN(V+\tau N), U^2V\rangle + U^2N\Lambda^{(0)}.$$

The homogeneous parts of degree three tell us that

$$U^3 \in \langle \alpha_4\beta_1 UV^2\rangle + \langle U^2V\rangle,$$

that being absurd.

Case (i/). If p results from a division, then by (5.5) the second operand must be X and Y free, so by (5.3) and (5.6)

$$p = \frac{\alpha_1\frac{V}{U} + \alpha_2\frac{V^2}{U} + \alpha_3\left(X + \sigma\frac{N}{U}\right)\left(\frac{V}{U} + \tau\frac{N}{U}\right) + \alpha_4\frac{XV - YU}{N} + \alpha_5 X + \alpha_6 Y + l_1}{\beta_1\frac{V}{U} + \beta_2\frac{V^2}{U} + l_2} \tag{5.8}$$

where $\alpha_1, \cdots, \alpha_6$, β_1, $\beta_2 \in k$ and l_1, $l_2 \in \Lambda_{\{U,V\}}$. Let $A = -\alpha_4 U + \alpha_6 N$, and $B = \beta_1 V + \beta_2 V^2 + Ul_2$, for short. We compare the coefficients of Y in (5.4) and get from (5.8) and (5.3)

$$\frac{V - aU}{N} \in \left\langle \frac{AU}{BN}\right\rangle + \left\langle \frac{U}{N}\right\rangle + \Lambda^{(0)}.$$

Multiplication by BN yields

$$B(V - aU) \in \langle AU\rangle + \langle BU\rangle + BN\Lambda^{(0)}. \tag{5.9}$$

Thus, for a suitable $\rho \in k$, $U|B(V - aU + \rho N)$, and thus $U|B$ (use the irreducibility of N), whence

$$\beta_1 = \beta_2 = 0, \quad B = Ul_2 \quad \text{and} \quad l_2 \notin k, \tag{5.10}$$

the latter coming from $p \notin \tilde{\Lambda}$ (see (5.8)). By (5.9) $A \neq 0$, and AU must participate in any linear combination behind (5.9). Thus $B|AU$, thus $l_2|A = -\alpha_4 U + \alpha_6 N$ (by (5.10)), implying $0 \neq l_2^{(1)}|\alpha_6 N$ (by (5.10)). Therefore $\alpha_6 = 0$ and $l_2| - \alpha_4 U = A \neq 0$, and thus

$$\langle l_2\rangle = \langle A\rangle = \langle U\rangle.$$

But then a comparison of the homogeneous parts of degree three in (5.9) yields

$$U^2(V - aU) \in \langle U^3\rangle \qquad (\text{by } B = Ul_2),$$

a contradiction.

Case (ii*). If p results from a multiplication, then by (5.5) one operand must be X and Y free, so by (5.3) and (5.6)

$$p = \left(\alpha_1\frac{U}{l} + \alpha_2\frac{V^2}{U} + \alpha_3 X\frac{V}{U} + \alpha_4\frac{XV - YU}{N} + \alpha_5 X + \alpha_6 Y + l_1\right)\left(\beta_1\frac{U}{l} + \beta_2\frac{V^2}{U} + l_2\right) \tag{5.11}$$

where $\alpha_1, \cdots, \alpha_6$, β_1, $\beta_2 \in k$, $l = \sigma U + V$, and w.l.o.g. l_1, $l_2 \in \Lambda^{\{1\}}_{\{\{U,V\}}}$. We compare the coefficients of Y in (5.4) and get from (5.11) and (5.3)

$$\frac{V - aU}{N} \in \left\langle \left(-\alpha_4\frac{U}{N} + \alpha_6\right)\left(\beta_1\frac{U}{l} + \beta_2\frac{V^2}{U} + l_2\right)\right\rangle + \left\langle \frac{U}{N}\right\rangle + \Lambda^{(0)}.$$

Multiplication by NUl yields

$$Ul(V - aU) \in \langle(-\alpha_4 U + \alpha_6 N)(\beta_1 U^2 + \beta_2 lV^2 + Ull_2)\rangle + \langle lU^2\rangle + NUl\Lambda^{(0)},$$

and the homogeneous parts of degree three tell us that

$$Ul(V - aU) \in \langle -\alpha_4\beta_1 U^3\rangle + \langle lU^2\rangle;$$

thus $U|l = \sigma U + V$, a contradiction.

Case (ii/). If p results from a division, then by (5.5) the second operand must be X and Y free, so by (5.3) and (5.6)

$$p=\frac{\alpha_1\frac{U}{l}+\alpha_2\frac{V^2}{U}+\alpha_3X\frac{V}{U}+\alpha_4\frac{XV-YU}{N}+\alpha_5X+\alpha_6Y+l_1}{\beta_1\frac{U}{l}+\beta_2\frac{V^2}{U}+l_2} \tag{5.12}$$

where $\alpha_1,\cdots,\alpha_6,\ \beta_1,\ \beta_2\in k,\ l=\sigma U+V$ and $l_1,\ l_2\in\Lambda_{\{U,V\}}$. Let $A=-\alpha_4U+\alpha_6N$, and $B=\beta_1U^2+\beta_2lV^2+Ull_2$, for short. Comparing the coefficients of Y in (5.4), we get from (5.12) and (5.3)

$$\frac{V-aU}{N}\in\left\langle\frac{AUl}{BN}\right\rangle+\left\langle\frac{U}{N}\right\rangle+\Lambda^{(0)}.$$

Multiplication by BN yields

$$B(V-aU)\in\langle AUl\rangle+\langle BU\rangle+BN\Lambda^{(0)}; \tag{5.13}$$

thus, for a suitable $\rho\in k$, $U|B(V-aU+\rho N)$, and thus $U|B$ (use the irreducibility of N), whence

$$\beta_2=0. \tag{5.14}$$

By (5.13) $A\neq 0$, and AUl must participate in any linear combination behind (5.13). Thus $B|AUl$. If $l=\sigma U+V\not\prec B$ $(\beta_1\neq 0)$ this implies that $(\beta_1U+ll_2)|(-\alpha_4U+\alpha_6N)=A\neq 0$, whence $\alpha_6=0$, and we get

$$l_2=0,\qquad \beta_1\neq 0. \tag{5.15}$$

Otherwise by (5.14)

$$\beta_1=\beta_2=0,\quad B=Ull_2\quad\text{and}\quad l_2\notin k, \tag{5.16}$$

the latter coming from $p\notin\tilde{\Lambda}$ (see (5.12)). Since $A\neq 0$, and AUl must participate in (5.13), we get $Ull_2|AUl\neq 0$; thus $l_2|A$, whence $\alpha_6=0$, and we get

$$\langle l_2\rangle=\langle A\rangle=\langle U\rangle\qquad\text{(by (5.16))}.$$

But then, together with (5.16), a comparison of the homogeneous parts of degree four in (5.13) would give

$$U^2l(V-aU)\in\langle U^3l\rangle,$$

that being impossible. Therefore (5.15) must hold. Now we look at the coefficients of X in (5.4). From (5.12), (5.3), (5.14), (5.15) we get

$$\frac{U}{N}\in\left\langle\alpha_3\frac{lV}{U^2}+\alpha_4\frac{lV}{NU}+\alpha_5\frac{l}{U}\right\rangle+\left\langle\frac{V}{U},\frac{V}{N}\right\rangle+\Lambda^{(0)}.$$

Multiplication by NU^2 yields

$$U^3\in\langle\alpha_3NlV+\alpha_4UlV+\alpha_5NUl\rangle+\langle NUV,\ U^2V\rangle+NU^2\Lambda^{(0)}$$

and a comparison of the homogeneous parts of degree three yields the absurdity

$$U^3\in\langle\alpha_4UlV\rangle+\langle U^2V\rangle.$$

This finishes the proof of Lemma 5.2. □

Proof of Theorem 5.1. By the way of contradiction we assume $(p_1, \cdots, p_4, p_5)$ to be an optimal computation sequence for $\{(XV-YU)/N, (Y(V-aU)+bXU/N\}$, so

$$\left\langle \frac{XV-YU}{N}, \frac{Y(V-aU)+bXU}{N} \right\rangle \subset \langle p_1, \cdots, p_4, p_5\rangle + \Lambda. \tag{5.17}$$

Case $(p_1 \in k(U, V))$. By a dimension argument, the left-hand side in (5.17) and $\langle p_1, \cdots, p_4\rangle + \Lambda$ intersect nontrivially, and by the action of Γ_D (preserving $p_1 \in k(U, V)$) w.l.o.g.

$$\frac{XV-YU}{N} \in \langle p_1, \cdots, p_4\rangle + \Lambda, \tag{5.18a}$$

(5.17) becoming tantamount to saying

$$L\left(\frac{Y(V-aU)+bXU}{N} \bmod p_1, \cdots, p_4\right) \leqq 1. \tag{5.18b}$$

Now, Proposition 4.1 (ii) 5 applies: By means of Σ (preserving (5.18b) and $p_1 \in k(U, V)$) the right-hand side in (5.18a) can be transformed into one of the alternatives given in Proposition 4.1 (ii) 5. But then (5.18b) contradicts Lemma 5.2.

Case $(p_1 \notin k(U, V))$. By Conclusion 3.28 (ii), p_1 can be transformed via Γ into the form

$$p_1 = \frac{X}{l} \quad \text{for some } l \in \Lambda^{(1)}_{\{U,V\}}. \tag{5.19}$$

In the following we reduce this case to the one just treated by showing that (5.19) implies that

$$p_2 \in k(U, V) \quad \text{and} \quad L(p_2) \leqq 1, \tag{5.20}$$

which allows interchanging p_1 and p_2.

Now as above, the left-hand side in (5.17) and $\langle p_1, \cdots, p_4\rangle + \Lambda$ intersect nontrivially, and by the action of Γ_D (preserving the form (5.19) of p_1) w.l.o.g.

$$\frac{XV-YU}{N} \in \langle p_1, \cdots, p_4\rangle + \Lambda.$$

Then, applying some *diagonal* element from Σ, $(p_1, \cdots, p_4)$ fulfills the statement 1-4 of Proposition 4.1 (ii), in particular (by Proposition 4.1 (ii) 1) $p_1 \in k(X, U, V)$. So this transform preserves the form (5.19) of p_1, and by Proposition 4.1 (ii) 3

$$\frac{V^2}{U} \in \left\langle \frac{X}{l}, p_2, p_3 \right\rangle + \Lambda_{\{X,U,V\}}.$$

Since $p_1 = X/l \notin k(U, V)$, Lemmas 1.9 and 3.4 (i) imply that $V^2/U \notin \langle p_1, p_2\rangle + \Lambda$. Therefore by Steinitz exchange

$$\langle p_1, p_2, p_3\rangle + \Lambda_{\{X,U,V\}} = \left\langle \frac{X}{l}, p_2, \frac{V^2}{U} \right\rangle + \Lambda_{\{X,U,V\}},$$

and combined with Proposition 4.1 (ii) 4

$$X\frac{V}{U} \in \left\langle \frac{X}{l}, p_2, \frac{V^2}{U} \right\rangle + \Lambda_{\{X,U,V\}} + \frac{N}{U}\left\langle \frac{X}{l}, p_2, \frac{V^2}{U} \right\rangle + \frac{N}{U}\Lambda_{\{X,U,V\}}. \tag{5.21}$$

This relation cannot reduce to

$$X\frac{V}{U}\in\left\langle\frac{X}{l},\frac{V^2}{U}\right\rangle+\Lambda_{\{X,U,V\}}+\frac{N}{U}\left\langle\frac{X}{l},\frac{V^2}{U}\right\rangle+\frac{N}{U}\Lambda_{\{X,U,V\}}, \tag{5.22}$$

for a comparison of the coefficients of X would yield

$$\frac{V}{U}\in\left\langle\frac{1}{l}\right\rangle+\Lambda^{(0)}+\frac{N}{U}\left\langle\frac{1}{l}\right\rangle+\frac{N}{U}\Lambda^{(0)}.$$

Multiplying this by Ul we would get

$$lV\in(\langle U\rangle+Ul\Lambda^{(0)}+\langle N\rangle+lN\Lambda^{(0)})^{(2)}=Ul\Lambda^{(0)}+\langle N\rangle,$$

that being absurd (discuss participation of N, and use the irreducibility of N). Thus, since (5.22) is impossible, (5.21) shows that

$$p_2\in k(U,V)[X],\qquad \deg_X p_2\leqq 1. \tag{5.23}$$

Some further coarse information about p_2 can be had if we adjoin $t=X/l$ to the ground field and delete the o-ary X. Then we have the optimal computation sequence

$$(p_2,p_3)\in CS(k(X,U,V),\Omega^{k(t)}_{\{U,V\}})\quad \text{for}\ \frac{V^2}{U}\qquad \left(t=\frac{X}{l}\right),$$

and Lemma 3.4 (ii) gives us the following alternatives:

$$k(t)\cdot(\langle p_2\rangle+\Lambda)=k(t)\cdot\left(\left\langle\frac{V}{U}\right\rangle+\Lambda\right),\quad \text{or} \tag{5.24a}$$

$$k(t)\cdot(\langle p_2\rangle+\Lambda)=k(t)\cdot\left(\left\langle\frac{U}{\sigma U+V}\right\rangle+\Lambda\right)\quad \text{for some } \sigma\in k(t),\quad \text{or} \tag{5.24b}$$

$$k(t)\cdot(\langle p_2\rangle+\Lambda)=k(t)\cdot(\langle(\sigma U+V)(\tau U+V)\rangle+\Lambda)\quad \text{for some } \sigma,\ \tau\in k(t) \tag{5.24c}$$

where always $t=X/l$.

We return to the original structure and discuss now p_2 explicitly.

Case (*). $p_2=(\alpha_1(X/l)+\alpha_2X+l_1)l_2$ where $\alpha_1,\ \alpha_2\in k$, and w.l.o.g. $l_1,\ l_2\in\Lambda^{(1)}_{\{U,V\}}$ (by (5.23)). If $\alpha_1=0$ then $L(p_2)\leqq 1$, and Conclusion 3.28 (i) implies that $\alpha_2=0$, thus (5.20). Otherwise, w.l.o.g. $\alpha_1=1$. Then

$$p_2=\frac{(1+\alpha_2l)l_2}{l}X+l_1l_2\qquad (l,\ l_2\in\Lambda^{(1)}_{\{U,V\}}\backslash\{0\}).$$

We show that this is impossible, proving (5.20) if p_2 results from a multiplication. By way of contradiction, assume p_2 to have the above form. A comparison of the coefficients of X in (5.21) gives

$$\frac{V}{U}\in\left\langle\frac{1}{l},\frac{(1+\alpha_2l)l_2}{l}\right\rangle+\Lambda^{(0)}+\frac{N}{U}\left\langle\frac{1}{l},\frac{(1+\alpha_2l)l_2}{l}\right\rangle+\frac{N}{U}\Lambda^{(0)},$$

and multiplication by Ul yields

$$lV\in\langle U,U(1+\alpha_2l)l_2\rangle+Ul\Lambda^{(0)}+N\langle 1,(1+\alpha_2l)l_2\rangle+lN\Lambda^{(0)}. \tag{5.25}$$

The homogeneous parts of degree two show that $U(1+\alpha_2l)l_2$ *must* participate in any linear combination behind (5.25) (by the irreducibility of N; discuss participation of

N), and then Steinitz exchange and a comparison of the homogeneous parts of degree three give

$$\alpha_2 U l l_2 \in N\langle l_2\rangle + lN\Lambda^{(0)}$$

(remember that $l,\ l_2 \in \Lambda^{(1)}_{\{U,V\}}\backslash\{0\}$). Thus

$$\alpha_2 = 0, \qquad p_2 = \left(\frac{X}{l} + l_1\right) l_2 \tag{5.26}$$

since the left-hand side must be divisible by the irreducible N. Since the transition (5.21) → (5.25) is a k-linear mapping and $U(1+\alpha_2 l)l_2$ must participate in (5.25), we conclude that p_2 must participate in (5.21). Thus, after Steinitz exchange, we get for the X free terms

$$l_1 l_2 \in \left\langle \frac{V^2}{U} \right\rangle + \Lambda_{\{U,V\}} + \frac{N}{U}\left\langle l_1 l_2, \frac{V^2}{U}\right\rangle + \frac{N}{U}\Lambda_{\{U,V\}}.$$

Multiplying by U^2, we get

$$U^2 l_1 l_2 \in (\langle UV^2\rangle + U^2\Lambda_{\{U,V\}} + N\langle U l_1 l_2,\ V^2\rangle + UN\Lambda_{\{U,V\}})^{(4)} = N\langle V^2\rangle + UN\Lambda^{(1)}_{\{U,V\}},$$

and since the left-hand side must be divisible by the irreducible N of degree two we obtain

$$l_1 l_2 = 0, \qquad \text{thus } l_1 = 0.$$

But this and (5.26) show that p_2 is linear over the extended coefficient field $k(t) = k(X/l)$, contrary to the possible alternatives in (5.24).

Case (/). If p_2 results from a division then

$$p_2 = \frac{\alpha_1 p_1 + \alpha_2 X + l_1}{\beta_1 p_1 + \beta_2 X + l_2} = \frac{\alpha_1 X + (\alpha_2 X + l_1)l}{\beta_1 X + (\beta_2 X + l_2)l}$$

where $\alpha_1,\ \alpha_2,\ \beta_1,\ \beta_2 \in k,\ l_1,\ l_2 \in \Lambda_{\{U,V\}}$. By (5.23) either

$$\beta_1 = \beta_2 = 0 \quad \text{and then } l_2 \notin k \tag{5.27}$$

(by the optimality of the computation sequence), or otherwise numerator and denominator (both having degree $\leqq 2$) of the above quotient representation of p_2 have a factor containing X in common, and then p_2 is already the quotient of two elements from $\Lambda_{\{U,V\}}$, thus the desired (5.20). In case of (5.27) we adjoin $t = p_1 = X/l$ to k and delete the o-ary X. So

$$p_2 = \frac{\alpha_1 t + \alpha_2 t l + l_1}{l_2} \qquad (l,\ l_1,\ l_2 \in \Lambda_{\{U,V\}}). \tag{5.28}$$

A comparison with (5.24 a, b) shows that numerator and denominator must be contained in $k(t)\cdot\Lambda^{(1)}_{\{U,V\}} = k(t)\cdot U + k(t)\cdot V$, thus $\alpha_1 t + l_1^{(0)} = l_2^{(0)} = 0$, thus

$$\alpha_1 = 0 \quad \text{and} \quad l_1,\ l_2 \in \Lambda^{(1)}_{\{U,V\}}$$

since t is transcendental over k. If $\alpha_2 = 0$, too, we again arrive at the desired (5.20) (see (5.28)). Otherwise w.l.o.g. $\alpha_2 = 1$, and

$$p_2 = \frac{X + l_1}{l_2} \quad \text{with } l_1,\ l_2 \in \Lambda^{(1)}_{\{U,V\}}, \quad l_2 \neq 0.$$

We show that this is impossible, completing the proof of (5.20). We return to (5.21) and compare the coefficients of X

$$\frac{V}{U} \in \left\langle \frac{1}{l'}, \frac{1}{l_2} \right\rangle + \Lambda^{(0)} + \frac{N}{U}\left\langle \frac{1}{l'}, \frac{1}{l_2} \right\rangle + \frac{N}{U}\Lambda^{(0)}.$$

Multiplying by Ull_2, we get

$$ll_2 V \in U\langle l_2, l\rangle + Ull_2\Lambda^{(0)} + N\langle l_2, l\rangle + ll_2 N\Lambda^{(0)},$$

thus for suitable $\rho, \sigma, \tau \in k$

$$N \mid (ll_2 V + \rho Ul_2 + \sigma Ul + \tau Ull_2)^{(3)} = ll_2(V + \tau U) \neq 0,$$

a contradiction, since N is irreducible of degree two.

This finishes the proof of Theorem 5.1. □

6. Counting additions and subtractions as well. Let (ω_1, ω_2) be any k-basis of the extension field K of k, let $\omega = \omega_1\omega_2^{-1}$ with

$$\omega^2 + a\omega + b = 0 \qquad (a, b \in k),$$

and $\omega_2^{-1} = e\omega + f$ $(e, f \in k)$. Division in basis (ω_1, ω_2) takes the form

$$\frac{X\omega_1 + Y\omega_2}{U\omega_1 + V\omega_2} = \frac{(e\omega + f)(X\omega + Y)}{U\omega + V}\omega_2 = \frac{((f - ae)X + eY)\omega + (-beX + fY)}{U\omega + V}\omega_2$$

$$= \frac{A}{N}\omega_1 + \frac{B}{N}\omega_2$$

where (by straightforward calculation using (0.2))

$$A = beXU + (f - ae)XV - fYU + eYV,$$

$$B = bfXU - beXV - (fa - be)YU + fYV,$$

$$N = bU^2 - aUV + V^2.$$

Note that for all $(e, f) \neq (0, 0)$ these are irreducible polynomials $\in k[X, Y, U, V]$, and pairwise relatively prime (as regularly variable transformed versions of the irreducible numerators and denominators occurring in (0.3)). We are now interested in the minimum number $L_+(A/N, B/N)$ of additions and subtractions necessary to compute A/N and B/N (so we leave multiplications and divisions free of charge and use the cost function $1_{\{+,-\}}$ on $\Omega^k_{\{X,Y,U,V\}}$).

LEMMA 6.1. *For all* $(e, f) \neq (0, 0)$ *we have*

(i) $L_+(A/N, B/N) \geqq 3$,

(ii) $L_+(A/N, B/N) \geqq 4$ if $K \supset k$ *is separable*, char $k = 2$.

Proof. (i) If $L_+(A/N, B/N) \leqq 2$ then there are

$$P = \alpha M_1 + \beta M_2 \qquad (\alpha, \beta \in k),$$
$$Q = \gamma M_3 P^i + \delta M_4 P^j \qquad (\gamma, \delta \in k;\ i, j \in \mathbb{N}) \tag{6.2}$$

where $M_1, \cdots, M_4$ are monomials such that

$$\frac{A}{N}, \frac{B}{N} \in k \cdot X^{\mathbb{Z}} Y^{\mathbb{Z}} U^{\mathbb{Z}} V^{\mathbb{Z}} P^{\mathbb{Z}} Q^{\mathbb{Z}}.$$

(Since multiplications and divisions are free P and Q can be assumed to be denominator free.) Thus there are exponents n, $m \in \mathbb{Z}$, not both zero, such that

$$A^n B^m N^{-n-m} \in k \cdot X^{\mathbb{Z}} Y^{\mathbb{Z}} U^{\mathbb{Z}} V^{\mathbb{Z}} P^{\mathbb{Z}}.$$

In any case, the left-hand side contains two of the irreducible A, B, N of degree two having a positive and a negative exponent, respectively. But the right-hand side never fulfills this. So $L_+(A/N, B/N) \leqq 2$ is impossible.

(ii) W.l.o.g. k perfect (i.e. square roots of elements of k are in k). By separability of $K \supset k$, and $\operatorname{char} k = 2$, $ab \neq 0$. By the way of contradiction we assume $L_+(A/N, B/N) = 3$ in this case. Then there are exponents n, $m \in \mathbb{Z}$, not both zero, such that

$$A^n B^m N^{-n-m} \in k \cdot X^{\mathbb{Z}} Y^{\mathbb{Z}} U^{\mathbb{Z}} V^{\mathbb{Z}} P^{\mathbb{Z}} Q^{\mathbb{Z}} \tag{6.3}$$

where P and Q are as in (6.2) (with $\alpha\beta\gamma\delta \neq 0$). Since multiplications and divisions are free, we may further assume that no variable divides P and Q, and that $j = 0$ in (6.2), say. Then (6.3) implies

$$\langle A^n B^m N^{-n-m} \rangle = \langle P^z Q^w \rangle \tag{6.4}$$

for some z, $w \in \mathbb{Z}$ with different sign (for $A^n B^m N^{-n-m}$, $A^{-n-m} N^{n+m} \notin k[X, Y, U, V]$).

Case (i $= 0$). Then $Q = \gamma M_3 + \delta M_4$. If $n \neq 0$ then $A \mid P$, or $A \mid Q$. So P (resp. Q) is not X free since A is not (by $(e, f) \neq (0, 0)$, $ab \neq 0$), and since no variable divides P (resp. Q), exactly one monomial in P (resp. Q) is divisible by X. Specializing $X \mapsto 0$, P (resp. Q) reduces to a monomial, thus $ef = 0$ since $A \mid P$ (resp. $A \mid Q$). Specializing $Y \mapsto 0$, the very same reasoning gives $e(f - ae) = 0$, thus (by $(e, f) \neq (0, 0)$)

$$e = 0 \quad \text{if} \quad n \neq 0. \tag{6.5}$$

If $m \neq 0$ then $B \mid P$, or $B \mid Q$, and an analogous argument gives $(fa - be)f = 0$ and $b^2 fe = 0$, thus

$$f = 0 \quad \text{if } m \neq 0. \tag{6.6}$$

Equations (6.5) and (6.6) cannot both be fulfilled, so in particular $n \neq -m$ (one of them being nonzero). Therefore (by (6.4)) there is some maximal power N^s dividing P (say), so

$$HN^s = P \tag{6.7}$$

for some $H \in k[X, Y, U, V]$. We claim that $H \in k$. First of all $H \in k[U, V]$, for otherwise say $X \mapsto 0$ would kill exactly one monomial in P, and N would divide a monomial, that being impossible. We show that H is a monomial, so $H \in k$ since no variable divides P. The left-hand side in (6.4) shows that $H \in k[U, V]$ must also divide $Q[N \nmid H]$. So, since one of A and B also divides Q, we see that H divides a monomial, specializing $X \mapsto 0$, say. Thus w.l.o.g. $H = 1$. Then, since no variable divides P, (6.7) gives

$$\begin{aligned} N^s = P &= \alpha M_1 + \beta M_2 = \alpha U^{2s} + \beta V^{2s} \quad \text{(say)} \\ &= (\sqrt{\alpha}\, U^s + \sqrt{\beta}\, V^s)^2 \quad (k \text{ is perfect, char } k = 2). \end{aligned}$$

So (by induction) s must be a power of 2, and $N \in \langle U^2, V^2 \rangle$, contradicting $a \neq 0$.

Case (i > 0). Here P and Q are relatively prime since no variable divides these, and a common divisor would be a divisor of the monomial M_4 (by $\delta \neq 0$). As above, not both A and B divide P and $\langle P \rangle \neq \langle N^s \rangle$ for all $s \in \mathbb{N}$. So say $A \mid P$. As above this implies that $e = 0$, and thus $f \neq 0$. We also have $N \nmid P$ since $X \mapsto 0$ reduces P to a monomial, thus $\langle A^s \rangle = \langle (XV - YU)^s \rangle = \langle P \rangle$ for some s, and $B = bfXU - faYU + fYV$

divides $Q = \gamma M_3 P^i + \delta M_4$ since $A \nmid Q$ (P, Q rel. prime) and $Q \notin k[U, V]$ (otherwise $Q = \delta M_4$, by X, $Y \mapsto 0$). So by (6.4) B and N are the only possible prime factors in Q. Specializing X, $U \mapsto 0$, we see that $V | M_4$. So, $V \mapsto 0$ reduces P and thus Q to a monomial; but not the divisor B (by $\text{fab} \neq 0$), a contradiction. Analogously, the case $B | P$. □

We summarize our results as follows.

THEOREM 6.8. *For any quadratic field extension $k \subset K$, and any k-basis (ω_1, ω_2) of K, division of two elements from K in basis (ω_1, ω_2) requires at least six nonscalar multiplications and divisions over k, and three additions and subtractions if $k \subset K$ is a radical extension, else four additions and subtractions. These lower bounds are achieved for bases $(\omega, 1)$, with some radical ω in case of a radical extension.*

Proof. Combine Theorem 5.1, Lemma 6.1 and computation (3.15). □

COROLLARY. *Complex division in real arithmetic requires exactly nine real arithmetic operations.*

Acknowledgments. Sincere thanks are due to Arnold Schönhage for encouraging me to face this problem and, in particular, for carefully reading the manuscript. Likewise I am grateful for a clarifying discussion with Walter Baur, who brought in the interpretation of adjoining intermediate results. Also thanks are due to the referee for his valuable comments on an earlier version of the paper and to Hanns-Jörg Stoss for carefully reading the manuscript.

REFERENCES

[1] A. ALDER AND V. STRASSEN, *On the algorithmic complexity of associative algebras*, Theoret. Comput. Sci., 15 (1981), pp. 201-211.

[2] H. ALT AND J. VAN LEEUWEN, *The complexity of basic complex operations*, Technical Report RUU-CS-79-4, University of Utrecht (1979), extended abstract in: Proc. Fundamentals of Computation Theory FTC '79 (1979), pp. 13-17.

[3] D. BINI AND M. CAPOVANI, *Lower bounds of the complexity of linear algebras*, Inform. Process. Lett., 9 (1979), pp. 46-47.

[4] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[5] B. BUCHBERGER, G. E. COLLINS AND R. LOOS, *Computer Algebra—Symbolic and Algebraic Computation*, Computing Suppl. 4, Springer, Berlin-New York, 1982.

[6] G. E. COLLINS, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Lecture Notes in Comput. Sci., 33 (1975), pp. 134-183.

[7] C. M. FIDUCCIA AND Y. ZALCSTEIN, *Algebras having linear multiplicative complexities*, J. Assoc. Comput. Mach., 24 (1977), pp. 311-331.

[8] H. F. DE GROOTE, *On varieties of optimal algorithms for the computation of bilinear mappings* I: *The isotropy group of a bilinear mapping*, Theoret. Comput. Sci., 7 (1978), pp. 1-24.

[9] ———, *On varieties of optimal algorithms for the computation of bilinear mappings* II: *Optimal algorithms for 2×2-matrix multiplication*, Theoret. Comput. Sci., 7 (1978), pp. 127-148.

[10] D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[11] J. VAN LEEUWEN AND P. VAN EMDE BOAS, *Some elementary proofs of lower bounds in complexity theory*, Linear Algebra Appl., 19 (1978), pp. 63-80.

[12] T. LICKTEIG, *Gaussian elimination is optimal for solving linear equations in dimension two*, Inform. Process. Lett., 22 (1986), pp. 277-279.

[13] I. MUNRO, *Some results concerning efficient and optimal algorithms*, Proc. 3rd Annual ACM Symp. on Theory of Computing, 1971, pp. 40-44.

[14] A. M. OSTROWSKI, *On two problems in abstract algebra connected with Horner's rule*, in Studies Presented to R. von Mises, Academic Press, New York, 1954, pp. 40-48.

[15] V. IA. PAN, *Methods of computing values of polynomials*, Russian Math. Surveys, 21 (1966), pp. 105-136.

[16] R. L. SMITH *Algorithm* 116: *complex division*, Comm. ACM., 5 (1962), p. 435.

[17] E. SPECKER AND V. STRASSEN, *Komplexität von Entscheidungsproblemen*, Lecture Notes in Comput. Sci., 43 (1976).

[18] H. J. STOSS, *Zur Darstellung einiger quadratischer Formen*, unpublished manuscript (Universität Konstanz, 1976).
[19] V. STRASSEN, *Berechnnung und Programm* I, Acta Inform., 1 (1973), pp. 320-335.
[20] ———, *Berechnung und Programm* II, Acta Inform., 2 (1973), pp. 64-79.
[21] ———, *Polynomials with rational coefficients which are hard to compute*, this Journal, 3 (1974), pp. 128-149.
[22] S. WINOGRAD, *On the multiplication of* 2×2 *matrices*, Linear Algebra Appl., 4 (1971), pp. 381-388.
[23] ———, *On the multiplication in algebraic extension fields*, Theoret. Comput. Sci., 8 (1979), pp. 359-377.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
007

AN APPLICATION OF DUALITY TO EDGE-DELETION PROBLEMS*

TAKAO ASANO†

Abstract. For a property π on graphs, the corresponding edge-deletion problem $P_{ED}(\pi)$ (on planar graphs) is stated as follows: given a (planar) graph G, find a set of edges of minimum cardinality whose deletion results in a graph satisfying π. We show that the edge-deletion problem $P_{ED}(\pi)$ on planar graphs is NP-hard if π is nontrivial and is determined by the weighted 3-connected components. As a corollary, the edge-deletion problem $P_{ED}(\pi)$ on planar graphs is NP-complete for several properties π, such as π = "series-parallel," "outerplanar," "without cocycles of cardinality $\geqq 3$," etc.

Key words. computational complexity, connectivity, duality, edge-contraction, edge-deletion, graph property, matroids, maximum subgraph, NP-complete, planar graph, outerplanar graph, series-parallel graph

AMS(MOS) subject classifications. 68Q15, 68R10, 05C40, 05C35

1. Introduction. For a property π on graphs, the corresponding edge-deletion problem $P_{ED}(\pi)$ is stated as follows: given a graph G, find a set of edges of minimum cardinality whose deletion results in a graph satisfying π. An equivalent formulation of this problem is the maximum subgraph problem: given a graph G, find a maximum subgraph of G that satisfies π.

Since the founding of the theory of NP-completeness by Cook [8] and Karp [17], [18], a wide variety of commonly encountered problems have been known to be NP-complete and the collection of such problems continues to grow almost daily [1], [11]. Recently, an effort has begun toward systematizing NP-completeness proofs and attacking groups of similar problems as a whole rather than individually [5], [6], [19], [20], [21], [26], [30], [31], [33]. Problems considered in these papers belong to the classes of node-deletion problems, edge-deletion problems and edge-contraction problems. For node-deletion problems, generic NP-completeness proofs with wide applicability were constructed [19]–[21], [30], [31], [33]. However, for edge-deletion problems, although NP-completeness proofs were constructed [5], [26], none of them is sufficiently "generic" to counterbalance the amount of work that one must invest if one is to verify it completely. In this paper, we give a sufficiently "generic" NP-completeness proof for edge-deletion problems.

Many graph problems may be viewed as edge-deletion problems. For example, the maximum matching problem, the max cut problem and the Chinese postman problem can be formulated as edge-deletion problems. In fact, if π = "all nodes of degree $\leqq 1$," then the edge-deletion problem $P_{ED}(\pi)$ is the maximum matching problem. If π = "bipartite" ("Eulerian," resp.), then $P_{ED}(\pi)$ is equivalent to the max cut problem (the Chinese postman problem, resp.) with weights $w(e)=1$ for all edges e. The maximum matching problem and the Chinese postman problem are polynomially solvable [9]. On the other hand, the max cut problem is NP-complete [12], [17].

Thus, it may be natural to ask what is a necessary and sufficient condition on a property π for the corresponding edge-deletion problem $P_{ED}(\pi)$ to be NP-complete (polynomially solvable). However, a characterization is difficult to obtain. Moreover,

* Received by the editors August 14, 1984; accepted for publication (in revised form) June 17, 1986.

† Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Bunkyo-ku, Tokyo, Japan 113. Present address, Department of Mechanical Engineering, Faculty of Science and Technology, Sophia University, Kioi-cho, Chiyoda-ku, Tokyo, Japan 102. This research was supported by the Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under grant 59550239 (1984).

the characterization will change if we restrict ourselves to planar graphs, because, for example, the max cut problem, which is NP-complete in general, turns out to be polynomially solvable if it is restricted to planar graphs [13], [23]. Thus it might be interesting to give a general sufficient condition on a property π for the corresponding edge-deletion problem $P_{\mathrm{ED}}(\pi)$ to be NP-hard even if it is restricted to planar graphs.

In this paper, we give such a sufficient condition: $P_{\mathrm{ED}}(\pi)$ on planar graphs is NP-hard if π is nontrivial on planar graphs, is determined by the weighted 3-connected components and is satisfied by small graphs. The class of properties π for which $P_{\mathrm{ED}}(\pi)$ are established to be NP-hard in the previous papers [5], [26] is a proper subset of the class of properties satisfying the conditions given in this paper [2]. Furthermore, we would like to lay special emphasis on the proof technique: the proof of the NP-hardness of $P_{\mathrm{ED}}(\pi)$ in this paper is simple and much easier to understand than that in the previous papers [5], [26]. It is based on duality (of matroids). Thus, we shall discuss the computational complexity of both the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ and the edge-contraction problem $P_{\mathrm{EC}}(\pi)$ simultaneously, where $P_{\mathrm{EC}}(\pi)$ is: given a graph G, find a set of edges of minimum cardinality whose contraction results in a graph satisfying π.

We first introduce a notion of properties determined by the 3-connected components and the idea of duality, and give two propositions on the edge-deletion problem for a property determined by the 3-connected components. Next, we introduce a new notion of properties, determined by the weighted 3-connected components, which is a generalization of a notion of properties determined by the 3-connected components, and then present the main result that $P_{\mathrm{ED}}(\pi)$ on planar graphs (general graphs, resp.) is NP-hard if π is nontrivial on planar graphs (general graphs, resp.), is determined by the weighted 3-connected components and is satisfied by small graphs. Finally, we examine the problem restricted to 3-connected planar graphs.

2. Preliminaries. For terminology on graph theory, we follow Harary [14]. We consider only finite undirected multigraphs. For a graph G we denote by $V(G)$ and $E(G)$ the node set and the edge set of G, respectively. For $U \subset V(G)$, we denote by $G-U$ the subgraph of G obtained from G by deleting all nodes in U and all edges incident with nodes in U. For $S \subset E(G)$, we denote by $G-S$ the *subgraph* of G obtained by deleting all edges in S and by G/S the *contraction* of G obtained by contracting all edges in S. For two disjoint subsets S_c and S_d of $E(G)$, $(G-S_d)/S_c$ is a *subcontraction* of G. An edge of G is a *loop* (*coloop*, resp.) if it is a cycle (cocycle, resp.) of G. Two distinct edges e and e' of G are *in parallel* (*in series*, resp.) if $\{e, e'\}$ is a cycle (cocycle, resp.) of G, and e and e' are then said to be *parallel-edges* (*series-edges*, resp.). A node v of a connected graph G is an *articulation point* if $G-v$ is disconnected. A connected graph G is 2-*connected* if G has no articulation point. 2-*connected components* of G are maximal 2-connected subgraphs of G. An unordered pair $\{u, v\}$ of distinct nodes in G is a *separation pair* of G if there exist two subgraphs G_1' and G_2' satisfying the following conditions (a)-(c):

(a) $V(G)=V(G_1')\cup V(G_2')$, $V(G_1')\cap V(G_2')=\{u, v\}$;

(b) $E(G)=E(G_1')\cup E(G_2')$, $E(G_1')\cap E(G_2')=\varnothing$, $|E(G_1')|\geqq 2$, $|E(G_2')|\geqq 2$;

(c) For some $e_1\in E(G_1')$ and $e_2\in E(G_2')$, there is a cycle of G containing both e_1 and e_2.

Then the graphs G_1' and G_2' are called *separation graphs* of G with respect to $\{u, v\}$. Define G_i $(i=1, 2)$ as the graph obtained from G_i' by adding a new edge (u, v). The graphs G_1 and G_2 are called *split graphs* of G with respect to $\{u, v\}$ and the common new edge is called a *virtual edge* (Fig. 1). Decomposing a graph G into two split graphs

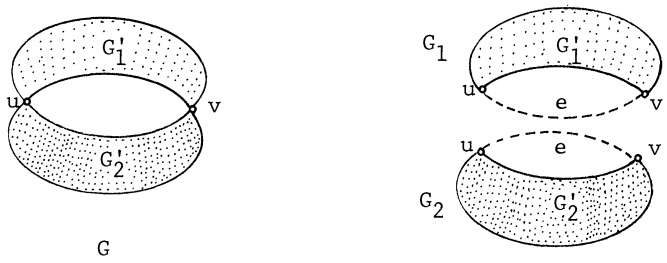
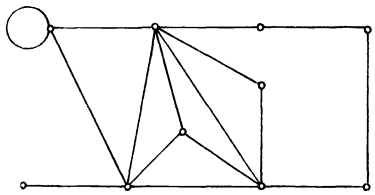


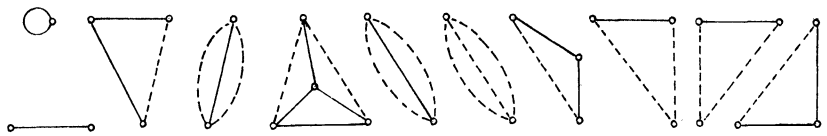
FIG. 1. *A separation pair* $\{u, v\}$ *of a graph* G *and split graphs* G_1 *and* G_2 *with a virtual edge* e.

G_1 and G_2 is called *splitting*. A 2-connected graph G is called 3-*connected* if G has no separation pair. A 3-connected graph with three or more nodes is a simple graph. Maximal 3-connected subcontractions of G are called 3-*connected components* of G. A decomposition of a graph G into 3-connected components is obtained by first dividing G into 2-connected components $G_1, G_2, \cdots, G_j$ and then repeating the splitting for each G_i $(i = 1, 2, \cdots, j)$ until no more splitting is possible. The 3-connected components of a graph are the following types: K_1, C_1, K_2, C_2, K_3, B_3 and 3-connected graphs with four or more nodes, where K_n denotes the complete simple graph with n nodes, C_n the cycle of length n and B_n the bond of consisting of n parallel edges connecting two nodes. Note that if D_1 and D_2 are two decompositions of a graph G into 3-connected components, then there is a bijection $f: D_1 \to D_2$ such that G_i is isomorphic to $f(G_i)$ for any G_i in D_1 [15], [22], [25]. Hereafter, we shall not distinguish between two isomorphic graphs and consider any two isomorphic graphs to be equal. Thus, the decomposition of a graph into 3-connected components can be considered to be unique (Fig. 2). For a planar graph G, we denote by G^* a dual graph of G. Let $D(G) = \{G_1, G_2, \cdots, G_j\}$ be a decomposition of G into 3-connected components. Then $\{G_1^*, G_2^*, \cdots, G_j^*\}$ is a decomposition of G^* into 3-connected components [25].

Throughout this paper, we consider only properties π on graphs such that they can be generalized to ones on matroids and, for any two isomorphic matroids M' and M'', M' satisfies π if and only if M'' satisfies π. Examples of such properties are



G



D(G)

FIG. 2. *A decomposition* $D(G)$ *of a graph* G *into* 3-*connected components.*

"planar," series-parallel," "outerplanar," "Eulerian," "k-colorable," etc. [24], [28]. This implies that we consider only properties π satisfying the following condition (graphs G' and G'' are 2-*isomorphic* if the corresponding polygon matroids $M(G')$ and $M(G'')$ are isomorphic, that is, there is a bijection $h: E(G') \to E(G'')$ such that, for any $C \subset E(G')$, C is a cycle of G' if and only if $h(C) = \{h(e) | e \in C\}$ is a cycle of G'' [29]):

(C0) π is invariant under 2-isomorphisms, that is, for any two 2-isomorphic graphs G' and G'', G' satisfies π if and only if G'' satisfies π [28].

Thus, hereafter, we will not distinguish between two 2-isomorphic graphs and will consider two-isomorphic graphs to be equal. Note that, for any planar graph G, all dual graphs of G are 2-isomorphic [29]. Thus, we consider any two dual graphs of G to be equal and use the term *the* dual graph of G.

A property π on graphs is *nontrivial on graphs satisfying the prescribed property* π' if it is false for infinitely many graphs satisfying π'. A property π is *hereditary on subgraphs* (*hereditary on contractions*, resp.) if, for any graph G satisfying π, all subgraphs (contractions, resp.) of G satisfy π. For properties π and π^*, π^* is the *dual* property of π if, for any planar graph G, G satisfies π if and only if the dual graph G^* of G satisfies π^*. A property π is *determined by the* 3-*connected components* if, for any graph G, G satisfies π if and only if all 3-connected components of G satisfy π [2].

Examples of a property determined by the 3-connected components are "planar," "series-parallel," "without cycles of length $\geqq 3$," "without cocycles of cardinality $\geqq 3$," etc.

Note that if π is determined by the 3-connected components then, for any graph G, G satisfies π if and only if all 2-connected components of G satisfy π, that is, π is *determined by the* 2-*connected components.* Since a decomposition of a graph into 3-connected components occurs only within isomorphisms, if π is determined by the 3-connected components then, for any graph G, G satisfies π if and only if, for arbitrary decomposition $D(G)$ of G into 3-connected components, all graphs in $D(G)$ satisfy π. Thus, if π is determined by the 3-connected components then, for any graph G with split graphs G_1 and G_2, G satisfies π if and only if both G_1 and G_2 satisfy π. Note also that if π is determined by the 3-connected components, then π satisfies the condition (C0), because, for two 2-isomorphic graphs G' and G'', there is a bijection $f: D(G') \to D(G'')$ such that, for any G_i in $D(G')$, G_i is isomorphic to $f(G_i)$, where $D(G')$ and $D(G'')$ are decompositions of G' and G'' into 3-connected components, respectively.

Let π be a property on graphs. Formally, the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ and the edge-contraction problem $P_{\mathrm{EC}}(\pi)$ are defined as follows:

Edge-deletion problem $P_{ED}(\pi)$.

Instance: A graph G and a nonnegative integer k.

Question: Is there a subset S of $E(G)$ with $|S| \leqq k$ such that the subgraph $G - S$ satisfies π?

Edge-contraction problem $P_{\mathrm{EC}}(\pi)$.

Instance: A graph G and a nonnegative integer k.

Question: Is there a subset S of $E(G)$ with $|S| \leqq k$ such that the contraction G/S satisfies π?

For a property π, let π^* be the dual property of π. Since edge-deletion and edge-contraction are dual operations on graphs and the dual graph G^* of a planar graph G can be obtained in a polynomial order of $|V(G)|$ and $|E(G)|$ by a linear time planar embedding algorithm [16], the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ on planar graphs

is polynomially solvable (NP-hard, resp.) if and only if the edge-contraction problem $P_{EC}(\pi^*)$ on planar graphs is polynomially solvable (NP-hard, resp.). Thus, we can prove the NP-hardness of $P_{ED}(\pi)$ by first proving the NP-hardness of $P_{EC}(\pi^*)$ and then using this duality of planar graphs. Note that, in general, it is very easy to prove the NP-hardness of edge-contraction problems [6]. Thus, it is not difficult to obtain the following proposition.

PROPOSITION 1. *The edge-deletion problem $P_{ED}(\pi)$ on simple 2-connected planar graphs is* NP*-hard if π satisfies the following conditions* (C1)-(C3):

(C1) *π is nontrivial on planar graphs.*

(C2) *π is determined by the 3-connected components.*

(C3) *π is satisfied by all graphs K_1, C_1, K_2, C_2 and K_3.*

Although π is trivial on planar graphs, we can obtain the following.

PROPOSITION 2. *The edge-deletion problem $P_{ED}(\pi)$ on simple 2-connected graphs is* NP*-hard if π satisfies the following condition* (D1) *as well as the conditions* (C2) *and* (C3) *in Proposition* 1:

(D1) *π is nontrivial (on general graphs).*

Note that there are properties trivial on planar graphs but nontrivial in general. "Planar" is an example of such a property. Examples of a property satisfying the conditions in Proposition 1 or 2 are "planar," "series-parallel," "without cocycles of cardinality $\geqq 3$," etc. We will not prove the propositions. Instead, in §§ 3 and 4 we introduce a new notion and give a main result from which Propositions 1 and 2 can be obtained as corollaries.

3. Properties determined by the weighted 3-connected components. Although properties of "planar," series-parallel," "without cocycles of cardinality $\geqq 3$," etc. satisfy the conditions in Proposition 1 or 2, none of "outerplanar," "co-outerplanar" (=dual property of "outerplanar") and "ladder" (="outerplanar" and "co-outerplanar") [24] is determined by the 3-connected components. However, each of them has a characterization similar to Kuratowski's characterization for planar graphs: a graph G is planar if and only if G has no subgraph homeomorphic to K_5 or $K_{3,3}$ [14]. For example, a graph G is outerplanar (co-outerplanar, resp.) if and only if G has no subgraph homeomorphic to K_4 or $K_{2,3}$ (K_4 or $K^*_{2,3}$, resp.) [14], [24] (Fig. 3). Thus these properties of "outerplanar," "co-outerplanar" and "ladder" may be closely related to properties determined by the 3-connected components.

In this section we introduce a new notion of properties, determined by the weighted 3-connected components, which is a generalization of a notion of 3-connected components, and in § 4 we shall prove a theorem for the edge-deletion problem for a

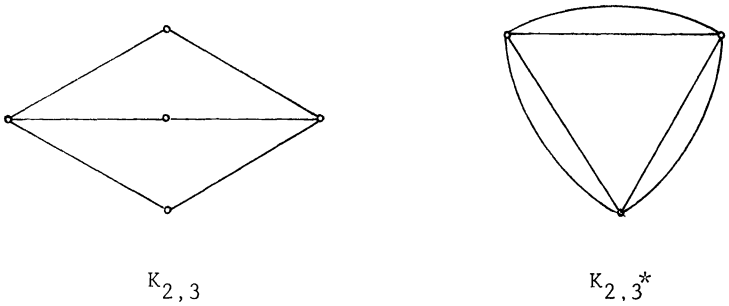


FIG. 3. *Graphs $K_{2,3}$ and $K^*_{2,3}$.*

property determined by the weighted 3-connected components, from which Propositions 1 and 2 can be obtained as corollaries. Of course, the class of properties determined by the weighted 3-connected components contains not only the class of properties determined by the 3-connected components but also properties such as "outerplanar," "co-outerplanar," "ladder," etc.

For a graph G, let G_1 and G_2 be split graphs with respect to a separation pair $\{u, v\}$ of G and e be the virtual edge of G_1 and G_2. Reassembling the two split graphs G_1 and G_2 into G is called *merging*. Merging is the inverse of splitting. Let $D(G)$ be the decomposition of a 2-connected graph G with four or more nodes into 3-connected components. Then we can represent $D(G)$ as $D(G) = \mathcal{G} \cup \mathcal{P}_3 \cup \mathcal{B}_3$, where $\mathcal{G}$ is the set of 3-connected graphs with four or more nodes in $D(G)$, $\mathcal{P}_3$ is the set of triangles K_3 in $D(G)$, and $\mathcal{B}_3$ is the set of triple bonds B_3 in $D(G)$. Suppose that the triangles in $\mathcal{P}_3$ are merged as much as possible to give a set of cycles $\mathcal{P}$ and the triple bonds in $\mathcal{B}_3$ are merged as much as possible to give a set of bonds $\mathcal{B}$. Graphs in $F(G) = \mathcal{G} \cup \mathcal{P} \cup \mathcal{B}$ are called 3-*blocks* of G and $F(G)$ is called a decomposition of G into 3-blocks (Fig. 4). Note that a decomposition of a 2-connected graph into 3-blocks is unique [15], [22], [25]. Any decomposition $D(G)$ of a 2-connected graph G can be obtained from the decomposition $F(G)$ of G into 3-blocks by appropriately splitting cycles in $\mathcal{P}$ into triangles and bonds in $\mathcal{B}$ into triple bonds.

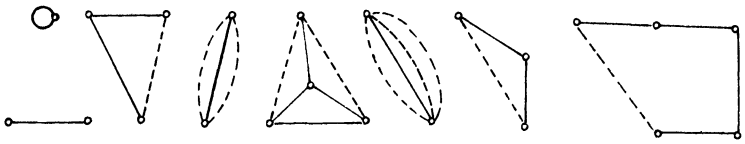


FIG. 4. *The decomposition of G into* 3*-blocks* (*G is the graph in Fig.* 2).

For a graph G, let $D(G) = \{G_1, G_2, \cdots, G_j\}$ be a decomposition of G into 3-connected components. Let c be a fixed positive integer. For each graph G_i in $D(G)$, we assign weights to all edges of G_i as follows: if e is a virtual edge then $c(e) = c$ and otherwise $c(e) = 1$. Let $(G_i; c)$ denote the graph G_i with weights $c(e)$ defined above. Let $(G_i; c)\#$ be a graph G_i with a weight function c' such that $c'(e) \geqq c(e)$ for all edges e in G_i. $(G_i; c)$ is called a 3-*connected component with weight* c of G, or simply a *weighted* 3-*connected component* of G. A property π is *determined by the* 3-*connected components with weight* c if, for any graph G, G violates π if and only if there is a 3-connected component G_i of G such that all $(G_i; c)\#$ violate π. A property π is *determined by the weighted* 3-*connected components* if there is a positive integer c such that π is determined by the 3-connected components with weight c. Note that if π is determined by the 3-connected components with weight c and satisfied by some small graphs then it is determined by the 3-connected components with weight c' for any $c' \geqq c$ (see Lemma 2). If virtual edges with weight c are considered to be paths of length c, then we often use terms such as "$(G_i; c)$ is the weighted 3-connected component of *path type*," "π is determined by the 3-connected components with weight c of *path type*," "π is determined by the weighted 3-connected components of *path type*," etc. (Fig. 5). Otherwise (i.e., if virtual edges are considered to be sets of parallel-edges), we use terms such as "$(G_i; c)$ is the weighted 3-connected component of *cut type*," "π is determined by the 3-connected components with weight c of *cut*

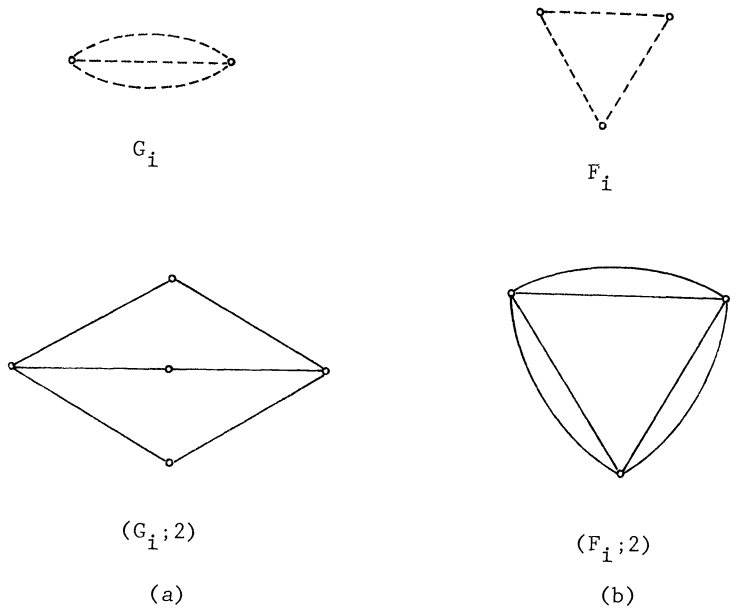


FIG. 5. *Examples of weighted 3-connected components.* (a) $(G_i; 2)$ *of path type.* (b) $(F_i; 2)$ *of cut type.*

type," "π is determined by the weighted 3-connected components of *cut type*," etc. (Fig. 5).

"Planar," "series-parallel," "outerplanar," etc. are examples of such a property. In fact, "planar," "series-parallel" and "without cocycles of cardinality $\geqq 3$" are determined by the 3-connected component with weight 1 of path type, and "planar," "series-parallel" and "without cycles of cardinality $\geqq 3$" are determined by the 3-connected components with weight 1 of cut type. Similarly, "outerplanar" and "ladder" ("co-outerplanar" and "ladder," resp.) are determined by the 3-connected components with weight 2 of path type (of cut type, resp.).

Note that a property is determined by the 3-connected components if and only if it is determined by the 3-connected components with weight 1 of path type or of cut type. Thus we use "determined by the 3-connected components" in place of "determined by the 3-connected components with weight 1." Note also that if π is determined by the weighted 3-connected components, then π is determined by the 2-connected components and also satisfies the condition (C0), that is, π is invariant under 2-isomorphisms. Concerning properties determined by the weighted 3-connected components, we have the following.

LEMMA 1. *Let $c \geqq 1$. For a graph G, let $D(G)$ be the arbitrary decomposition of G into 3-connected components.*

(a) *Let π be determined by the 3-connected components with weight c of path type and satisfied by all graphs K_1, K_2, K_3 and C_4. Then the following* (a-1) *and* (a-2) *are equivalent for all simple graphs G.*
   (a-1) *For some graph G_i in $D(G)$, all $(G_i; c)\#$ violate π.*
   (a-2) *G violates π, that is, for some 3-connected component H of G, all $(H; c)\#$ violate π.*

(b) *Let π be determined by the 3-connected components with weight c of cut type and satisfied by all graphs K_1, C_1, B_3 and B_4. Then the following* (b-1) *and* (b-2) *are equivalent for all graph G without coloops or series-edges.*
   (b-1) *For some graph G_i in $D(G)$, all $(G_i; c)\#$ violate π.*

(b-2) *G violates π, that is, for some 3-connected component H of G, all $(H; c)\#$ violate π.*

Proof. For simplicity, we prove only (a). Part (b) can be obtained by dualizing the argument below.

Proof of (a). Since it is evident that (a-1) implies (a-2), we prove only that (a-2) implies (a-1). Without loss of generality we can assume that G is 2-connected. Suppose for every graph G_i in $D(G)$ some $(G_i; c)\#$ satisfies π. Let $F(G) = \mathscr{G} \cup \mathscr{P} \cup \mathscr{B}$ be the decomposition of G into 3-blocks, where $\mathscr{G}$ is the set of 3-connected graphs with four or more nodes in $F(G)$, $\mathscr{P}$ the set of polygons in $F(G)$ and $\mathscr{B}$ the set of bonds in $F(G)$. Since $F(G)$ is unique, any decomposition of G into 3-connected components is obtained from $F(G)$ by decomposing $\mathscr{P}$ and $\mathscr{B}$ appropriately. Thus, $\mathscr{G} \subset D(G)$. Furthermore, since π is determined by the weighted 3-connected components and satisfied by graphs K_3 and C_4, all cycles satisfy π. Thus, for every 3-connected component H with three or more nodes in G, some $(H; c)\#$ satisfies π. Let B_n be any bond in $\mathscr{B}$. Since G is simple and consequently B_n contains at most one edge of G, decompositions of B_n into triple bonds are unique to within isomorphisms preserving virtual edges, that is, if $D_1(B_n)$ and $D_2(B_n)$ are two decompositions into triple bonds then we can always find a bijection $f: D_1(B_n) \to D_2(B_n)$ such that, for any H_i in $D_1(B_n)$, there is an isomorphism $g: H_i \to f(H_i)$ with preserving virtual edges. (For a multigraph G, this is not true; see Fig. 6.) Thus, for every 3-connected component H of G, some $(H; c)\#$ satisfies π, and (a-2) implies (a-1). □

LEMMA 2. *Let $c \geqq 2$.*

(a) *Let π be satisfied by all graphs K_1, K_2, K_3 and C_4. Then, for simple graphs, π is determined by the 3-connected components with weight c of path type if and only if π is determined by the 3-connected components with weight 2 of path type.*

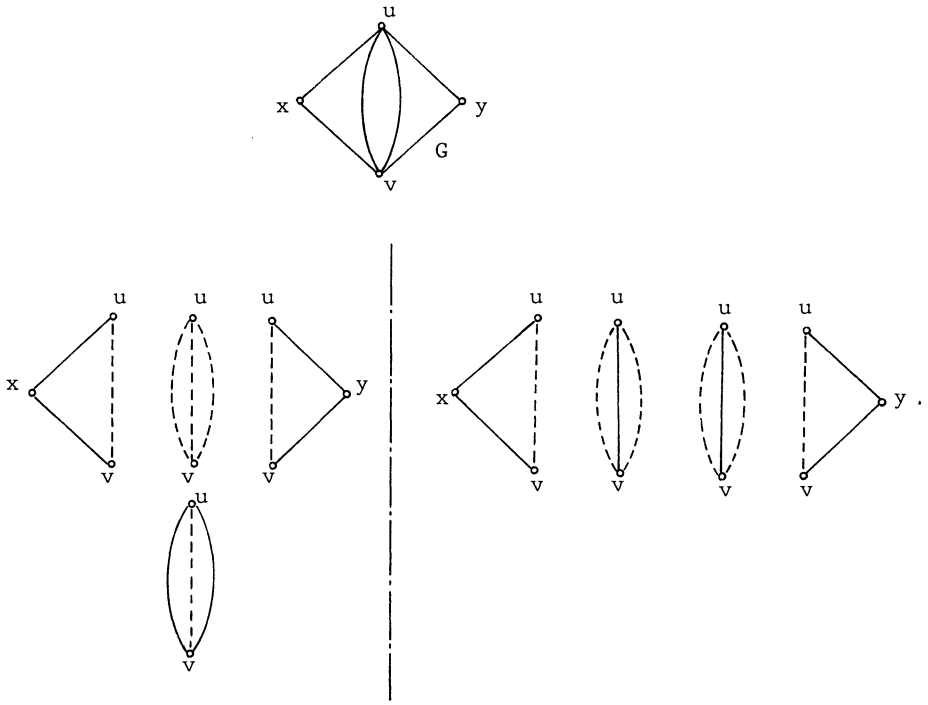


FIG. 6. *Graph G and two decompositions of G into 3-connected components.*

(b) *Let π be satisfied by all graphs K_1, C_1, B_3 and B_4. Then, for graphs without coloops or series-edges, π is determined by the 3-connected components with weight c of cut type if and only if π is determined by the 3-connected components with weight 2 of cut type.*

Proof. For simplicity, we prove only (a). Part (b) can be obtained by dualizing the arguments described below.

Proof of (a). We prove only the necessity. The sufficiency is obtained similarly. Assume that π is determined by the 3-connected components with weight c. Let G be an arbitrary simple graph.

If G violates π, then there is a 3-connected component G_i such that all $(G_i; c)\#$ violate π. Since π is satisfied by all graphs K_1, K_2, K_3 and C_4, G_i is a 3-connected graph with four or more nodes or $G_i = B_3$. We shall show that all $(G_i; 2)\#$ violate π. For an arbitrarily fixed graph $(G_i; 2)\#$, let $H \equiv (G_i; 2)\#$ have a weight function $c': E(G_i) \to Z^+$. That is, H is the graph obtained from G_i by replacing each edge e of G_i with a path of length $c'(e)$. By the definition, $c'(e) \geqq 2$ for all virtual edges e of G_i. Let H_i be a 3-connected component of H distinct from a triangle. Then H_i is isomorphic to G_i and all virtual edges of G_i are virtual in H_i. Thus, all $(H_i; c)\#$ violate π since all $(G_i; c)\#$ violate π. This implies that H violates π. Thus all $(G_i; 2)\#$ violate π.

Let G satisfy π. Suppose that, for some 3-connected component G_i of G, all $(G_i; 2)\#$ violate π. Then all $(G_i; c)\#$ violate π and consequently, G violates π, because π is determined by the 3-connected components with weight c. This is a contradiction. Thus, for any 3-connected component G_i of G, some $(G_i; 2)\#$ satisfies π.

Thus π is determined by the 3-connected components with weight 2. □

4. Edge-deletion problem for a property determined by the weighted 3-connected components. In this section we examine the computational complexity of edge-deletion problem $P_{\mathrm{ED}}(\pi)$ for π satisfying the following conditions (D1)-(D3).

(D1) π is nontrivial.

(D2) π is determined by the weighted 3-connected components.

(D3) If π is determined by the weighted 3-connected components of path type then π is satisfied by all graphs K_1, C_1, K_2, C_2, K_3 and C_4. If π is determined by the weighted 3-connected components of cut type then π is satisfied by all graphs K_1, C_1, K_2, C_2, B_3 and B_4.

Examples of such a property are "planar," "series-parallel," "outerplanar," etc.

THEOREM 1. *The edge-deletion problem $P_{\mathrm{ED}}(\pi)$ on simple 2-connected graphs is* NP*-hard if π satisfies the conditions* (D1)-(D3). *Furthermore, the restriction to simple 2-connected planar graphs is also* NP*-hard if, in addition, π is nontrivial on planar graphs, that is, π satisfies the condition* (C1), *as well as the conditions* (D1)-(D3).

Propositions 1 and 2 can be easily obtained from Theorem 1 as corollaries. To prove the theorem, we show that the following planar connected node cover problem P_{CNC}, which is known to be NP-complete [10], is polynomially transformable to $P_{\mathrm{ED}}(\pi)$.

Planar connected node cover problem P_{CNC}:

Instance: A connected simple planar graph G and a positive integer k.

Question: Is there a connected node cover N of size k or less for G?

(A subset N of $V(G)$ is a *connected node cover of size* $|N|$ if the subgraph of G induced by N is connected and $N \cap \{u, v\} \neq \emptyset$ for each edge (u, v) of G.)

Note that the idea and technique of transforming P_{CNC} to edge-deletion problems was first proposed in [26], and shown to be quite useful in proving the NP-hardness of edge-deletion and edge-contraction problems [5], [6]. Our proof is also based on

this idea and technique, but it employs duality (of matroids) and is therefore directly applicable to edge-contraction problem as well. In fact, the argument below can be regarded as the matroidal dual extension of the one in [6].

Proof of Theorem 1. If π is determined by the 3-connected components with weight 1, then it is determined by the 3-connected components with weight 2. Thus, we can fix $c=2$ in the condition (D2) by Lemma 2. For simplicity, we only consider that the conditions (D2) and (D3) are restricted to those of path type; that is, we assume as follows:

(D2) π is determined by the 3-connected components with weight 2 of path type.

(D3) π is satisfied by all graphs K_1, C_1, K_2, C_2, K_3 and C_4.

We can also discuss the other case similarly.

Let M be a simple planar graph with the minimum number of edges that violates π if π satisfies the condition (C1). Otherwise (i.e., if π satisfies the condition (D1) but not the condition (C1)) let M be a simple graph with the minimum number of edges that violates π. Let M' be a 3-connected component of M such that all $(M'; 2)$# violate π. Then M' is a 3-connected graph with four or more nodes or $M'=K_3^*(=B_3)$, since π satisfies the conditions (D1), (D2) and (D3). Note that M is the graph obtained from M' by replacing each virtual edge of M' with a path of length 2. If M' contains a virtual edge, then let e be any virtual edge of M'. Otherwise choose any edge e of M' and consider it as a virtual edge. Then the resulting graph also violates π because π satisfies the conditions (D2) and (D3). Thus, without loss of generality, we can assume that M' has a virtual edge and that e is a virtual edge of M'. Let L be the graph obtained from M' by replacing each virtual edge except e with a path of length 2. Let E' be the set of those edges of M which correspond to e. Then $L-e=M-E'$ (Fig. 7).

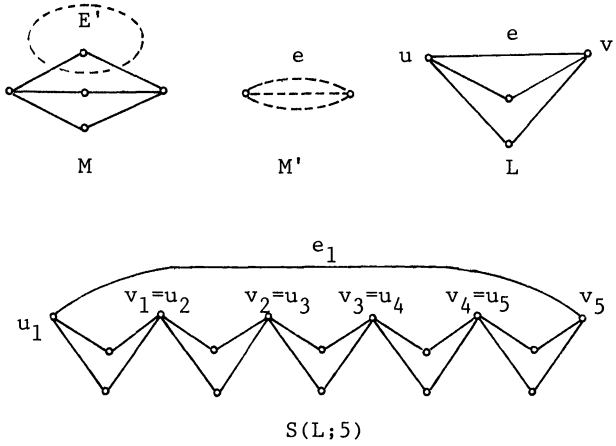


FIG. 7. *Graphs M, M', L and S(L; 5).*

Let $[G, k]$ be an instance of P_{CNC}, where G is a connected simple planar graph and k is a positive integer. We can assume, without loss of generality, that G is a simple 2-connected planar graph all of whose nodes are of degree $\geqq 3$ (see Appendix). Let $k'=m+k-1$ $(m=|E(G)|)$. Consider $k'+1$ disjoint copies $L_1, L_2, \cdots, L_{k'+1}$ of L. Let $e_i=(u_i, v_i)$ be the edge of L_i corresponding to the edge $e=(u, v)$ of L. Let $S(L; k'+1)$ be the graph obtained from graphs L_1-e_1, $L_2-e_2, \cdots, L_{k'+1}-e_{k'+1}$ by identifying each v_i $(i=1, 2, \cdots, k')$ with u_{i+1} and adding a new edge $(u_1, v_{k'+1})$ (Fig.

7). We also denote the new edge $(u_1, v_{k'+1})$ by e_1. Clearly, $S(L; k'+1)$ is planar if M is planar. The following observations are critical in this paper.

Fact 1. All subgraphs of $S(L; k'+1)-e_1$ satisfy π.

Fact 2. For any $S_d \subset E(S(L; k'+1))$ with $|S_d| \leqq k'$ and $e_1 \notin S_d$, if e_1 is not a coloop of the subgraph $S(L; k'+1)-S_d$ then $S(L; k'+1)-S_d$ violates π.

Since every 2-connected component of $S(L; k'+1)-e_1$ is $L-e=M-E'$, it satisfies π by the choice of M. Thus, every 2-connected component of a subgraph of $S(L; k'+1)-e_1$ has no more edges than $M-E'$ and consequently, it satisfies π. Thus, we have Fact 1 by the condition (D2). Since $|S_d| \leqq k'$ and e_1 is not a coloop of $S(L; k'+1)-S_d$, $S(L; k'+1)-S_d$ contains M' as a 3-connected component (more precisely, $S(L; k'+1)-S_d$ contains $M=(M'; 2)$ as a weighted 3-connected component). Thus, we have Fact 2, because all $(M'; 2)\#$ violate π.

Let $G(2)$ be the graph obtained from G by introducing a new node in the middle of every edge of G. Let $A(2)$ be the set of new nodes, i.e., $A(2)=V(G(2))-V(G)$. Embed the graph $G(2)$ in the plane by a linear time planar embedding algorithm [16]. For every pair of nodes a and a' in $A(2)$ which are adjacent to a common node in $V(G)$ and share a common face of $G(2)$, we add a new edge (a, a'). The resulting graph is denoted by G_2. Clearly, G_2 is planar. Let G_2^* be the dual graph of G_2. From now on, for a graph H with the dual graph H^*, we denote by x^* the dual-edge of H^* corresponding to the edge x of H. Note that the subgraph of G_2^* induced by the edges in $E(G(2))^*=\{x^* \in E(G_2^*) | x \in E(G(2))\}$ is a disjoint set of cycles (Fig. 8). For each edge $(a, a')^*$ of G_2^* $(a, a' \in A(2))$, we consider a (disjoint) copy of $S(L; k'+1)$ and

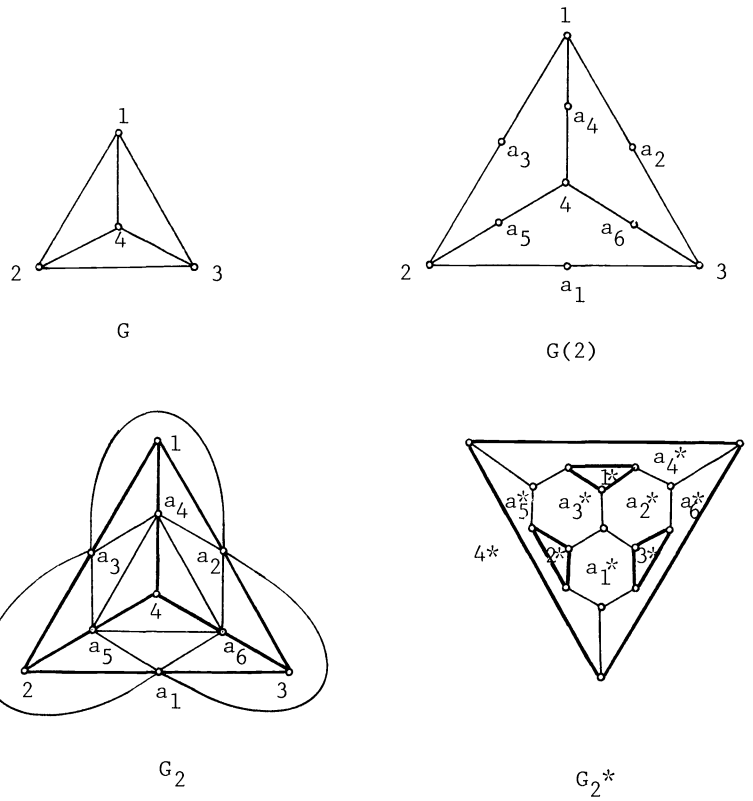


FIG. 8. *Graphs G, $G(2)$, G_2 and G_2^*, where v^* is the face of the dual graph G_2^* of G_2 corresponding to the node v of G_2.*

connect it through the copied edge of $e_1=(u_1, v_{k'+1})$ to the edge $(a, a')^*$ so that the copied edge of e_1 is in series to the edge $(a, a')^*$. Then delete all the copied edges of e_1. The resulting graph is denoted by G_3. Let G_4 be the graph obtained from G_3 by contracting all edges of the form $(a, a')^*$. For example, if $\pi=$"outerplanar," then the reduction looks like in Fig. 9. Note that G_4 is the graph obtained from G_2^* by identifying each edge of the form $(a, a')^*$ in G_2^* with the copied edge of e_1 of $S(L; k'+1)$ and then delete all edges of the form $(a, a')^*$. (G_3 is introduced to prove Proposition 3 below easily and to assure that the reduction is based on duality.)

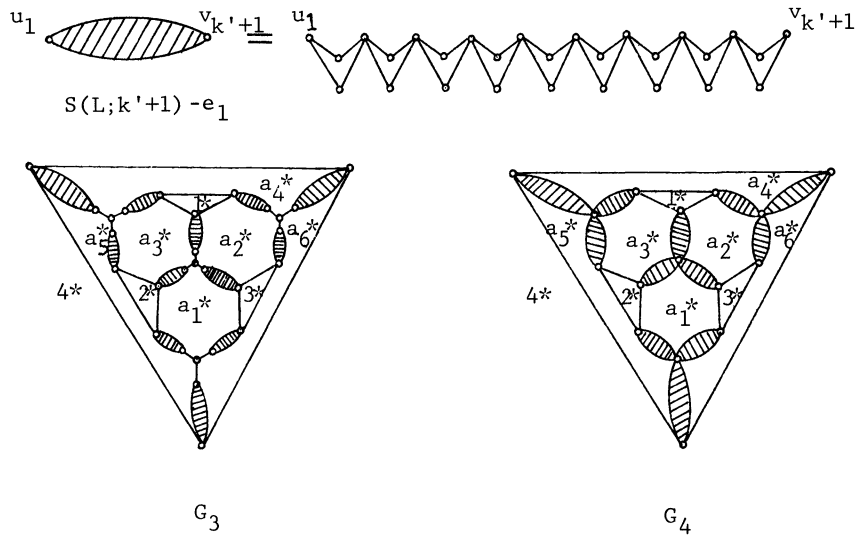


FIG. 9. *Graphs* $S(L; k'+1)-e_1$, G_3 *and* G_4 *in the case of* $G=K_4$, $M=K_{2,3}$ *and* $k=3$.

Note that, if M is planar, then G_4 and G_3 are planar. Since we have assumed that G is a 2-connected graph whose nodes are of degree $\geqq 3$ (Appendix), G_4 is a simple 2-connected graph. G_4 can be obtained from G in a polynomial order of $|V(G)|$ and $|E(G)|$, because $|V(M)|$ and $|E(M)|$ are independent of the input size of P_{CNC}. Thus, we have only to prove the following.

PROPOSITION 3. *There is a subset* S *of* $E(G_4)$ *with* $|S|\leqq k'$ *such that the subgraph* G_4-S *satisfies* π *if and only if* G *has a connected node cover of size* $\leqq k$, *or, equivalently, if and only if* $G(2)$ *has a Steiner tree for* $A(2)$ *of size* $\leqq k'$ [10]. (A subtree T that contains all nodes in A(2) is called a Steiner tree for $A(2)$ of size $|E(T)|$.)

For $X\subset E(G_2)$, let $X^*=\{x^*\in E(G_2^*)|x\in X\}$. Before proving the proposition, we should observe the following three points.

(I) For $S\subset E(G_4)$, if all edges of the form $(a, a')^*$ are coloops in the subgraph G_3-S, then G_4-S satisfies π.

(II) For $S\subset E(G_4)$ with $|S|\leqq k'$, if the subgraph G_4-S satisfies π, then all edges of the form $(a, a')^*$ are coloops in the subgraph G_3-S.

(III) For $S\subset E(G_4)$ with $|S|\leqq k'$, if the subgraph G_4-S satisfies π, then there is a subset S' of $E(G(2))^*$ with $|S'|\leqq k'$ such that the subgraph G_4-S' satisfies π.

If we have (I)-(III) above, then we can easily prove the proposition as follows.

Sufficiency of the proposition. Let $N\subset V(G)$ be a connected node cover of G of size $\leqq k$. Then, $G(2)$ has a Steiner tree T for $A(2)$ of size $\leqq k'$ ($|E(T)|\leqq k'$) [10]. Thus,

all nodes of $A(2)$ are contracted to a common node in the contraction $G_2/E(T)$ and all edges of the form (a, a') are loops. This implies that both in $G_2^* - E(T)^*$ and in $G_3 - E(T)^*$, all edges of the form $(a, a')^*$ are coloops. Thus, $G_4 - E(T)^*$ satisfies π by (I) (Fig. 10).

Necessity of the proposition. Let S be a subset of $E(G_4)$ with $|S| \leqq k'$ such that $G_4 - S$ satisfies π. Then by (II) and (III), we can assume that $S \subset E(G(2))^*$ and that all edges of the form $(a, a')^*$ are coloops both in $G_3 - S$ and in $G_2^* - S$. This implies that, in the dual graph G_2/X of $G_2^* - S$ ($S = X^*$), all edges of the form (a, a') are loops and all nodes of $A(2)$ are contracted to a common node. Thus, in the subgraph $G(2)|X$ of $G(2)$ induced by the edges in X, all nodes of $A(2)$ are contained in the same connected component. Let T be a spanning tree of this component. Then T is a Steiner tree of $G(2)$ for $A(2)$ of size $\leqq k'$. Thus, G has a connected node cover of size $\leqq k$ [10].

Now we return to proofs of (I)-(III). Points (I) and (II) are easily obtained from Facts 1 and 2. Point (III) is obtained from (I) and (II).

Proof of (I). Suppose all edges of the form $(a, a')^*$ are coloops in $G_3 - S$. $G_3 - S$ is divided into two subgraphs: one is a graph G' that is a disjoint union of subgraphs of copies of $S(L; k'+1) - e_1$; the other is a graph $G_2^{*\prime}$ that is a disjoint union of subgraphs of G_2^*. By Fact 1 and the condition (D2), G' satisfies π. The graph $G_2^{*\prime}$ also satisfies π because every 3-connected component of $G_2^{*\prime}$ is K_1, K_2 or K_3. Thus, $G_4 - S$ satisfies π by the condition (D2), since $G_4 - S$ is obtained from $G_3 - S$ by

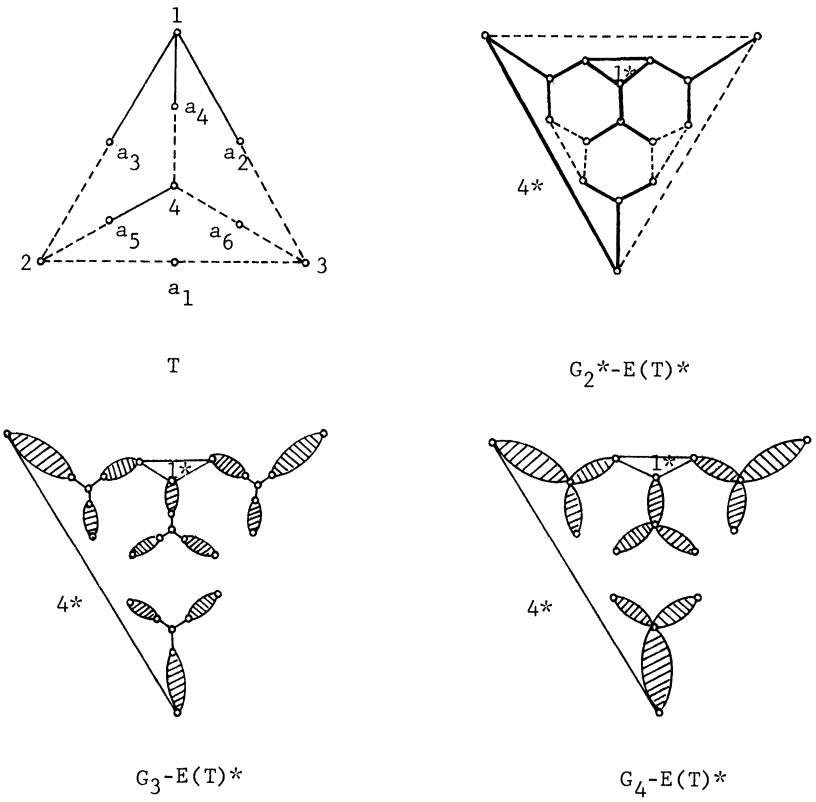


FIG. 10. *A Steiner tree T of $G(2)$ (dashed lines), the subgraph $G_2^* - E(T)^*$ of G_2^* (solid lines), the subgraph $G_3 - E(T)^*$ of G_3 and the subgraph $G_4 - E(T)^*$ of G_4.*

contracting all edges of the form $(a, a')^*$ which are coloops and 3-connected components in $G_3 - S$.

Proof of (II). Let S be a subset of $E(G_4)$ with $|S| \leqq k'$ such that $G_4 - S$ satisfies π. Suppose there is an edge of the form $(a, a')^*$ which is not a coloop in $G_3 - S$. Let H be the copy of $S(L; k'+1)$ attached to the edge $(a, a')^*$ of G_2^*. Let $S_d = S \cap E(H)$. Then $H - S_d$ is a split graph of $G_4 - S$. By Fact 2, $H - S_d$ violates π. This implies $G_4 - S$ violates π since π satisfies the condition (D2), a contradiction. Thus, we have (II).

Proof of (III). Let S be a subset of $E(G_4)$ with $|S| \leqq k'$ such that $G_4 - S$ satisfies π. By (II), all edges of the form $(a, a')^*$ are coloops in $G_3 - S$. If some copy of $L - e$ in $S(L; k'+1)$ contains exactly one edge x in S, then we can discard x; $G_4 - (S - x)$ also satisfies π by (I), because x is not a coloop in $L - e$ and all edges of the form $(a, a')^*$ are coloops in $G_3 - (S - x)$. If some copy of $L - e$ in a copy of $S(L; k'+1)$ contains two edges x, y in S, then defines S'' by

$$S'' = (S - \{x, y\}) \cup \{(b, v)^*, (b', v)^*\},$$

where b and b' are the two nodes in $A(2)$ such that this copy of $S(L; k'+1)$ is attached to $(b, b')^*$ and v is the node of G adjacent to both b and b' in G_2. Then, $|S''| \leqq k'$ and the subgraph $G_4 - S''$ also satisfies π by (I), because all edges of the form $(a, a')^*$ are coloops in $G_3 - S''$ (Fig. 11). Repeating this argument, we have (III).

Thus we have the theorem. □

As corollaries of Theorem 1, we have the following result as well as Propositions 1 and 2.

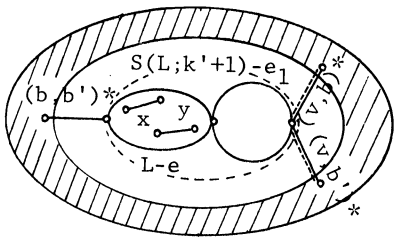


FIG. 11. *Graph* G_3.

COROLLARY 1. *The edge-deletion problem* $P_{ED}(\pi)$ *on simple* 2*-connected graphs is* NP*-complete for the following properties* π: (a) *planar,* (b) *without cocycles of cardinality* $\geqq 3$, (c) *series-parallel,* (d) *outerplanar,* (e) *co-outerplanar,* (f) *ladder. Furthermore, the restriction to simple* 2*-connected planar graphs for properties* (b)-(f) *is also* NP*-complete.*

The argument described in the proof of Theorem 1 has much more similarity to the one for the edge-contraction problem in [6], [26] than to the one for the edge-deletion problem in [5], [26], although it was for the edge-deletion problem. This is because the reduction is based on duality. Thus, we can examine the computational complexity of edge-deletion and edge-contraction problems almost simultaneously. Now, let us call the condition (C)* the *dual condition* of the condition (C) if, for π and its dual property π^*, π satisfies (C)* if and only if π^* satisfies (C). For example, (C1)* = (C1) and

(C3)* π is satisfied by all graphs K_1, C_1, K_2, C_2 and B_3.

Note that π is determined by the 3-connected components if and only if its dual property π^* is determined by the 3-connected components, because, for a planar graph G and its dual graph G^*, G_i is a 3-connected component of G if and only if the dual graph G_i^* of G_i is a 3-connected component of G^* [25] and G_i satisfies π if and only if G_i^*-satisfies π^*. Thus, (C2)* = (C2). Similarly we have (D2)* = (D2) and (D3)* = (D3). Thus, we have the following.

COROLLARY 2. *The edge-contraction problem $P_{EC}(\pi)$ on 2-connected graphs without coloops or series-edges is* NP*-complete if π satisfies the conditions* (D1)-(D3). *Furthermore, the restriction to 2-connected planar graphs without coloops or series-edges is also* NP*-hard if, in addition, π satisfies the condition* (C1).

COROLLARY 3. *The edge-contraction problem $P_{EC}(\pi)$ on 2-connected graphs without coloops or series-edges is* NP*-complete for the following properties π*: (a) *planar*, (b) *without cycles of length* $\geqq 3$, (c) *series-parallel*, (d) *outerplanar*, (e) *co-outerplanar*, (f) *ladder. Furthermore, the restriction to 2-connected planar graphs without coloops or series-edges for properties* (b)-(f) *is also* NP*-complete.*

5. Edge-deletion problems on 3-connected planar graphs. It is not difficult to understand that the NP-hardness proof of the edge-deletion problem in [26] is the NP-hardness proof of the weighted-edge-deletion problem on graphs without series-edges or parallel-edge, where the *weighted-edge-deletion problem* $P_{WED}(\pi)$ is as follows:

Instance: A graph G, a weight function $w: E(G) \to Z^+$ (the set of positive integers) and a positive integer k.

Question: Is there a subset $S \subset E(G)$ with $w(S) = \sum_{e \in S} w(e) \leqq k$ such that the subgraph $G - S$ satisfies π?

In this section, we show that the (simple) edge-deletion problem $P_{ED}(\pi)$ on 3-connected planar graphs is NP-hard if π satisfies the conditions (C1) and (C2) and the following conditions (C4) and (C5).

(C4) π is satisfied by the graphs K_3 and K_3^*.

(C5) π is hereditary on subgraphs.

"Series-parallel" is an example of such a property. We have already shown in a previous paper that the (simple) edge-deletion problem $P_{ED}(\pi)$ on 3-connected graphs (not restricted to planar graphs) is NP-hard if π satisfies some restricted conditions [5]. However, the proof was complicated. We shall give a simple proof using duality.

THEOREM 2. *The edge-deletion problem $P_{ED}(\pi)$ on 3-connected planar graphs is* NP*-hard if π satisfies the conditions* (C1), (C2), (C4) *and* (C5).

Proof. The reduction described here is almost the same as before.

Let $[G, k]$ be an instance of the planar connected node cover problem, where G is a simple 2-connected planar graph whose nodes are of degree $\geqq 3$ (Appendix) and k is a positive integer. Let $G(2)$, $A(2)$, k', G_2 and G_2^* be as in the proof of Theorem 1.

Let M be a planar graph of the minimum number of edges that violates π. Note that M always exists and it is a 3-connected planar graph with four or more nodes because π satisfies the conditions (C1), (C2) and (C4). Thus, M is a simple graph, every node of M^* is of degree $\geqq 3$, and every face of M consists of three or more edges. Let $e = (u, v)$ be an edge of M. Let $e_u = (u, u')$, $e_v = (v, v')$ be two distinct edges of M that share the common face with e (Fig. 12). Now consider $k'+1$ disjoint copies $M_1, M_2, \cdots, M_{k'+1}$ of M. Let $e_i = (u_i, v_i)$ be the edge of M_i corresponding to the edge $e = (u, v)$. Let $S'(M; k'+1)$ be the graph obtained from $M_1 - e_1, M_2 - e_2, \cdots, M_{k'+1} - e_{k'+1}$ by (i) identifying each v_i $(i = 1, 2, \cdots, k')$ with u_{i+1}; (ii) adding new vertices w_i and new edges (v_i, w_i), (v_i', w_i), (u_{i+1}', w_i) $(i = 1, 2, \cdots, k')$; (iii) deleting the edges

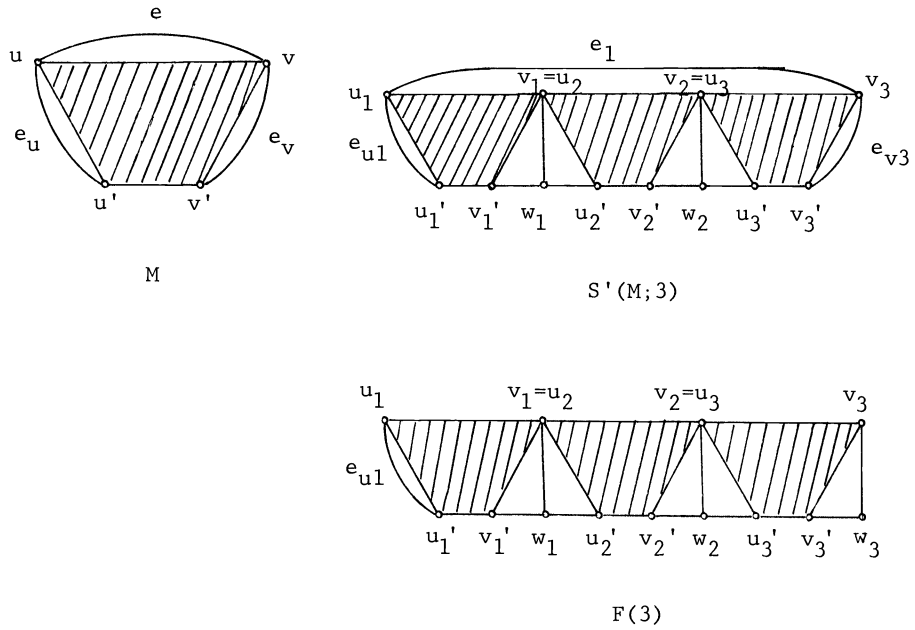


FIG. 12. *Edges e, $e_u=(u, u')$ and $e_v=(v, v')$ of a graph M and the graphs $S'(M; 3)$ and $F(3)$.*

$e_{vi}=(v_i, v_i')$, $e_{ui+1}=(u_{i+1}, u_{i+1}')$ $(i=1, 2, \cdots, k')$; (iv) adding a new edge $(u_1, v_{k'+1})$. The new edge $(u_1, v_{k'+1})$ is also denoted by e_1. Clearly, $S'(M; k'+1)$ is a 3-connected planar graph (Fig. 12). One can easily observe the following, which correspond to Facts 1 and 2.

Fact 3. All subgraphs of $S'(M; k'+1)-e_1$ satisfy π.

Fact 4. For any $S_d \subset E(S'(M; k'+1))$ with $|S_d| \leqq k'$ and $e_1 \notin S_d$, if e_1 is not a coloop of the subgraph $S'(M; k'+1)-S_d$, then $S'(M; k'+1)-S_d$ violates π.

Note that, in $S'(M; k'+1)-e_1$, $\{v_i, v_i'\}$, $\{v_i, w_i\}$ and $\{v_i, u_{i+1}'\}$ $(i=1, 2, \cdots, k')$ are all separation pairs. Thus, it is easy to see that every 3-connected component of $S'(M; k'+1)-e_1$ is isomorphic to a 3-connected component of $M-e$ or has three or fewer nodes. By the choice of M, this implies that $S'(M; k'+1)-e_1$ satisfies π, because π satisfies conditions (C2) and (C4). Thus, we have Fact 3 by the condition (C5). Since $|S_d| \leqq k'$, there is a copy $M-\{e, e_u, e_v\}$ in $S'(M; k'+1)$ such that S_d contains no edge of the copy. Thus, by appropriately deleting some edges of $S'(M; k'+1)-S_d$, we have a subgraph of $S'(M; k'+1)-S_d$ which contains M as a 3-connected component. This implies that the subgraph of $S'(M; k'+1)-S_d$ violates π. Thus, by the condition (C5), we have Fact 4.

Let $F(k'+1)$ be the graph obtained from $S'(M; k'+1)$ by adding a new node $w_{k'+1}$ and new edges $(w_{k'+1}, v_{k'+1})$ and $(w_{k'+1}, v_{k'+1}')$ and deleting $e_{vk'+1}=(v_{k'+1}, v_{k'+1}')$ (Fig. 12). Let G_5 be the graph obtained from G_2^* as follows: For each node f^* of G_2^* corresponding to the face f of G_2 that is bounded by only edges of the form (a, a') $(a, a' \in A(2))$, let $x_{f1}, x_{f2}, \cdots, x_{fs}$ be the adjacent nodes of f^* in counterclockwise order. Delete all edges (f^*, x_{fj}) $(j=1, 2, \cdots, s)$ and split f^* into $f_1, f_2, \cdots, f_s$. Consider disjoint copies $F_1, F_2, \cdots, F_s$ of $F(k'+1)$ and identify each u_1 of F_j with x_{fj} and $v_{k'+1}$ of F_j with f_j. Furthermore, identify $w_{k'+1}$ of F_{j-1} with f_j (Fig. 13). Then it is easy to

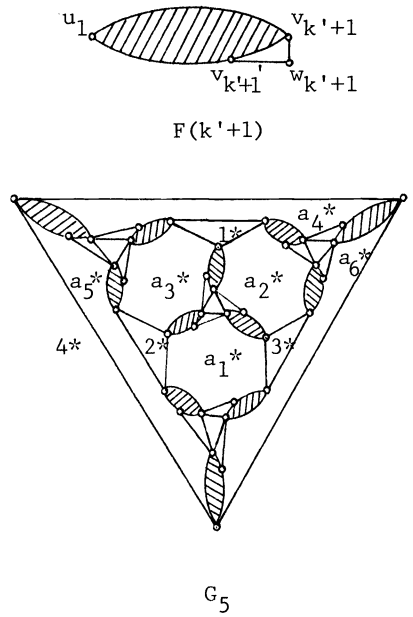


FIG. 13. *Graphs $F(k'+1)$ and G_5 in the case of $G = K_4$.*

observe that G_5 is a 3-connected planar graph. By the similar observation as in the proof of Proposition 3, we have the following proposition.

PROPOSITION 4. *There is a subset S of $E(G_5)$ with $|S| \leqq k'$ such that the subgraph $G_5 - S$ satisfies π if and only if $G(2)$ has a Steiner tree for $A(2)$ of size $\leqq k'$.* □

We leave a detailed proof for the readers. By duality and Theorem 2, we have the following corollary for the edge-contraction problem.

COROLLARY 4. *The edge-contraction problem $P_{EC}(\pi)$ on 3-connected planar graphs is* NP*-hard if π satisfies the following condition* (C5)* *as well as conditions* (C1), (C2) *and* (C4).

(C5)* *π is hereditary on contractions.*

6. Concluding remarks. We have shown that, if a property π is nontrivial on planar graphs, is determined by the weighted 3-connected components and is satisfied by all graphs K_1, C_1, K_2, C_2, K_3 and C_4, then the edge-deletion problem $P_{ED}(\pi)$ is NP-hard even if it is restricted to simple 2-connected planar graphs. Although π may be trivial on planar graphs, $P_{ED}(\pi)$ is NP-hard if π is nontrivial, is determined by the weighted 3-connected components and is satisfied by all graphs K_1, C_1, K_2, C_2, K_3 and C_4. Furthermore, if π is hereditary on subgraphs as well as is nontrivial on planar graphs, is determined by the 3-connected components and is satisfied by all graphs with three or fewer nodes, then $P_{ED}(\pi)$ is NP-hard even if it is restricted to 3-connected planar graphs. As a corollary, we have that the edge-deletion problem $P_{ED}(\pi)$ on simple 2-connected graphs is NP-complete for the following properties π: (a) planar, (b) series-parallel, (c) outerplanar, (d) co-outerplanar, (e) ladder, (f) without cocycle of cardinality $\geqq 3$. Furthermore, the restriction to simple 2-connected planar graphs for properties (b)-(f) is also NP-complete. The corresponding dual (edge-contraction) problem is also shown to be NP-complete by our reduction because it is based on duality.

For a fixed 3-connected graph H, if the subgraph homeomorphism problem for H is polynomially solvable, then, by Theorem 1, $P_{ED}(\pi)$ is NP-complete for the property $\pi =$ "not having a subgraph homeomorphic to H," because π is determined

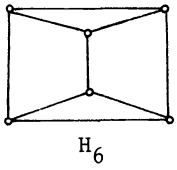


FIG. 14. *Graph H_6.*

by the 3-connected components [2]. Since the subgraph homeomorphism problem for the fixed graph $K_{3,3}$ is linear time computable [3], the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ is NP-complete for the property $\pi =$ "not having a subgraph homeomorphic to $K_{3,3}$." Similarly, $P_{\mathrm{ED}}(\pi)$ on planar graphs is NP-complete for $\pi =$ "not having subgraph homeomorphic to H_6" (Fig. 14) [3].

The reason why we did not consider the case when K_3 violates π in the proof of Theorem 1 is as follows: For a property π violated by K_3 and satisfied by all graphs with two or fewer nodes, if π is determined by the 3-connected components and hereditary on subgraphs, then a graph G satisfies π if and only if G has no cycle of length $\geqq 3$. Thus the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ is the problem of finding a (maximum weight) spanning tree of a graph. Thus, in this case, $P_{\mathrm{ED}}(\pi)$ is polynomially solvable.

The technique employed in this paper is also applied to matroid problems, and the edge-deletion problem $P_{\mathrm{ED}}(\pi)$ on binary matroids is obtained to be NP-complete for properties π such as $\pi =$ "regular," "graphic" and "cographic" [4].

Appendix.

THEOREM A. *The planar connected node cover problem P_{CNC} is* NP*-complete even if it is restricted to simple 2-connected planar graphs whose nodes are of degree* $\geqq 3$.

Proof. Let H be a connected simple planar graph. Embed H by a linear time planar embedding algorithm [16]. Let H have an articulation point v. Let H_1 and H_2 be two 2-connected components of H containing the node v such that edge (v, v_1) of H_1 and edge (v, v_2) of H_2 are in the same face of H (Fig. A1). Let H' be the graph obtained from H by adding new nodes x, y and z and new edges (x, v), (x, y), (x, z), (y, v), (y, z), (z, v_1) and (z, v_2) (Fig. A1). Then H' is planar. It is easy to observe that H has a connected node cover of size k if and only if H' has a connected node cover of size $k+2$. Neither H_1 nor H_2 is a 2-connected component of H'. Thus the number of 2-connected components of H' is $c-1$, where c is the number of 2-connected components of H. Repeating this argument, we can obtain a simple 2-connected planar

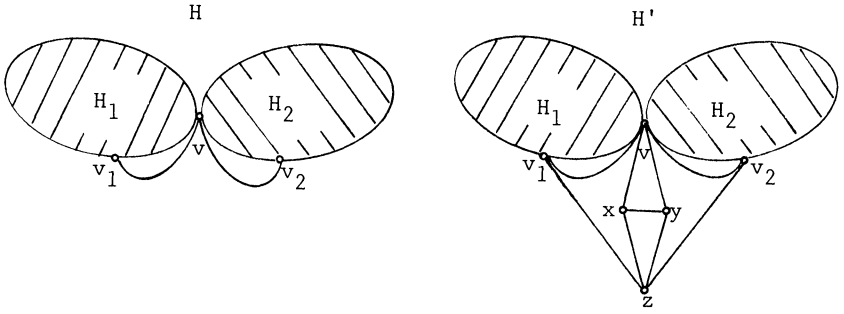


FIG. A1. *Graphs H and H'.*

graph H'' (in a polynomial order of the size of H) such that H has a connected node cover of size k if and only if H'' has a connected node cover of size $k+2c$.

Thus we can assume that H is a simple 2-connected planar graph. Let H have a node v of degree 2 and let (v, u) and (v, w) be two edges of H (Fig. A2). Let H' be the graph obtained from H by deleting the node v and adding new nodes x, y, z and t and new edges (u, x), (u, y), (x, y), (x, t), (y, z), (t, z), (t, w) and (z, w). Then H' is planar. It is easy to observe that H has a connected node cover of size k if and only if H' has a connected node cover of size $k+2$. Repeating this argument, we can obtain a simple 2-connected planar graph H'' whose nodes are of degree $\geqq 3$ such that H has a connected node cover of size k if and only if H'' has a connected node cover of size $k+2d$, where d is some constant.

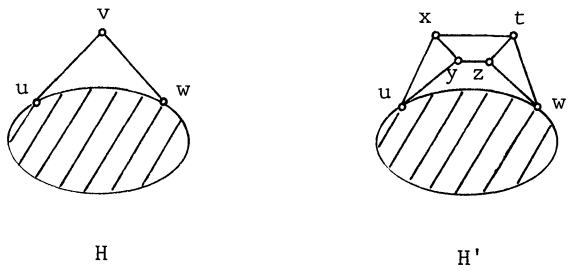


FIG. A2. *Graphs H and H'.*

Thus we have the theorem. (In [27], it is stated that P_{CNC} is NP-complete even if it is restricted to 3-connected planar graphs.) □

Acknowledgments. I would like to express my gratitude to Professor Masao Iri and Dr. Hiroshi Imai of the University of Tokyo for valuable discussions and suggestions. I also wish to thank the referees for their valuable comments for improving the presentation of this 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] T. ASANO, *Necessary and sufficient conditions for a property on graphs to be characterizable in terms of k-connected* ($k=1, 2, 3$) *forbidden graphs*, Trans. Inst. Electronics and Communication Engineers of Japan, E-66 (1983), pp. 666-670.

[3] ———, *An approach to the subgraph homeomorphism problem*, Theoret. Comput. Sci., 38 (1985), pp. 249-267.

[4] ———, *Edge-deletion and edge-contraction problems for matroids*, unpublished note.

[5] T. ASANO AND T. HIRATA, *Edge-deletion and edge-contraction problems*, Proc. 14th Annual ACM Symp. on Theory of Computing, San Francisco, CA, 1982, pp. 245-254.

[6] ———, *Edge-contraction problems*, J. Comput. System Sci., 26 (1983), pp. 197-208.

[7] T. H. BRYLAWSKI, *A combinatorial model for series-parallel networks*, Trans. Amer. Math. Soc., 154 (1971), pp. 1-22.

[8] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third Annual ACM Symp. on Theory of Computing, Shaker Heights, NY, 1971, pp. 151-158.

[9] J. EDMONDS AND E. L. JOHNSON, *Matching, Euler tours, and the Chinese postman*, Math. Programming, 5 (1973), pp. 88-124.

[10] M. R. GAREY AND D. S. JOHNSON, *The rectilinear Steiner tree problem is* NP-*complete*, SIAM J. Appl. Math., 32 (1977), pp. 826-834.

[11] ———, *Computers and Intractability: A Guide to the Theory of* NP*-completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
[12] M. R. GAREY, D. S. JOHNSON AND L. STOCKMEYER, *Some simplified* NP*-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237-267.
[13] F. O. HADLOCK, *Finding a maximum cut of a planar graph in polynomial time*, this Journal, 4 (1975), pp. 221-225.
[14] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
[15] J. E. HOPCROFT AND R. E. TARJAN, *Dividing a graph into triconnected components*, this Journal, 2 (1973), pp. 135-158.
[16] ———, *Efficient planarity testing*, J. Assoc. Comput. Mach., 21 (1974), pp. 549-568.
[17] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computation, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85-103.
[18] ———, *On the computational complexity of combinatorial problems*, Networks, 5 (1975), pp. 45-68.
[19] M. S. KRISHNAMOORTHY AND N. DEO, *Node-deletion* NP*-complete problems*, this Journal, 8 (1979), pp. 619-625.
[20] J. M. LEWIS, *On the complexity of the maximum subgraph problem*, Proc. 10th Annual ACM Symp. on Theory of Computing, San Diego, CA, 1978, pp. 265-274.
[21] J. M. LEWIS AND M. YANNAKAKIS, *The node-deletion problem for hereditary properties is* NP*-complete*, J. Comput. System Sci., 20 (1980), pp. 219-230.
[22] S. MACLANE, *A structural characterization of planar combinatorial graphs*, Duke Math. J., 3 (1937), pp. 460-472.
[23] G. I. ORLOVA AND Y. G. DORFMAN, *Finding the maximum cut in a graph*, Engrg. Cybernetics, 10 (1972), pp. 502-506.
[24] N. TOMIZAWA, *On a specialization sequence from general matroids to ladder graphs with special emphasis on the characterization of ladder matroids*, RAAG Research Notes, Third Series, No. 191, Research Assoc. of Applied Geometry, Tokyo, Japan, 1973.
[25] W. T. TUTTE, *Connectivity in Graphs*, Univ. of Toronto Press, Toronto, 1966.
[26] T. WATANABE, T. AE AND A. NAKAMURA, *On the removal of forbidden graphs by edge-deletion or edge-contraction*, Discrete Appl. Math., 3 (1981), pp. 151-153. (See also, *On the NP-hardness of edge-deletion and edge-contraction problems*, Discrete Appl. Math., 6 (1983), pp. 63-78.)
[27] T. WATANABE AND A. NAKAMURA, *The connected node cover problem for* 3*-connected graphs*, Technical Report AL81-79, Inst. Electronics and Communication Engineers of Japan, 1981 (in Japanese).
[28] D. J. A. WELSH, *Matroid Theory*, Academic Press, New York, 1976.
[29] H. WHITNEY, 2*-isomorphic graphs*, Amer. J. Math., 55 (1933), pp. 245-254.
[30] M. YANNAKAKIS, *Node- and edge-deletion* NP*-complete problems*, Proc. 10th Annual ACM Symp. on Theory of Computing, San Diego, CA, 1978, pp. 253-264.
[31] ———, *The effect of a connectivity requirement on the complexity of maximum subgraph problems*, J. Assoc. Comput. Mach., 26 (1979), pp. 618-630.
[32] ———, *Edge-deletion problems*, this Journal, 10 (1981), pp. 297-309.
[33] ———, *Node-deletion problems on bipartite graphs*, this Journal, 10 (1981), pp. 310-327.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
008

ON THE AXIOMATIZATION OF "IF-THEN-ELSE"*

IRÈNE GUESSARIAN† AND JOSÉ MESEGUER‡

Abstract. The equationally complete proof system for "if-then-else" of Bloom and Tindell (this Journal, 12 (1983), pp. 677–707) is extended to a complete proof system for many-sorted algebras with extra operations, predicates and equations among those. We give similar completeness results for continuous algebras and program schemes (infinite trees) by the methods of algebraic semantics. These extensions provide a purely equational proof system to prove properties of functional programs over user-definable data types.

Key words. if-then-else, complete proof system, equational logic, equational deduction, continuous algebras, algebraic semantics, program schemes

AMS(MOS) subject classifications. 68Q55, 68Q60, 68Q65, 03C05

1. Introduction. Algebraic Semantics (see [25] for an overview of current research directions) advocates the use of algebraic methods in (among other things) programming language design, programming methodology, and in proving properties of programs. The purpose of this paper is to give an equational axiomatization of "if-then-else" that can be used to prove properties of programs by purely equational methods. The context we have in mind is that of a functional programming language allowing user-defined, equationally axiomatized, abstract data types. Recursive programs that use the basic data type operations can then be defined in the language by sets of mutually recursive equations using if-then-else. For instance, having defined a binary associative and commutative operation *mult* on some (perhaps parameterized) data type, one could extend that operation to lists of elements of that data type by means of the recursive equation

$$\textit{map-mult}\ (L, L') = \textbf{if}\ L = \textit{nil}\ \textbf{then}\ \textit{nil}\ \textbf{else}$$
$$\textbf{if}\ L' = \textit{nil}\ \textbf{then}\ \textit{nil}\ \textbf{else}$$
$$(\textit{mult}\ (\textit{head}\ (L), \textit{head}(L'))\ \textit{map-mult}\ (\textit{tail}\ (L), \textit{tail}\ (L')))$$

where we have used juxtaposition to denote list concatenation and *head* and *tail* extract the first element and the rest of a list respectively. One would want to use the commutativity of *mult* and axioms for if-then-else (together with structural induction) in a proof of commutativity for *map-mult*. Automated deduction by equational methods is an area undergoing intense development (see for instance [28] and [15]); in our view, this makes the equational approach to proof systems for functional programs particularly promising.

1.1. The result of Bloom and Tindell. The class of if-then-else interpretations is not equationally definable. However, Bloom and Tindell [5] have shown that equality in that class can be axiomatized by a finite set of equations. We shall presently make

* Received by the editors April 27, 1984; accepted for publication (in revised form) June 20, 1986. This research was supported in part by a gift from the System Development Foundation, National Science Foundation grant MCS 8201380, and Office of Naval Research contracts N00014-80-0296 and N00014-82-C-033 and Agence de l'Informatique contract 83-695.

† Université Paris VII, UER Mathématiques, LITP, Laboratoire associé au CNRS LA 248, 75251 Paris, Cedex 05, France.

‡ Computer Science Laboratory, SRI International, Menlo Park, California 94025; and Center for the Study of Language and Information, Stanford University, Stanford, California 94305.

precise this, somewhat puzzling, situation. Given a set E of equations, let $\bar{E}$ denote the set of all equations provable from E by the rules of equational deduction. We shall write $E \vdash t = t'$ to denote the *provability relation*, i.e., iff $(t = t') \in \bar{E}$. The *satisfaction relation* $A \models t = t'$ holds of an algebra A and an equation $t = t'$ iff every assignment of values in A to the variables of t and t' makes both sides of the equation equal. In general, for $\mathscr{A}$ a class of algebras and E a set of equations we shall write $\mathscr{A} \models E$ iff $A \models t = t'$ for all A in $\mathscr{A}$ and all $t = t'$ in E. A class $\mathscr{A}$ of algebras on function symbols Σ is *equationally definable* iff there is a set E of equations such that $\mathscr{A} = \{A \in \mathrm{Alg}_\Sigma | A \models E\}$ where Alg_Σ denotes the class of all Σ-algebras; we then write $\mathscr{A} = \mathrm{Alg}_\Sigma^E$ to denote that $\mathscr{A}$ is defined by equations E. The usual algebraic structures such as monoids, groups, rings, etc. are equationally definable by the usual axioms. The following Completeness Theorem for equational logic is originally due to Birkhoff [3].

COMPLETENESS THEOREM.

$$E \vdash t = t' \quad \textit{iff } \mathrm{Alg}_\Sigma^E \models t = t'.$$

In addition, $E \vdash t = t'$ iff the terms t and t' are equal in the relatively free algebra on countably many generators associated to the equations E. Another fundamental theorem of equational logic is the model-theoretic characterization of equationally definable classes of algebras. It is also due to Birkhoff [3].

VARIETY THEOREM. *A class of algebras is equationally definable iff it is closed under products, subalgebras, and quotients.*

Viewing if-then-else as a ternary operation, $[-, -, -]$, we can consider the class $\mathscr{C}$ of all algebras where the operation $[x, y, z]$ is interpreted as an if-then-else, i.e.,

$$[x, y, z] = \begin{cases} z & \text{if } x = F, \\ y & \text{if } x = T, \\ \bot & \text{if } x = \bot \text{ or otherwise.} \end{cases}$$

It is easy to see that $\mathscr{C}$ is not equationally definable, since it is not closed under products. For instance, for a product of two such algebras we have

$$[(T, F), (x, x'), (y, y')] = (x, y')$$

so that such a product is outside $\mathscr{C}$.

Although $\mathscr{C}$ is not equationally definable, we can still ask whether equality in the class $\mathscr{C}$ can be effectively axiomatized. We make this precise; for $\mathscr{A}$ an arbitrary class of algebras we shall denote by $\equiv_{\mathscr{A}}$ the class of all equations satisfied by $\mathscr{A}$, i.e., $(\equiv_{\mathscr{A}}) = \{t = t' | \mathscr{A} \models t = t'\}$. We shall say that a set E of equations *axiomatizes equality* for the class $\mathscr{A}$, or that E is a (*complete*) *equational axiomatization* of equality in the class $\mathscr{A}$ iff $\bar{E} = (\equiv_{\mathscr{A}})$. Such an axiomatization E provides a *complete proof system* for proving the equalities semantically satisfied by the class $\mathscr{A}$ since by definition

$$\mathscr{A} \models t = t' \quad \text{iff } E \vdash t = t'.$$

Of course, in such a case $\mathscr{A} \subseteq \mathrm{Alg}_\Sigma^E$, but the containment may be strict. Bloom and Tindell have given a finite set of axioms Ax such that

$$\overline{Ax} = (\equiv_{\mathscr{C}}).$$

We have already seen that the containment $\mathscr{C} \subseteq \mathrm{Alg}_\Sigma^{Ax}$ is strict. Bloom and Tindell's elegant answer has been given in the simpler case (and some variants thereof, see the Appendix), when there are no additional operations or equations, just one sort of elements with a ternary operation $[-, -, -]$ and constants $T, F, \bot$. It would be nice to have such a completeness result available in the context of proving properties of functional programs, but this requires extending their axiomatization to that, more general, context.

It is the main purpose of this paper to provide such an extension. One should allow for several sorts of data besides the Booleans, as well as for data type operations and predicates for such data elements. Also, equations among the operations and predicates should be permitted. Recursive, possibly nonterminating, programs defined on such basic operations should also be included in the framework by further extending it to handle continuous algebras.

1.2. Summary of results for the unordered case. Considering the result of Bloom and Tindell one is led to ask the following questions:

(1) Does the approach extend to the case of additional sorts? (In particular, this would permit having a sort *Bool* different from the other data elements.)

(2) If so, can we throw in additional operations and predicates (i.e., operations going from the remaining data to *Bool*) subject to certain equational constraints?

This paper answers questions (1) and (2) in the affirmative. In fact, we first give a complete equational axiomatization of the *Boolean* if-then-else, i.e., the class consisting of (up to isomorphism) just one algebra with elements $T, F, \perp$ and with $[-,-,-]$ interpreted as if-then-else. We then give a similar complete axiomatization of the many-sorted case, where $[-,-,-]$ is "polymorphic." We then extend the result to a complete axiomatization of the case when there are additional operations on the non-Boolean data and predicates, together with arbitrary equations among those operations and predicates. Additional Boolean operations are not needed, because the Boolean if-then-else is by itself powerful enough to define the usual Boolean operations "and," "or," etc.

1.3. Summary of results for continuous algebras. Following ideas in Scott's Lattice of Flow Diagrams [27], algebraic semantics was extended to continuous algebras over a complete partial order by several authors [24], [9]. Again, free algebras are algebras of terms, but now terms can be infinite. A recursive equation defining a recursive program (more generally a family of simultaneous recursive equations) can then be "solved" or "unfolded' in such a continuous algebra as the infinite term which is the limit of its finite unfoldments. The *semantics* of such a program in a particular interpretation can then be obtained in a homomorphic way from its unfoldment, due to the initiality of the free algebra. As in classical universal algebra, there is a Completeness Theorem, and satisfaction of equations or inequations can be studied in an appropriate relatively free algebra. Assuming that the set E of equations (or more generally inequations of the form $t \leqq t'$) only involves finite terms, such a theorem was given by [4], [7], [11] and [23]. A more general completeness theorem for continuous theories where inequalities could be imposed over arbitrary terms was proved in [19]. For classes of continuous algebras defined by inequalities between finite terms, a Birkhoff variety theorem was proved in [21], and for the more general classes defined by continuous theories in [20]. In the case where all the inequalities in E involve only finite terms, the completeness theorem has a particularly nice form. Given a set E of inequations of the form $t \leqq t'$ for (finite) trees t, t', the rules of inequational deduction, together with the (in)equations in E as axioms, form a complete proof system for proving all valid inequations between finite trees. Moreover, the class of all continuous algebras that satisfy E is *algebraic*, i.e., the behavior of infinite trees (recursive programs) is characterized by the behaviour of their finite subtrees (results of finite computations); in other words, an inequation $T \leqq T'$ between two infinite trees T, T' is valid in such a class iff:

(*) for any finite computation t of T, there exists a finite computation t' of T', such that $t \leqq t'$ is valid.

(*) is a very simple induction rule, which, together with the axioms of E and the rules of inequational logic (i.e., reflexivity, antisymmetry, transitivity, full invariance), forms a complete proof system. Of special interest is the case where the set E of axioms is finite.

We can now reconsider the situation described at the beginning of this introduction, of a functional language with user-definable, equationally axiomatized, abstract data types in which (possibly nonterminating) programs are defined by recursive equations. To prove properties about such programs we can use the equations defining the data types (plus structural induction principles whenever the data types are initial) together with axioms for the if-then-else and the induction principle (*). The generalization of Bloom and Tindell's approach to this context seeks completeness results for the if-then-else axioms and thus attempts answering questions such as the following:

(3) Can the positive answer to (2) be extended to continuous algebras (again, with additional operations, predicates, and equations among those) providing a complete proof system for finite terms?

(4) If so, can such a completeness result be additionally extended to proofs of equality $T = T'$ (resp. inequality $T \leqq T'$) between two infinite terms by means of the induction rule (*)? (This would provide a complete proof system for program schemes with if-then-else.)

We answer question (3) in the affirmative, and present a partial answer to question (4), namely for equality between infinite terms when there are no additional operations. We also show that (4) can be answered in the affirmative in full generality provided that one has an effective axiomatization by inequations of the inequalities between finite terms satisfied by the continuous if-then-else interpretations; in other words, we show that the if-then-else interpretations form an algebraic class. We end the paper with a discussion of related work by other authors.

2. Axiomatizing if-then-else.

2.1. The one-sorted axiomatization of Bloom and Tindell. For Σ a signature, possibly many sorted, let Alg_Σ denote the category of Σ-algebras. Let A be an algebra in Alg_Σ, X a set of variables, and $\alpha : X \to A$ a valuation. Let $T_\Sigma(X)$ be the free Σ-algebra over generators X; we denote by $\bar{\alpha}$ the unique homomorphism $T_\Sigma(X) \to A$ extending α. For t, t' in $T_\Sigma(X)$, let e be the equation $t = t'$, we say that A *satisfies* e, and write $A \models e$ (or $A \models t = t'$) if and only if $\bar{\alpha}(t) = \bar{\alpha}(t')$, for every valuation α. If E is a set of equations, we say that A *satisfies* E, and write $A \models E$, iff for all e in E, $A \models e$. We will assume that all carriers of all sorts of Σ-algebras are nonempty, since they will all contain an element $\perp$, hence the definition of satisfaction does not depend on X [8]. For t in $T_\Sigma(X)$ and A an algebra in Alg_Σ, we denote by t_A the mapping $A^n \to A$ defined by: for each valuation $\alpha : \{x_1, \cdots, x_n\} \to A$, $t_A(\alpha(x_1), \cdots, \alpha(x_n)) = \bar{\alpha}(t)$.

In the rest of this section, we will consider the following notion of if-then-else: let $T, F, \perp$ be constants and $[-,-,-]$ a ternary operation. Let $\Sigma = \{T, F, \perp, [-,-,-]\}$ be a one sorted signature, and let $\mathscr{C}$ be the following class of algebras:

$\mathscr{C} = \{A \text{ in } \mathrm{Alg}_\Sigma \mid [-,-,-]: A^3 \to A \text{ satisfies (i)}\}$ where

(i)
$$[x, y, z] = \begin{cases} z & \text{if } x = F, \\ y & \text{if } x = T, \\ \perp & \text{if } x = \perp \text{ or otherwise.} \end{cases}$$

We abbreviate (i) by saying that $[-,-,-]$ is an if-then-else.

For a class $\mathscr{A}$ of Σ-algebras, define the following congruence $\equiv_{\mathscr{A}}$ on $T_\Sigma(X)$: $t \equiv_{\mathscr{A}} t'$ iff for all A in $\mathscr{A}$, $A \models t = t'$.

For properties of $\equiv_{\mathscr{A}}$ see [13].

Note that the above defined class $\mathscr{C}$ is not a variety, namely is not of the form:

$$\mathrm{Alg}_\Sigma^E = \{A \text{ in } \mathrm{Alg}_\Sigma^E | A \text{ satisfies all equations } e \text{ in } E\}$$

for some set E of equations. This follows from Birkhoff's Variety Theorem (cf. Introduction). However, Bloom and Tindell [5], gave a complete set of equational axioms from which we can deduce all the equations in $\equiv_{\mathscr{C}}$, via equational deduction only.

Let Ax be the following axioms ($Ax(3; \perp)$ in [5]).

(1) $[T, x, y] = x$,
(2) $[x, x, y] = [x, T, y]$,
(3) $[F, x, y] = y$,
(4) $[x, y, x] = [x, y, F]$,
(5) $[\perp, x, y] = \perp$,
(6) $[x, \perp, \perp] = \perp$,
(7) $[x, [y, z, u], [y, v, w]] = [y, [x, z, v], [x, u, w]]$ "premise interchange" axiom,
(8) $[x, [x, y, z], w] = [x, y, w]$ "redundancy" axiom,
(9) $[x, y, [x, z, w]] = [x, y, w]$ "redundancy" axiom,
(10) $[[x, y, z], u, v] = [x, [y, u, v], [z, u, v]]$ "premise simplification" axiom.

We can define:

$$\mathrm{Alg}_\Sigma^{Ax} = \{A \text{ in } \mathrm{Alg}_\Sigma | A \text{ satisfies } Ax\}.$$

For E a set of Σ-equations, $\bar{E}$ denotes the closure of E under the rules of equational deduction.[1] By the completeness theorem for equational logic, if Alg_Σ^E denotes the class of Σ-algebras that satisfy E, one has

$$\equiv_{\mathrm{Alg}_\Sigma^E} = \bar{E}.$$

THEOREM 1 [5]. $\equiv_{\mathscr{C}} = \overline{Ax}$. □

In model-theoretic terms, this means that the least variety containing $\mathscr{C}$ is Alg_Σ^{Ax}. In proof-theoretic terms, this means that the equational theory of $\mathscr{C}$ is the same as $\overline{Ax}$. Note that the inclusion $\mathscr{C} \subseteq \mathrm{Alg}_\Sigma^{Ax}$ is strict.

To make the result of Bloom and Tindell fully usable, one might like to allow other operations besides the if-then-else and might like to distinguish data of various sorts among themselves and also from the Boolean values. Thus, there are several directions in which we will generalize the theorem [5], namely:

- allowing for many sorts (§ 2);
- allowing for other operations and possibly equations holding among them (§ 3);
- allowing for an ordering and continuous algebras (§ 4).

2.2. A complete axiomatization of if-then-else in Boolean algebras with $\perp$. Let $\mathscr{C}_b$ be the following class of algebras:

$$\mathscr{C}_b = \{A/A \text{ is a } \Sigma - \textit{algebra} \text{ with domain } \{\perp, T, F\} \text{ with the standard interpretation, and } [-,-,-]: A^3 \to A \text{ is an } \textit{if-then-else}\}.$$

Notice that $\mathscr{C}_b \subset \mathscr{C}$. Then Ax can be extended to a set Ax_b of equational axioms such that for any trees t, t' in $T_\Sigma(V)$:

$$\mathscr{C}_b \models t = t' \quad \text{iff } Ax_b \vdash t = t'.$$

The same would hold also for infinite trees T, T' in $CT_\Sigma(V)$.

[1] That is, reflexivity, symmetry, transitivity, compatibility with the operations, and full invariance. This set of rules is sound and complete also for the many-sorted case because the carriers of the various sorts are nonempty [8].

Preliminary notation (from [5]). Let t be a tree in $T_\Sigma(V)$, n a node in the domain of t (n in $\{1,2,3\}^*$). $t(n)$ is the label of t at node n. The tree t is said to be *simple premise* iff whenever $t(n)$ is an if-then-else then $t(n1)$ is in V (i.e., premises of conditionals are labeled by variables in t). By axioms (1), (3), (5) and (10) any tree is Ax-provably equal to a simple premise tree. Let t be simple premise, and $\alpha: V \to A$ a valuation, then path (α, t) in $\{2,3\}^*\{\perp\}^*$ is defined by:

if t is in $V \cup \{T, F, \perp\}$ then path $(\alpha, t) = \varepsilon$ (empty word)
otherwise, $t = [v, t_2, t_3]$ and then

$$\text{path}(\alpha, t) = \begin{cases} 2\,\text{path}(\alpha, t_2) & \text{if } \alpha(v) = T, \\ 3\,\text{path}(\alpha, t_3) & \text{if } \alpha(v) = F, \\ \perp & \text{otherwise.} \end{cases}$$

Let t be a simple premise tree and n a nonpremise node in t (n is in $\{2,3\}^*$). n is *accessible* iff n is an initial segment of some path (α, t) for some valuation α. t is said to be *accessible* iff all nonpremise nodes are accessible. Every tree is Ax-provably equal to an accessible tree.

An accessible tree t is *reduced* iff for every subtree $[v, t_2, t_3]$ of t, v is not the label of any leaf in t_2 or t_3. Every tree is Ax-provably equal to a reduced tree.

Add now to Ax the following

(11) $\quad x = [x, T, F]$

which is obviously valid in $\mathscr{C}_b$, though not in $\mathscr{C}$. Let $Ax_b = Ax \cup (11)$. A reduced tree t is in *normal form* iff all nonpremise leaves of t are labeled by T, F or $\perp$.

LEMMA 2. *Every t is Ax_b-provably equal to a tree in normal form.*

Proof. First reduce t, then substitute each nonpremise leaf labeled by a variable v in V by $[v, T, F]$. The resulting tree is still reduced and moreover is in normal form. □

LEMMA 3. *Let t be in normal form, A an algebra in $\mathscr{C}$, α a valuation $V \to A$, and $n = \text{path}(\alpha, t)$. Let $\bar{\alpha}$ also denote the unique homomorphism $T_\Sigma(V) \to A$ extending α. Then $\bar{\alpha}(t) = \alpha(t(n))$ if n is in $\{2,3\}^*$ and $\bar{\alpha}(t) = \perp$ if n is in $\{2,3\}^*\perp$.*

Proof. By induction on the depth $d(t)$ of t. The result is obvious if $d(t) = 1$; otherwise $t = [v, p, q]$ with p, q in $\{\perp, T, F\}$; then by case analysis $t = [v, t_2, t_3]$ and:

either $\alpha(v) = T$ and $\bar{\alpha}(t) = \bar{\alpha}(t_2) = \alpha(t_2(\text{path}(\alpha, t_2)))$ (by induction)

$$= \alpha(t(\text{path}(\alpha(T))) \quad \text{(by definition of path } (\alpha, t))$$

or $\alpha(v) = F$ and similarly $\bar{\alpha}(t) = \alpha(t(n))$

or lastly $\bar{\alpha}(t) = \perp = \text{path}(\alpha, t)$. □

LEMMA 4. *Let $\alpha: V \to I$ with I in $\mathscr{C}$; there exists $\alpha': V \to I_b$, I_b in $\mathscr{C}_b$ (hence having domain $\{\perp, T, F\}$) such that for all t in normal form: $\bar{\alpha}(t) = \bar{\alpha}'(t)$.*

Proof. Define $\alpha'(v) = \alpha(v)$ if $\alpha(v)$ is in $\{T, F\}$, $\perp$ otherwise. By definition of path, path $(\alpha, t) = \text{path}(\alpha', t)$; hence by Lemma 3 $\bar{\alpha}(t) = \bar{\alpha}'(t)$. □

THEOREM 5. *For t, t' in normal form,*

$$\mathscr{C} \models t = t' \Leftrightarrow \mathscr{C}_b \models t = t'.$$

Proof. ($\Rightarrow$) is clear since $\mathscr{C}_b \subseteq \mathscr{C}$. For ($\Rightarrow$) assume $\mathscr{C} \not\models t = t'$, hence for some I in $\mathscr{C}$ $I \not\models t = t'$ i.e., there exist I in $\mathscr{C}$, $\alpha: V \to I$ such that $\bar{\alpha}(t) \neq \bar{\alpha}(t')$. Let $\alpha': V \to I_b (= \{\perp, T, F\})$ be as in Lemma 4, then $\bar{\alpha}'(t) \neq \bar{\alpha}'(t')$ hence $I_b \not\models t = t'$ and $\mathscr{C}_b \not\models t = t'$. □

COROLLARY 1. (i) *for t, t' in normal form*: $\mathscr{C}_b \models t = t' \Leftrightarrow Ax \vdash t = t'$; (ii) *for arbitrary* t, t' $\mathscr{C}_b \models t = t' \Leftrightarrow Ax_b \models t = t'$.

Proof. Item (i) results from [5] and the previous theorem. Item (ii) is (i) together with Lemma 2. □

COROLLARY 2.

$$\mathscr{C}_b \models t = t' \Leftrightarrow \forall \alpha : V \to I_b = \{\perp, T, F\}, \qquad \bar{\alpha}(t) = \bar{\alpha}(t')$$
$$\Leftrightarrow Ax_b \vdash t = t'.$$

□

Remark. $Ax_b \vdash [x, x, x] = x$.

2.3. Many-sorted case. Assume now that the algebras we consider have two sorts: a sort *Bool* and a sort *Data* (the case with any finite number of sorts would be similar). Let $\mathscr{C}_d$ be the class of 2-sorted Σ_d-algebras, where

$$\Sigma_d = \{\perp, T, F, \perp_d, [-,-,-]_b, [-,-,-]_d\}$$

$[-,-,-]_b$ is an *if-then-else* of sort *Bool* × *Bool* × *Bool* → *Bool*, $\perp_d$ is the undefined constant of sort Data and $[-,-,-]_d$ is an if-then-else of sort *Bool* × *Data* × *Data* → *Data*, and such that $A_{|\Sigma}$ is in $\mathscr{C}_b$ (see Fig. 1). Note that this implies that the only elements of sort *Bool* in A are T, F and $\perp$.

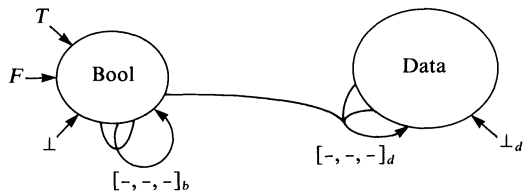


FIG. 1

We assume that $[-,-,-]$ is a shorthand representing *if-then-else* of both sorts. This creates no ambiguity because we require that terms be well formed with respect to sorts. Let $Ax_d = Ax_b \cup Ax_2$, where Ax_2 denotes the axioms (1), (3) and (5)-(10) for the sort *Data*, namely for x, y, z of sort *Bool*, t, u, v, w of sort *Data*:

(1′) $[T, u, v] = u$,
(3′) $[F, u, v] = v$,
(5′) $[\perp, u, v] = \perp$,
(6′) $[x, \perp, \perp] = \perp$,
(7′) $[x, [y, t, u], [y, v, w]] = [y, [x, t, v], [x, u, w]]$ "premise interchange" axiom,
(8′) $[x, [x, u, v], w] = [x, u, w]$ "redundancy" axiom,
(9′) $[x, u, [x, v, w]] = [x, u, w]$ "redundancy" axiom,
(10′) $[[x, y, z], u, v] = [x, [y, u, v], [z, u, v]]$ "premise simplification" axiom.

THEOREM 6 (Completeness).

$$\mathscr{C}_d \models t = t' \quad \Leftrightarrow \quad Ax_d \vdash t = t'.$$

Proof. The soundness part (⇐) is obvious. The completeness part (⇒) splits in 2 parts: for terms of sort *Bool* and for terms of sort *Data*.

(a) Terms of sort *Bool*: The mapping $\mathscr{C}_d \to \mathscr{C}_b$ forgetting the sort *Data* is surjective; to any A in $\mathscr{C}_b$ associate $A' = A \cup \{\perp_d\}$, i.e., the algebra where the only term of sort *Data* is $\perp$. Hence, for t, t' of sort *Bool*,

$$\mathscr{C}_d \models t = t' \Rightarrow \mathscr{C}_b \models t = t' \Rightarrow Ax_b \vdash t = t' \Rightarrow Ax_d \vdash t = t'.$$

(b) Terms of sort *Data*: As in [5] every term of sort *Data* is Ax_d-provably equal to a simple premise reduced tree (the meaning is the same as in the Boolean case). The proof is the same as in [5] since this proof is a purely syntactic one showing that any tree is Ax-provably equal to a reduced t', and all the axioms in Ax are in Ax_d (up to a well-formedness constraint with respect to sorts).

Recall from [5] the definition of the function $S(n, t)$, with $(n, t) \in \{2, 3\}^* \times T_{\Sigma_d}(V)$:

$$S(\varepsilon, t) = \varnothing \quad \text{and for } t = [p, t_2, t_3],$$

$$S(2n, t) = \{p\} \cup S(n, t_2) \quad \text{and} \quad S(3n, t) = \{\bar{p}\} \cup S(n, t_3).$$

Intuitively, $S(n, t)$ represents the set of Boolean premises which lead to node n in t.

INSERTION LEMMA 7 [5]. *Let n be a nonpremise node of the simple premise tree t of sort Data and let p be in V_b. If p (resp. $\bar{p}$) is in $S(n, t)$, then $Ax_2 \vdash t = t(n \leftarrow [p, t|n, a])$ (resp. $Ax_2 \vdash t = t(n \leftarrow [p, a, t|n])$), where a is a new variable in V_d which does not occur in t, $t|n$ denotes the subtree of t at node n and $t(n \leftarrow t')$ denotes the tree deduced from t by substituting tree t' at node n.*

The proof is a purely syntactic one involving only the axioms (7) to (10), hence it translates immediately by just taking terms of sort *Data* and axioms Ax_2.

PROPOSITION 8. *Let $t = [p, t_2, t_3]$ and $t' = [q, t'_2, t'_3]$ be reduced trees of sort Data such that $\mathscr{C}_d \models t = t'$ and $p \neq q$; then there exists $t'' = [p, t''_2, t''_3]$ such that:*

(i) $Ax_2 \vdash t' = t''$,

(ii) $d(t'') \leqq d(t') + 1$.

Proof. We first prove that either p or $\bar{p}$ is in $S(n, t'_i)$ for any nonpremise leaf n of t'_i, for $i = 2, 3$. Let n be a nonpremise leaf in e.g. t'_2 such that neither p nor $\bar{p}$ is in $S(n, t'_2)$. Let $z = t'_2(n)$.

- *Case* $z = \bot$. Then $t_1 = [q, t'_2(n \leftarrow [p, \bot, \bot]), t'_3]$ is reduced and Ax_2-provably equal to t', and has one less nonpremise leaf such that neither p nor $\bar{p}$ are in $S(n, t'_2)$. This transformation can take care of all leaves of t' labeled by $\bot$, while increasing the depth $d(t')$ by at most one.
- *Case* $z \neq \bot$. Since z is in V_d and t' is reduced, q does not occur at all in t'_2; moreover, we assumed that neither p nor $\bar{p}$ occur in $S(n, t'_2)$, hence we have complete freedom to choose an assignment

$$\alpha : V = V_b \cup V_d \to J = \{\bot, T, F\} \cup \{\bot, a\}$$

such that: $\alpha(p) = \bot$, $\alpha(q) = T$, path $(\alpha, t'_2) = n$, $\alpha(z) = a$ and: $\bar{\alpha}(t) = \bot$, $\bar{\alpha}(t') = a$, a contradiction. (Notice that V_b denotes the variables of sort *Bool* and V_d denotes the variables of sort *Data*.) By the Insertion Lemma above we can replace each such leaf by a tree of the form $[p, z, z]$. This transformation can take care of all leaves of t' not labeled by $\bot$, while increasing the depth $d(t')$ by at most one; this increase in depth occurs only in leaf nodes not labeled $\bot$, hence unaffected by the previous transformation.

So globally, the depth of t' increases at most by one. By applying then the interchange axiom (7′):

$$Ax_2 \vdash t' = t'' = [p, t''_2, t''_3] \quad \text{with } d(t'') \leqq d(t') + 1. \qquad \square$$

Proof of Completeness Theorem 6. Let t, t' be of sort *Data* such that:

$$\mathscr{C}_d \models t = t' \quad \text{then } Ax \vdash t = t'.$$

We may assume t, t' reduced and will proceed by induction on $d(t) + d(t')$ where $d(t)$ is the depth of t.

(a) $d(t) = 1$ and $d(t') \geqq 1$, $t = x$ or $t = \bot$.

(a.1) $t = x \Rightarrow t' = x$, otherwise $t' = z$ or $t' = [p, t_2, t_3]$ in either case, there exists $\alpha : V = V_b \cup V_d \to J = \{\bot, T, F\} \cup \{\bot, a\}$ such that $\bar{\alpha}(t) = \alpha \neq \bar{\alpha}(t') = \bot$.

(a.2) $t = \bot$ if $d(t') = 1$ then $t' = \bot$;

otherwise, $d(t') \neq 1$ and $t' = [p, t_2, t_3]$;

then for $i = 2, 3$ $Ax \vdash t_i = \bot$;
if for some i, e.g., $i = 2$: $\mathscr{C}_d \not\models t_2 = \bot$, then taking $\alpha(p) = T$ we can conclude $\mathscr{C}_d \not\models t' = \bot$, a contradiction. By induction then,

$$Ax \vdash t_i = \bot \quad \text{for } i = 2, 3 \text{ whence } Ax \vdash t' = [p, \bot, \bot] = \bot.$$

(b) $d(t) \neq 1$ and $d(t') \neq 1$ then $t = [p, t_2, t_3]$ and $t' = [q, t'_2, t'_3]$.

(b.1) $p = q$. Let V_b denote the set of variables of sort *Bool* in $t \cup t'$ and let V_d denote the set of variables of sort *Data* in $t \cup t'$. Let

$$\alpha : (V_b - \{p\}) \cup V_d \to A \quad \text{in } \mathscr{C}_d$$

be an arbitrary valuation. Extend

$$\alpha \text{ to } \alpha' : V_b \cup V_d \to A \quad \text{by } \alpha'(p) = T.$$

Then

$$\begin{aligned} \bar{\alpha}(t_2) &= \bar{\alpha}'(t_2) && [t \text{ is reduced hence there are no } p\text{'s in } t_2] \\ &= \bar{\alpha}'(t) = \bar{\alpha}'(t') && [\text{since } \mathscr{C}_d \models t = t'] \\ &= \bar{\alpha}'(t'_2) = \bar{\alpha}(t'_2) && [t' \text{ is reduced hence there are no } p\text{'s in } t'_2]. \end{aligned}$$

Hence $\mathscr{C}_d \models t_2 = t'_2$; similarly $\mathscr{C}_d \models t_3 = t'_3$, whence the induction step in this case since $d(t) + d(t')$ is replaced by $d(t_i) + d(t'_i)$, $i = 2, 3$.

(b.2) $p \neq q$. By Proposition 8,

$$Ax_2 \vdash t' = t'' = [p, t''_2, t''_3] \quad \text{with } d(t'') \leqq d(t') + 1.$$

So, we are back to the case $p = q$, with $d(t) + d(t')$ replaced by $d(t_i) + d(t''_i) \leqq d(t) - 1 + d(t')$, $i = 2, 3$. □

3. A complete axiomatization of if-then-else with additional operations, predicates and equations. So far, we have considered the if-then-else in isolation. In practice, of course, the if-then-else will be used in a *context*, with various kinds of data elements in addition to the Booleans and a variety of functions operating on the data and predicates testing the data for different properties. From an algebraic point of view, this context can be summarized very simply. It is a many-sorted algebra [22], with one sort for each different kind of data, and having as operations: the given functions, predicates, and, of course, the if-then-else. Taking a many-sorted approach is crucial for our algebraic semantics to be of more than theoretical interest, since forcing everything into the same sort leads in practice to nonsense expressions such as dividing a Boolean by an integer, etc. In the many-sorted approach such nonsense is ruled out ex definitio. Perhaps more importantly, the *models* of interest in programming practice have, in the overwhelming majority of cases, a natural many-sorted structure. Simulating that structure by a one-sorted treatment would be conceptually unsatisfactory, and technically very awkward, even allowing that a satisfactory such simulation were at all possible.

For simplicity, we will treat the two-sorted case, with one sort *Bool* for Boolean values and another sort *Data* for the non-Boolean data elements. The generalization to an arbitrary number of non-Boolean sorts is straightforward.

Let now

$$\Omega = \Sigma_d \cup \bigcup_{n\in N} \Lambda_n \cup \bigcup_{n\in N} \Pi_n$$

where $\lambda_n \in \Lambda_n$ is an operation of sort $Data^n \to Data$ and $\pi \in \Pi_n$ is a predicate of sort $Data^n \to Bool$. The only operation with both Boolean and non-Boolean arguments is $[-,-,-]_d$. Let $\Lambda = \cup\{\Lambda_n | n \in N\}$ and $\Pi = \cup\{\Pi_n | n \in N\}$.

The signature Ω can be pictured as in Fig. 2:

Let E_Ω be the following set of axioms (12, 12′, 13, 13′), expressing the strictness of the operations and the predicates and their commutativity with respect to the if-then-else: for any λ in Λ_n, any π in Π_n, $n \geqq 1$:

$$(12) \qquad \begin{cases} \lambda(\bot, x_2, \cdots, x_n) = \bot, \\ \cdots \\ \lambda(x_1, \cdots, x_{n-1}, \bot) = \bot, \end{cases} \quad \text{"strictness,"}$$

$$(12') \qquad \begin{cases} \pi(\bot, x_2, \cdots, x_n) = \bot, \\ \cdots \\ \pi(x_1, \cdots, x_{n-1}, \bot) = \bot, \end{cases} \quad \text{"strictness,"}$$

$$(13) \qquad \lambda(x_1, \cdots, x_{k-1}, [u, v, w], x_{k+1}, \cdots, x_n) = [u, \lambda(x_1, \cdots, x_{k-1}, v, x_{k+1}, \cdots, x_n), \lambda(x_1, \cdots, x_{k-1}, w, x_{k+1}, \cdots, x_n)]$$

for $k = 1, \cdots, n$, "commutativity" axiom,

$$(13') \qquad \pi(x_1, \cdots, x_{k-1}, [u, v, w], x_{k+1}, \cdots, x_n) = [u, \pi(x_1, \cdots, x_{k-1}, v, x_{k+1}, \cdots, x_n), \pi(x_1, \cdots, x_{k-1}, w, x_{k+1}, \cdots, x_n)]$$

for $k = 1, \cdots, n$, "commutativity" axiom.

THEOREM 9 (completeness). *Let* $\Omega_0 = \{\bot, T, F\} \cup \Lambda \cup \Pi$, *and* E *a set of equations involving* Ω_0*-terms and containing the strictness equations* (12) *and* (12′). *Define* $Ax_\Omega = Ax_d \cup E_\Omega$, *and* $\mathscr{C}_\Omega$ *as the class of* Ω*-algebras* A *such that* $\mathbf{A}_{|\Sigma d} \in \mathscr{C}_d$ *and* $\mathbf{A}_{|\Lambda\cup\Pi} \in \mathrm{Alg}^{E_\Omega}_{\Lambda\cup\Pi}$. *Let* $\mathscr{C}^E_\Omega = \{A \in \mathscr{C}_\Omega | A \text{ satisfies } E\}$. *Then*:

$$\mathscr{C}^E_\Omega \models t_1 = t_2 \quad \textit{iff } Ax_\Omega \cup E \vdash t_1 = t_2.$$

Proof. The "if" direction is clear except for the validity of the axioms (13), (13′) which are fairly natural. They are similar to axiom 1.2.10 of [5], and also to the "case analysis" rules of [16]. They are sound because of the strictness assumption (12), (12′) on the λ's and the π's. Hence these axioms are semantically valid in $\mathscr{C}^E_\Omega$.

For the "only if" direction, we first state and prove two lemmas. The intuition behind Lemma 10 is the following: first, use $Ax_\Omega \cup E$ to put every pair of trees in a

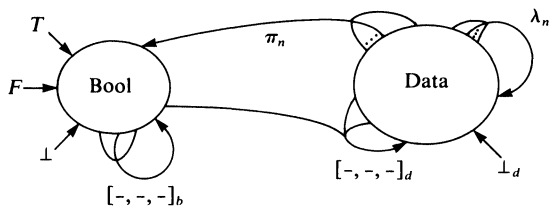


FIG. 2

"quasi-normal form" (the t_i'''s) where the root parts (the t_i''s) are trees with if-then-else only, and the lower parts consist of trees on the other operations. Moreover, all occurrences of trees in the lowest parts which are E-equivalent, are substituted by the same single representative of their equivalence class. The strategy for the completeness theorem will then be as follows: having extensively used the axioms concerning the extra operations to obtain the "quasi-normal form," we do not need them any more, namely, if two trees are $\mathscr{C}_\Omega^E$-equivalent, then their root parts are $\mathscr{C}_d$-equivalent.

LEMMA 10. *Let t_i be in $T_\Omega(Y)$ for $i=1,2$. Then, there exist t_i' in $T_{\Sigma_d}(V)$, such that, for $i=1,2$:*

$$Ax_\Omega \cup E \vdash t_i = t_i'' \quad \text{where } t_i'' = t_i'(u_j \leftarrow \theta_j, w_k' \leftarrow p_{j_k}(\vec{\theta_k'}), w_l)$$

and

$U=\{u_j\}$ *is a set of variables of sort Data,*

$W=\{w_l\}$ *is a set of variables of sort Bool,*

$W'=\{w_i'\}$ *is a set of variables of sort Bool,*

$V=U\cup W\cup W'$ *and all leaves in t_i' are labeled by a variable in V or a Boolean constant in $\{T, F, \perp\}$,*

$Y=X\cup W$.

For each k, $\vec{\theta_k'}=(\theta'_{k_1},\cdots,\theta'_{k_n})$ is a vector, and the θ_j's, θ'_{k_i}'s are pairwise distinct trees in $T_{\Lambda-\{\perp\}}(X)\cup\{\perp\}$, where X is a set of variables of sort Data.

Moreover:

(i) *the equivalence classes modulo E of the θ_j's and the θ'_{k_i}'s are pairwise distinct elements in $T_{\Omega_0,E}(X)_{Data}$ (namely the terms of sort Data in $T_{\Omega_0,E}(X)$)*;

(ii) *the equivalence classes modulo E of the $p_{j_k}(\vec{\theta_k'})$'s are also pairwise distinct, and for all k, $p_{j_k}(\vec{\theta_k'}) \not\equiv_E c \in \{T, F, \perp\}$.*

Proof. By structural induction (on the depths of t_1, t_2). The intuitive idea is to shuffle the trees t_i, $i=1,2$, by using the commutativity and strictness axioms in such a way that all the occurrences of the if-then-else be grouped together at the root, and all the other operations be grouped together at the lower part of the tree. Moreover, we use the strictness of predicates and operations to simplify trees in $T_\Lambda(X)$ to trees in $T_{\Lambda-\{\perp\}}(X)$ or to $\perp$. Note that $X\cap U$ may be nonempty. If any two subtrees θ, θ' in $\{\theta_j, \theta'_{k_i}, p_{j_k}(\vec{\theta_k'})\}$ are E-equivalent, $E\vdash\theta=\theta'$, and we can replace θ' by θ. We can thus "rename" all occurrences of E-equivalent subtrees by the same single subtree. If $p_{j_k}(\vec{\theta_k'}) \equiv_E c \in \{T, F, \perp\}$, then $E\vdash p_{j_k}(\vec{\theta_k'})=c$ and we can replace $p_{j_k}(\vec{\theta_k'})$ by the constant c in t_i'': this amounts to substituting the constant c for w_k' in t_i' (see Fig. 3). □

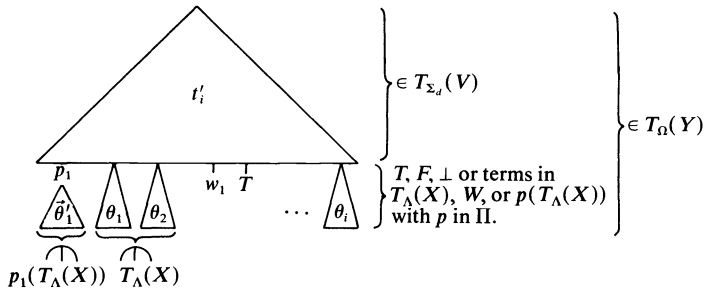


FIG. 3

An immediate consequence of Lemma 10, which we will use implicitly, is that for all algebras A in $\mathscr{C}_\Omega$ the derived operations t_{i_A} and t''_{i_A} defined by t_i and t''_i coincide.

LEMMA 11. *Let t' be in $T_{\Sigma_d}(V)$, A an algebra in $\mathscr{C}_d$ and $\alpha : V \to A$ a valuation which can be decomposed into $\alpha_b : W \to A_{Bool}$ and $\alpha_d : U \to A_{Data}$. Let $n = \text{path}(\alpha_b, t')$, then*

$$\bar{\alpha}(t') = \begin{cases} \alpha(t'(n)) & \text{if } n \in \{2,3\}^*, \\ \bot & \text{if } n \in \{2,3\}^*\bot, \end{cases}$$

where $\bar{\alpha}$ denotes the unique homomorphism $T_{\Sigma_d}(V) \to A$ extending α.

Proof. By induction on the depth of t'. □

We can now proceed with the proof of Theorem 9 ($\Rightarrow$). We prove:

(iii) $$\mathscr{C}_\Omega^E \models t_1 = t_2 \quad \Rightarrow \quad \mathscr{C}_d \models t'_1 = t'_2.$$

This, together with the completeness theorem for the class $\mathscr{C}_d$ (Theorem 6), will imply that $Ax_d \vdash t'_1 = t'_2$, hence by substitution $Ax_d \vdash t''_1 = t''_2$, and then, combining with Lemma 10 and noting that $Ax_d \subset Ax_\Omega$, $E \cup Ax_\Omega \vdash t_1 = t_2$, the desired result.

In order to prove (iii), we assume $\mathscr{C}_d \not\models t'_1 = t'_2$ and derive a contradiction. If $\mathscr{C}_d \not\models t'_1 = t'_2$, then, there exist A in $\mathscr{C}_d$ and a valuation $\alpha : V \to A$ such that $\bar{\alpha}(t'_1) \neq \bar{\alpha}(t'_2)$. Let us split α into its Boolean part $\alpha_b : W \cup W' \to \{\bot, T, F\} = A_{Bool}$, and its data part $\alpha_d : U \to A_{Data}$. Then, for $i = 1, 2$, $\bar{\alpha}(t'_i) = \alpha(v_i)$, where $n_i = \text{path}(\alpha_b, t'_i)$ and v_i is the label of the leaf n_i in t'_i if this path does not end with $\bot$, and is $\bot$ otherwise, namely:

$$v_i = \begin{cases} t'_i|n_i & \text{if } n_i \notin \{2,3\}*\bot, \\ \bot & \text{otherwise.} \end{cases}$$

We now define an Ω-algebra B in $\mathscr{C}_\Omega^E$, together with a valuation $\beta : Y \to B$, such that $\bar{\beta}(t_1) \neq \bar{\beta}(t_2)$.

The domain of B is defined as follows:

$$B_{Bool} = \{\bot, T, F\},$$

$$B_{Data} = T_{\Omega_0,E}(X)_{Data}.$$

Recall that $T_{\Omega_0,E}(X)_{Data}$ denotes the set of elements of sort *Data* in the initial $\Omega_0 - E$-algebra $T_{\Omega_0,E}(X)$. Note that here, splitting E into the equations of sort *Data* and of sort *Bool*, $E = E_b \cup E_d$, we have $T_{\Omega_0,E}(X)_{Data} = T_{\Lambda,E_d}(X)$. The Λ-operations of B_{Data} are defined as usual in $T_{\Lambda,E_d}(X)$. The Π-operations (predicates) are defined on B_{Data} as follows:

$$p_{j_B}([\vec{\theta}]_E) = \begin{cases} c & \text{if } p_j(\vec{\theta}) \equiv_E c \in \{\bot, T, F\}, \\ \alpha_b(w'_k) & \text{if there exists a } k \text{ such that } p_j(\vec{\theta}) \equiv_E p_{j_k}(\overrightarrow{\theta'_k}), \\ \bot & \text{otherwise.} \end{cases}$$

This is possible since we assumed the $[p_{j_k}(\overrightarrow{\theta'_k})]_E$ pairwise distinct and distinct from the constants $\bot$, T, F; $[\theta]_E$ denotes the equivalence class of θ modulo E, and similarly, for $\vec{\theta} = (\theta_1, \cdots, \theta_n)$ we denote by $[\vec{\theta}]_E$ the vector $([\theta_1]_E, \cdots, [\theta_n]_E)$. The p_{j_B}'s are well defined. The p_{j_B}'s can be viewed as factoring through $T_{\Omega_0,E}(X)_{Bool}$, the set of classes of terms of sort *Bool* in the free Ω_0-E-algebra. See Fig. 4.

Finally, the if-then-else operations $[-,-,-]_b$ and $[-,-,-]_d$ are defined as usual. B thus defined is in $\mathscr{C}_\Omega$ and satisfies all the equations in E, by construction.

Define now the valuation $\beta : Y \to B$ by: $\beta_b(w_l) = \alpha_b(w_l)$ and $\beta_d(x) = x$. Using then β_b together with the values of the terms of the form $p_{j_k}(\overrightarrow{\theta'_k})$ which occur in premise

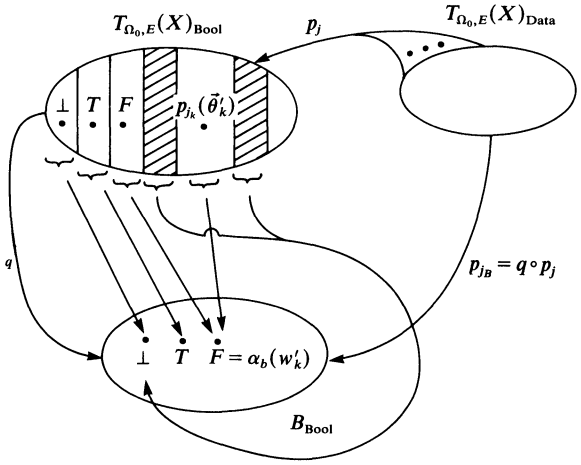


FIG. 4

position in the if-then-else operators of t''_i, we can define path (β, t''_i) and path $(\beta, t''_i) =$ path $(\alpha_b, t'_i) = n_i$ for $i = 1, 2$. Similarly, $\bar{\beta}(t''_i) = \bar{\beta}(t''_i|nn_i)$ where $''t_i|n_i$ is the subtree of $''t_i$ at node n_i if $n_i \in \{2, 3\}^*$, and $\bar{\beta}(t''_i) = \perp$ otherwise (namely if $n_i \in \{2, 3\}^*\perp$).

Then, either

(a) t_i, t''_i and t'_i are of sort *Data*, $v_i = u_i$ and $t''_i|n_i = \theta_i$. Hence in this case $\bar{\alpha}(t'_1) = \alpha(u_1) \neq \alpha(u_2) = \bar{\alpha}(t'_2)$ implies $u_1 \neq u_2$ and thus $\theta_1 \not\equiv_E \theta_2$, and $\bar{\beta}(\theta_1) \neq \bar{\beta}(\theta_2)$ since $\bar{\beta}(\theta_i) = [\theta_i]_E$ for terms θ_i in $T_\Lambda(X)$. Hence finally $\bar{\beta}(t_1) = \bar{\beta}(t''_1) \neq \bar{\beta}(t''_2) = \bar{\beta}(t_2)$, or

(b) t_i, t''_i and t'_i are of sort *Bool* and

- either $v_i = w_l$ and $\bar{\beta}(t''_i|n_i) = \beta_b(w_l) = \alpha_b(w_l) = \bar{\alpha}(t'_i)$, or
- $v_i = w'_k$ and $\bar{\beta}(t''_i|n_i) = p_{j_{k_B}}([\vec{\theta}'_k]_E) = \alpha_b(w'_k) = \bar{\alpha}(t'_i)$.

So, in both cases, using the definition either of β_b or of the predicate operations on B, $\bar{\beta}(t''|n_i) = \bar{\alpha}(t'_i)$, hence $\bar{\beta}(t_1) = \bar{\beta}(t''_1) \neq \bar{\beta}(t''_2) = \bar{\beta}(t_2)$ which contradicts $\mathscr{C}_\Omega^E \models t_1 = t_2$. □

COROLLARY 12. *Let* $Ax_\Omega = Ax_d \cup E_\Omega$. *Let* $\mathscr{C}_\Omega$ *be the class of* Ω*-algebras* A *such that* $\mathbf{A}_{|\Sigma_d} \in \mathscr{C}_d$ *and* $\mathbf{A}_{|\Lambda \cup \Pi} \in \text{Alg}^{E_\Omega}_{\Lambda \cup \Pi}$; *then*:

$$\mathscr{C}_\Omega \models t_1 = t_2 \quad \textit{iff } Ax_\Omega \vdash t_1 = t_2. \qquad \square$$

Regarding our choice of axioms and signatures for the many-sorted case, the following remarks are in order:

(i) Additional Boolean operations are not needed, since all the usual Boolean operations can be expressed as derived operations in terms of the if-then-else. For instance, we can define $\vee$ and $\sim$ [16, 18, 26] by:

$$x \vee y = [[y, x, x], T, y],$$

$$\sim x = [x, F, T].$$

(ii) The strictness of the operations and predicates means that the basic data type operations and predicates with the exclusion of the if-then-else correspond exactly to partial functions when the bottom element is removed. This is in contrast to the if-then-else operator, [-, -, -], which is nonstrict and cannot be treated as a ternary partial operation when bottom is removed.

(iii) When the approximation ordering, making $\perp$ less defined than any other term, is added (as will be done in § 4) the strictness condition makes $T_{\Omega_0,E}(X)$ into a flat c.p.o. which is at the same time the free continuous algebra on X for the equations E. Nonflatness arises by the addition of the nonstrict operator $[-,-,-]$, which then gives rise to infinite terms corresponding for instance to unfoldments of programs defined by recursive equations in terms of the basic operations and predicates with the help of the if-then-else.

(iv) Since we assume that the basic operations and predicates are equationally definable, the equations E need not involve the if-then-else operator. It is only the recursive equations corresponding to programs using the basic operations that will require use of the if-then-else.

(v) The completeness theorem might still be valid for the case of equations mixing the if-then-else with the basic operations and predicates (one should probably require that the equations do not collapse the Boolean sort) but we have not checked that case.

4. Adding ordering and continuity.

4.1. If-then-else without additional operations. The results of the present subsection, as well as the proofs, are valid for any of the signatures $\Sigma, \Sigma_b, \Sigma_d$ introduced in § 2, and the corresponding classes $\mathscr{C}, \mathscr{C}_b, \mathscr{C}_d$ of algebras with if-then-else operations. So, unless otherwise specified, we will take the shorthand notation Σ, $\mathscr{C}$ and Ax, to denote any of Σ, $\mathscr{C}$ and Ax (resp. Σ_b, $\mathscr{C}_b$ and Ax_b, or Σ_d, $\mathscr{C}_d$ and Ax_d).

We assume familiarity with the basic notions about ω-c.p.o.'s, (ω)-continuous functions, (ω)-continuous algebras, and satisfaction of equations and inequations (as explained, for instance, in [13], [21]).

Notation. For Σ a signature, let Alg_Σ denote the category of Σ-algebras, $\mathrm{Alg}(o)_\Sigma$ (resp. $\mathrm{Alg}(\omega)_\Sigma$, $\mathrm{Alg}(d)_\Sigma$) denote the class of ordered Σ-algebras with monotone operations, (resp. continuous Σ-algebras with continuous operations, discrete Σ-algebras with a flat ordering and continuous operations).

Let $V: \mathrm{Alg}(\omega)_\Sigma \to \mathrm{Alg}_\Sigma$ be the forgetful functor sending each continuous Σ-algebra to its underlying Σ-algebra, and let $\mathscr{C}(\omega)$ (resp. $\mathscr{C}(d)$) be the class of continuous (resp. discrete) algebras in $\mathscr{C}$. Formally:

$$\mathscr{C}(\omega) = \{I \in \mathrm{Alg}(\omega)_\Sigma \mid V(I) \in \mathscr{C}\},$$
$$\mathscr{C}(d) = \{I \in \mathrm{Alg}(d)_\Sigma \mid V(I) \in \mathscr{C}\}.$$

There are inclusions

(i) $\mathscr{C}(d) \subseteq \mathscr{C}(\omega)$;

(ii) $V(\mathscr{C}(d)) \subseteq V(\mathscr{C}(\omega)) \subseteq \mathscr{C}$,

$$\mathrm{Alg}(\omega)_\Sigma^{Ax} = \{A \text{ in } \mathrm{Alg}(\omega)_\Sigma \mid A \text{ satisfies } Ax\},$$
$$\mathrm{Alg}(d)_\Sigma^{Ax} = \{I \text{ in } \mathrm{Alg}(d)_\Sigma \mid I \text{ satisfies } Ax\}.$$

Noting that:

(1) discrete algebras are continuous, and

(2) all algebras in $\mathscr{C}$ (resp. $\mathscr{C}(x)$, $x = d, \omega$) satisfy the axioms Ax, hence are in Alg_Σ^{Ax} (resp. $\mathrm{Alg}(x)_\Sigma^{Ax}$), we obtain the inclusions:

(iii)
$$\begin{array}{ccc} \mathscr{C}(d) & \subseteq & \mathscr{C}(\omega) \\ \cap| & & \cap| \\ \mathrm{Alg}(d)_\Sigma^{Ax} & \subseteq & \mathrm{Alg}(\omega)_\Sigma^{Ax} \end{array}.$$

We shall see that all the inclusions in diagram (iii) are strict: for horizontal inclusions it is clear; for vertical inclusions this will follow from Theorem 19 in this section.

Translating diagram (iii) via the forgetful functor V (forgetting the ordering) yields the following diagram (iv) of classes of unordered algebras:

$$\text{(iv)} \qquad \begin{array}{ccccc} V(\mathscr{C}(d)) & \subseteq & V(\mathscr{C}(\omega)) & \subseteq & \mathscr{C} \\ |\cap & & |\cap & & |\cap \\ V(\mathrm{Alg}\,(d)_\Sigma^{Ax}) & \subseteq & V(\mathrm{Alg}\,(\omega)_\Sigma^{Ax}) & \subseteq & \mathrm{Alg}_\Sigma^{Ax} \end{array}.$$

From Theorem 17 and the corollary of Theorem 14 we can deduce that diagram (iv) collapses as follows:

$$\text{(iv)}' \qquad \begin{array}{ccccc} \mathscr{C} & = & \mathscr{C} & = & \mathscr{C} \\ \cap & & \cap & & \cap \\ V(\mathrm{Alg}\,(d)_\Sigma^{Ax}) & \subseteq & V(\mathrm{Alg}\,(\omega)_\Sigma^{Ax}) & \subseteq & \mathrm{Alg}_\Sigma^{Ax} \end{array}.$$

For $\mathscr{A}$ a class of Σ-algebras, $\equiv_{\mathscr{A}}$ denotes the set of Σ-equations $t = t'$ satisfied by all the algebras in $\mathscr{A}$.

For $\mathscr{A}$ a class of ordered Σ-algebras, $\leqq_{\mathscr{A}}$ denotes the set of Σ-inequations $t \leqq t'$ satisfied by all algebras in $\mathscr{A}$. We denote by $\equiv_{\mathscr{A}}$ the set of equations $\equiv_{V(\mathscr{A})}$.

Let Σ be a signature containing a special element $\perp$. Let $T_\Sigma(X)$ denote the free Σ-algebra on generators X, ordered by the least compatible ordering $<$ such that: $\perp < t$, for all t in $T_\Sigma(X)$. Recall that the rules of inequational deduction are: reflexivity, transitivity, monotonicity of the operations (i.e., $t_i \leqq t_i'$, $i = 1, \cdots, n$, entail $f(t_1, \cdots, t_n) \leqq f(t_1', \cdots, t_n')$) and substitution closure or full invariance (i.e., for any morphism $h : T_\Sigma(X) \to T_\Sigma(X)$, $t \leqq t'$ entails $h(t) \leqq h(t')$). For E a set of Σ-inequations of the form $t \leqq t'$, for t, t' in $T_\Sigma(X)$, $\leqq_E$ denotes the closure of $(E \cup <)$ under the rules of inequational deduction [13], [4], [23], [7], etc. By the completeness theorem for ordered algebras, if $\mathrm{Alg}\,(o)_\Sigma^E$ denotes the class of all ordered algebras satisfying E, then

$$\leqq_{\mathrm{Alg}\,(o)_\Sigma^E} = \leqq_E.$$

CONVENTION 13. *For E a set of Σ-equalities, let $E' = \{t \leqq t',\ t' \leqq t \mid t = t' \text{ is in } E\}$. Similarly, $\mathrm{Alg}\,(o)_\Sigma^E$ will denote the class of all ordered Σ-algebras which satisfy the inequations in E', and $\leqq_E = \leqq_{E'}$ will denote the closure of $(E' \cup <)$ under the rules of inequational deduction.*

Considering the orderings associated with the classes in diagram (iii) we get:

$$\text{(v)} \qquad \begin{array}{ccc} \leqq_{\mathscr{C}(d)} & \supseteq & \leqq_{\mathscr{C}(\omega)} \\ \cup| & & \cup| \\ \leqq_{\mathrm{Alg}(d)^{Ax}} & \supseteq & \leqq_{Ax} \end{array}.$$

From Theorem 16 and the subsequent remark we can deduce that the horizontal inclusions collapse into equalities. Theorem 19 shows that the only remaining vertical inclusion is strict. Considering the equivalences associated with the classes in diagram (iv) we get:

$$\text{(vi)} \qquad \begin{array}{ccccc} \equiv_{\mathscr{C}(d)} & \supseteq & \equiv_{\mathscr{C}(\omega)} & \supseteq & \equiv_{\mathscr{C}} \\ \cup| & & \cup| & & \cup|. \\ \equiv_{\mathrm{Alg}(d)^{Ax}} & \supseteq & \equiv_{\mathrm{Alg}(\omega)^{Ax}} & \supseteq & \overline{Ax} \end{array}$$

THEOREM 14. *All the inclusions in* (vi) *are equalities.*

Proof. By Theorem 1 it is enough to show $V(\mathscr{C}(d)) \supseteq \mathscr{C}$. If A is an algebra in $\mathscr{C}$, let A^d denote the continuous Σ-algebra obtained by giving to A the discrete ordering $(x > y \Leftrightarrow y = \perp)$, and leaving the Σ operations as before. This is a continuous Σ-algebra

as a consequence of the more general characterization of Lemma 15 below. We thus have $V(A^d) = A$, which proves the theorem. □

COROLLARY. $V(\mathscr{C}(d)) = \mathscr{C}$. □

LEMMA 15. *Let A be an ordered set with bottom element $\perp$ and constants $T, F, \perp \neq T \neq F \neq \perp$. Let $[-,-,-]: A^3 \to A$ be the function:*

$$[x, y, z] = \begin{cases} y & \text{if } x = T, \\ z & \text{if } x = F, \\ \perp & \text{otherwise.} \end{cases}$$

Then (1) *$[-,-,-]$ is monotonic iff T and F are maximal elements*; (2) *$[-,-,-]$ is continuous if, in addition, T and F are not "reachable by chains," i.e. there is no properly ascending chain $\{a_n\}$ such that $T = \sqcup a_n$, and similarly for F.* (2) *says that T and F are finite elements of the domain.*

Proof. (1) ($\Leftarrow$) Suppose $(x, y, z) > (x', y', z')$ (componentwise ordering). If $x \neq T, F$, then $x' \neq T, F$ and we have

$$[x, y, z] = [x', y', z'] = \perp.$$

So we can reduce to the case

$$\begin{array}{l} (T, y, z) > (x', y', z') \\ (F) \\ \text{If } x' \neq T \text{ (resp. } \neq F) \text{ we are again done,} \\ \quad \text{so it is enough to consider the case:} \\ (T, y, z) > (T, y', z') \\ (F) \qquad (F) \end{array}$$

in either case $[-,-,-]$ is a projection function, so all is fine.

(1) ($\Rightarrow$) Trivial; suppose $a > T$
(F)
then $[T, a, \perp] = a > [a, a, \perp] = \perp$ contradicting the monotonicity.

(2) The only trouble for continuity comes from sequences $\{a_n\}$ such that $\sqcup a_n = T$, and for all n $a_n \neq T$.

In that case we have

$$[a_n, x, y] = \perp \quad \text{but} \quad [T, x, y] = x.$$

Since there are no properly ascending chains reaching T or F, the above situation is ruled out, and $[-,-,-]$ is continuous. □

Note that the hypotheses of Lemma 15 are trivially satisfied in the many-sorted case, where A_{Bool} is reduced to $\{T, F, \perp\}$.

THEOREM 16. $\leqq_{\mathscr{C}(d)} = \leqq_{\mathscr{C}(\omega)}$.

Proof. By (v) $\leqq_{\mathscr{C}(d)} \supseteq \leqq_{\mathscr{C}(\omega)}$; hence it is enough to show that if a Σ-inequality $t \leqq t'$ does not hold for some I in $\mathscr{C}(\omega)$, there is a discrete I' in $\mathscr{C}(d)$ for which $t \leqq t'$ does not hold either.

Suppose $t \leqq t'$ does not hold for I in $\mathscr{C}(\omega)$. We show that it also fails for $V(I)^d$. Otherwise, it would mean that for each $(a_1, \cdots, a_n) \in I^n$ either $t_I(a_1, \cdots, a_n) = t'_I(a_1, \cdots, a_n)$ or $t_I(a_1, \cdots, a_n) = \perp$. Of course, this forces $t \leqq t'$ to hold in I, a contradiction. □

Remark. It has been shown by Benoit [2] that $\leqq_{\mathrm{Alg}(d)^{Ax}} = \leqq_{Ax}$: hence (v) collapses to the inclusion $\leqq_{Ax} \subseteq \leqq_{\mathscr{C}(\omega)}$, which is strict by Theorem 19.

THEOREM 17. *The three vertical inclusions in* (iv) *are strict.*

Proof. Recall that:

$$\text{(iv)}\qquad \begin{array}{ccccc} V(\mathscr{C}(d)) & \subseteq & V(\mathscr{C}(\omega)) & \subseteq & \mathscr{C} \\ |\cap & & |\cap & & |\cap \\ V(\mathrm{Alg}\,(d)^{Ax}_{\Sigma}) & \subseteq & V(\mathrm{Alg}\,(\omega)^{Ax}_{\Sigma}) & \subseteq & \mathrm{Alg}^{Ax}_{\Sigma} \end{array}.$$

It is enough to exhibit a discrete $I \in \mathrm{Alg}\,(d)^{Ax}$ such that $V(I)$ is not in $\mathscr{C}$. This follows immediately from the:

LEMMA 18. *There is a discretely ordered algebra satisfying the axioms Ax, for which the function* $[-,-,-]$ *is not the if-then-else.*

Proof. The proof is based on the use of Birkhoff's Theorem. Let I' be a nontrivial algebra in $\mathscr{C}$, e.g., $I' = \{\bot, F, T, a\}$, and let $I = I' \times I'$. By Birkhoff's Theorem, I is in $\mathrm{Alg}^{Ax}_{\Sigma}$. If moreover I is given the discrete ordering: $y \leqq x$ iff $y = (\bot, \bot)$ or $y = x$, then $I \in \mathrm{Alg}\,(d)^{Ax}_{\Sigma}$, hence $V(I)$ belongs to all the varieties of algebras on the lower row of (iv). Now $V(I)$ does not belong to $\mathscr{C}$, since $\mathscr{C}$ is not closed under products: as mentioned in the Introduction, this follows from the fact that the product of two nontrivial algebras with if-then-else has never $[-,-,-]$ as an if-then-else, e.g., $[(T, F), (F, F), (T, T)] = (F, T)$, which is different from both (F, F) and (T, T). □

THEOREM 19. *The vertical inclusions in* (iii) *and* (v) *are strict.*

Proof. Recall that diagram (v) relates the orderings associated with the algebras in (iii) as follows:

$$\text{(v)}\qquad \begin{array}{ccc} \leqq_{\mathscr{C}(d)} & \supseteq & \leqq_{\mathscr{C}(\omega)} \\ \cup| & & \cup| \\ \leqq_{\mathrm{Alg}(d)^{Ax}} & \supseteq & \leqq_{Ax} \end{array}.$$

Hence, it is enough to show the result for (v). For this, it is enough to show an inequality $t \leqq t'$ in $\leqq_{\mathscr{C}(\omega)}$ such that $t \leqq t'$ does not belong to $\leqq_{\mathrm{Alg}(d)^{Ax}}$. The inequality:

$$[y, x, x] \leqq x$$

belongs clearly to $\leqq_{\mathscr{C}(\omega)}$ and it does not belong to $\leqq_{\mathrm{Alg}(d)^{Ax}}$ since for the discrete I constructed in the proof of Theorem 17 we have:

$$\text{for } y = (\bot, T) \text{ and } x = (T, T): [y, x, x] = (\bot, T) \not\leqq x = (T, T).$$

Since I belongs to $\mathrm{Alg}\,(d)^{Ax}$, the theorem is proved. □

We shall call a class of discrete algebras *d-equational* [12] iff it is of the form $\mathrm{Alg}\,(d)^{E}_{\Sigma}$ for some set of equations E.

THEOREM 20. $\mathscr{C}(d)$ *is not d-equational* [12].

Proof. Suppose there exists some set E of equations such that

$$\mathscr{C}(d) = \{I / I \text{ is discrete and } I \text{ satisfies } E\}.$$

Then, one would have (1) $\mathscr{C}(d) = \mathrm{Alg}\,(d)^{E} \subset \mathrm{Alg}\,(o)^{E}$ since $\mathrm{Alg}\,(d) \subseteq \mathrm{Alg}\,(o)$. By Theorem 14, $\overline{Ax} = \equiv_{\mathscr{C}(d)}$. By the Completeness Theorem for ordered algebras, $\leqq_{\mathrm{Alg}(o)^{E}} = \leqq_{E}$. Hence, $\equiv_{\mathrm{Alg}(o)^{E}} \supseteq \bar{E}$. So, (1) implies

$$\overline{Ax} = \equiv_{\mathscr{C}(d)} \supset \equiv_{\mathrm{Alg}(o)^{E}} \supset \bar{E} \supset E.$$

But, we know from Theorem 19 that there exists an I in $\mathrm{Alg}\,(d)^{Ax}$, which satisfies Ax and a fortiori E, and which does not belong to $\mathscr{C}(d)$. □

Theorem 14 showed that Ax, together with the rules of equational logic, forms a complete proof system for proving all equalities between finite terms valid in $\mathscr{A}$, where $\mathscr{A}$ is any of the classes $\mathscr{C}(d), \cdots, \mathrm{Alg}^{Ax}$. We will now extend this result to infinite

trees. We shall denote by $CT_\Sigma(X)$ the continuous algebra of infinite Σ-terms (trees) with variables in X, and by $T, T', \cdots$ etc. its elements.

For T, T' infinite trees, we write

$$T \equiv^{\infty}_{\mathscr{A}} T' \quad \text{iff } (\forall A \in \mathscr{A})\colon A \models T = T',$$

$$T \leqq^{\infty}_{\mathscr{A}} T' \quad \text{iff } (\forall A \in \mathscr{A})\colon A \models T \leqq T'.$$

$T = \bigsqcup t_n$ expresses T as the limit of a chain of finite terms $t_n \in T_\Sigma(X)$.

We will show now that Ax remains a complete proof system for proving equations which hold between infinite terms built on a finite number of variables. The intuition is that any infinite term is $\mathscr{A}$ equivalent ($\equiv^{\infty}_{\mathscr{A}}$) to a finite term. This stems from the following lemma:

LEMMA 21. *Let $\mathscr{A} = \mathscr{C}(d)$. For any set V_n of n variables, there exists a finite $F_n \subset \mathscr{A}$ such that all algebras in F_n have a finite domain and for any T_1, T_2 in $CT_\Sigma(V_n)$:*

$$T_1 \equiv^{\infty}_{\mathscr{A}} T_2 \Leftrightarrow T_1 \equiv^{\infty}_{F_n} T_2.$$

Proof. Only the ($\Leftarrow$) implication needs a proof. Let:

$$F_n = \{I \in \mathscr{D} \mid \mathrm{dom}(I) \text{ consists of at most } n+4 \text{ elements}\}.$$

$n+4$ represents the cardinality of the possible data $V_n \cup \{\bot, T, F, \bot_d\}$. F_n is finite up to isomorphism. Let $I \in \mathscr{A}$. Let $\alpha = (a_1, \cdots, a_n)$ be a valuation $\alpha : V_n \to I$. Let I_α be I restricted to $\{\bot, T, F, a_1, \cdots, a_n, \bot_d\}$. $I_\alpha \in F_n$. Moreover, for any T in $CT_\Sigma(V_n)$, $T_I(a_1, \cdots, a_n) = T_{I_\alpha}(a_1, \cdots, a_n)$. Hence $T_1 \not\equiv^{\infty}_{\mathscr{A}} T_2 \Rightarrow T_1 \not\equiv^{\infty}_{F_n} T_2$. □

The same lemma also holds replacing $\mathscr{A}$ by any of the classes in diagram (vi).

We now show how to lift the result of Bloom and Tindell [5] to infinite trees. We will show that the axiom system Ax remains complete for proving equality between infinite trees if we replace standard equational logic by inequational logic together with the induction rule:

For infinite trees $T = \bigsqcup t_n$, $T' = \bigsqcup t'_n$ ($\{t_n\}$ and $\{t'_n\}$ increasing sequences):

$$(*) \qquad \text{If } \forall n\ \exists m\ t_n \leqq_{Ax} t'_m \quad \text{then } T \leqq^{\infty}_{Ax} T'.$$

Define then $T =^{\infty}_{Ax} T'$ iff $T \leqq^{\infty}_{Ax} T'$ and $T' \leqq^{\infty}_{Ax} T$.

THEOREM 22. *Ax, together with the rules of inequational logic, and the induction rule $(*)$ gives a complete proof system for deriving all valid equalities $T \equiv^{\infty}_{\mathscr{A}} T'$ for $T, T' \in CT_\Sigma(V_n)$ with $\mathscr{A} = \mathscr{C}(d)$ or $\mathscr{A} = \mathscr{C}(\omega)$.*

Proof. By the proof of Theorem 26, $\equiv^{\infty}_{\mathscr{C}(d)} = \equiv^{\infty}_{\mathscr{C}(\omega)}$. Thus, it suffices to consider $\equiv^{\infty}_{\mathscr{C}(d)}$. Completeness amounts to the following:

$$\text{For } T, T' \text{ infinite trees, } T \equiv^{\infty}_{\mathscr{A}} T' \Leftrightarrow T =^{\infty}_{Ax} T'.$$

The ($\Leftarrow$) implication is clear. To show the ($\Rightarrow$) implication, consider first the case where T, T' are finite trees. Then notice that, by Theorem 14,

$$\equiv_{\mathscr{A}} = \equiv_{\mathrm{Alg}(\omega)^{Ax}} = \overline{Ax}.$$

By the Completeness Theorem for continuous algebras we have

$$(=^{\infty}_{Ax}) = \equiv^{\infty}_{\mathrm{Alg}(\omega)^{Ax}}.$$

Thus, for t, t' finite trees we have

$$(\dagger) \qquad t \equiv_{\mathscr{A}} t' \Leftrightarrow t =_{Ax} t' \Leftrightarrow t\ \overline{Ax}\ t'.$$

For the case where the trees are infinite, we shall use the following:

LEMMA 23. *Let $\{t_n\}$ be an increasing sequence and $T = \bigsqcup t_n$ an infinite tree in $CT_\Sigma(V_n)$. There exists a bound $m(T)$ such that $T \equiv^{\infty}_{\mathscr{A}} t_n \equiv_{\mathscr{A}} t_{m(T)}$ for each $n \geqq m(T)$.*

Proof. We will use Lemma 21. Notice that, if I is a finite interpretation and $T = \sqcup t_n$ then the sequence t_{n_I} is eventually constant and equal to T_I. So letting $F_n = \{I_1, \cdots, I_k\}$, for any $j = 1, \cdots, k$, there exists an n_j and a finite $t_{n_j} < T$ such that $T_{I_j} = (t_{n_j})_{I_j}$. Let $m(T) = \sup\{n_j | j = 1, \cdots, k\}$. $T \equiv^{\infty}_{F_n} t_{m(T)}$ and since the sequence $\{t_n\}$ is increasing, $T \equiv^{\infty}_{\mathscr{A}} t_n \equiv_{\mathscr{A}} t_{m(T)}$ for each $n \geqq m(T)$. □

For $T = \sqcup t_n$, $T' = \sqcup t'_n$ infinite trees we then have $T \equiv^{\infty}_{\mathscr{A}} t_n$, resp. $T' \equiv^{\infty}_{\mathscr{A}} t'_{n'}$, for each $n \geqq m(T)$, resp. $n' \geqq m(T')$. Thus, to prove that if $T \equiv^{\infty}_{\mathscr{A}} T'$ then $T =_{Ax} T'$ notice that the hypothesis forces $t_n \equiv_{\mathscr{A}} t'_n$ for each $n \geqq \max(m(T), m(T'))$ and by (†)

$t_n =_{Ax} t'_n$ for each $n \geqq \max(m(T), m(T'))$ in other words,

$t_n \leqq_{Ax} t'_n$ and $t'_n \leqq_{Ax} t_n$ for each $n \geqq \max(m(T), m(T'))$ which by the induction rule (*) gives $T =^{\infty}_{Ax} T'$. □

4.2. Adding extra operations, predicates and equations. We will in the present section extend the previous results to algebras over a signature which contains other function symbols besides the if-then-else. For any signature Ω and set E of Ω-equations, we define as previously:

$\mathrm{Alg}^E_\Omega = \{A \text{ in } \mathrm{Alg}_\Omega | A \text{ satisfies } E\}$,
$\mathrm{Alg}(o)^E_\Omega = \{A \text{ in } \mathrm{Alg}(o)_\Omega | A \text{ satisfies } E\}$,
$\mathrm{Alg}(\omega)^E_\Omega = \{I \text{ in } \mathrm{Alg}(\omega)_\Omega | I \text{ satisfies } E\}$,
$\mathrm{Alg}(d)^E_\Omega = \{I \text{ in } \mathrm{Alg}(d)_\Omega \text{ (discrete } \Omega - \textit{algebras}) | I \text{ satisfies } E\}$.
Consider the following signatures:

$\Sigma_d = \{[-,-,-]_b, T, F, \bot, [-,-,-]_d, \bot_d\}$,
$\Omega_0 = \{T, F, \bot\} \cup \bigcup_{n \in N} \Lambda_n \cup \bigcup_{n \in N} \Pi_n$ such that $\Omega_0 \cap \Sigma_d = \{T, F, \bot, \bot_d\}$,
$\Omega = \Sigma_d \cup \Omega_0$.

We again adopt a shorthand notation and abbreviate both $[-,-,-]_b$ and $[-,-,-]_d$ to $[-,-,-]$.

Let $E_\Omega = \{(12), (12'), (13), (13')\}$ be the strictness and commutativity axioms for operations and predicates. Let $Ax' = Ax_\Omega \cup E = Ax_d \cup E_\Omega \cup E$. Let E be a set of Ω-equations containing the strictness axioms (12), (12'). Recall that, for an Ω-algebra A, $A_{|\Sigma_d} \in \mathscr{C}_d$ means that:

(i) $[-,-,-]$ is an if-then-else, and
(ii) $A_{Bool} = \{T, F, \bot\}$.
We define:

$$\mathscr{C}^E = \{A \in \mathrm{Alg}^E_\Omega | A_{|\Sigma_d} \in \mathscr{C}_d\},$$
$$\mathscr{C}(\omega)^E = \{A \in \mathrm{Alg}(\omega)^E_\Omega | A_{|\Sigma_d} \in \mathscr{C}_d\},$$
$$\mathscr{C}(d)^E = \{A \in \mathrm{Alg}(d)^E_\Omega | A_{|\Sigma_d} \in \mathscr{C}_d\}.$$

There are inclusions

(vii)
$$\begin{array}{ccc} \mathscr{C}(d)^E & \subseteq & \mathscr{C}(\omega)^E \\ |\cap & & |\cap \\ \mathrm{Alg}(d)^{Ax'}_\Omega & \subseteq & \mathrm{Alg}(\omega)^{Ax'}_\Omega \end{array},$$

(viii)
$$\mathscr{C}^E \subseteq \mathrm{Alg}^{Ax'}_\Omega,$$

(ix)
$$\begin{array}{ccc} \leqq_{\mathscr{C}(d)^E} & \supseteq & \leqq_{\mathscr{C}(\omega)^E} \\ |\cup & & |\cup \\ \leqq_{\mathrm{Alg}(d)^{Ax'}_\Omega} & \supseteq & \leqq_{Ax'} \end{array}.$$

PROPOSITION 24. *In general, the vertical inclusions in* (vii) *and* (ix) *are strict.*

Proof. They are strict for $\Omega = \Sigma_d$ and $E = \{(12), (12')\}$, by Theorem 19 where $\Lambda \cup \Pi = \varnothing$. □

The following theorem, relying on Theorem 9, holds only in the many-sorted case.

THEOREM 25. *Let* $\Omega_0 = \{\bot, T, F\} \cup \Lambda \cup \Pi$, $\Omega = \Omega_0 \cup \{[-,-,-]_b, [-,-,-]_d\}$, $\mathscr{C} = \mathscr{C}_\Omega$, *let E be a set of equations involving only* Ω_0*-terms, and containing the strictness axioms* (12), (12'), *and let* $Ax' = Ax_\Omega \cup E$; *then all inclusions in* (x) *are equalities.*

$$\text{(x)} \qquad \begin{array}{ccccc} \equiv_{\mathscr{C}(d)^E} & \supseteq & \equiv_{\mathscr{C}(\omega)^E} & \supseteq & \equiv_{\mathscr{C}^E} \\ | \cup & & | \cup & & | \cup \\ \equiv_{\mathrm{Alg}(d)_\Omega^{Ax'}} & \supseteq & \equiv_{\mathrm{Alg}(\omega)_\Omega^{Ax'}} & \supseteq & \equiv_{Ax'} \end{array}.$$

Proof. By Theorem 9 it is enough to show $V(\mathscr{C}(d)^E) \supseteq \mathscr{C}^E$. If A is an algebra in $\mathscr{C}^E$, let A^d denote the continuous Ω-algebra obtained by giving to A the discrete ordering. The Λ-operations and the predicates are continuous because of the strictness axioms (12), (12') in E. The if-then-else operations are continuous because of Lemma 15 (§ 4.1). A^d is thus a continuous Ω-algebra and $V(A^d) = A$ whence the theorem. □

An immediate restatement of Theorem 25 is the completeness of the axiom system Ax', e.g., for the class $\mathscr{C}(d)^E$ of discrete Ω-algebras, with if-then-else operations, together with other operations and predicates, satisfying the equations in E, i.e.,

$$\mathscr{C}(d)^E \models t_1 = t_2 \quad \text{iff } Ax' \vdash t_1 = t_2.$$

So far, in the continuous case we have given completeness results under certain restrictions, namely restricting ourselves to finite terms, or considering equality between infinite terms under the assumption that there are no additional operations or predicates. Combined with induction rules, this provides proof systems that are certainly sound. To get completeness in the most general case, i.e., for infinite terms in the presence of additional operations, predicates and equations, the following (as far as we know open) question needs to be answered. We shall see below that an affirmative answer to such a question would provide the desired completeness result for infinite terms in the most general case.

> Is there an effective axiomatization by inequations of all the finite inequalities satisfied by the continuous if-then-else interpretations?

In Theorem 19 we showed that the inequality $[x, y, y] \leqq y$ is satisfied by all the continuous if-then-else interpretations but it does not follow from the equations for if-then-else. This shows that our axioms are not complete for deducing inequalities. Note that a system of inequations that is complete for proving the inequalities between finite terms satisfied by the if-then-else interpretations will obviously deduce the same finite equations as the present one, but may be strictly more powerful to prove not only inequalities $T \leqq T'$ between infinite terms, but even to prove equalities $T = T'$. This is so because the completion of a partially ordered set under limits of chains need not preserve monotonic injections. For instance, given a one-sorted signature consisting of constants $\{a_n : n \geqq 0\}$ and $\{b_n : n \geqq 0\}$ the systems of inequalities $E = \{a_n \leqq a_{n+1} \mid n \geqq 0\} \cup \{b_n \leqq b_{n+1} \mid n \geqq 0\}$ and $E' = E \cup \{a_{2n} \leqq b_{2n} \mid n \geqq 0\} \cup \{b_{2n+1} \leqq a_{2n+1} \mid n \geqq 0\}$ prove exactly the same equalities between finite terms, but E' is strictly stronger than E for infinite terms (see Fig. 5).

Let us now consider the class $\mathscr{C}(\omega)^E$ of all continuous if-then-else interpretations with additional operations and predicates and satisfying equations E. Let Ax'' be a system of inequations that is complete to prove all inequalities between finite terms satisfied by the class $\mathscr{C}(\omega)^E$. Of course such a set of axioms exists, since we could just take the infinite set of all inequalities satisfied by the class, but an effective description

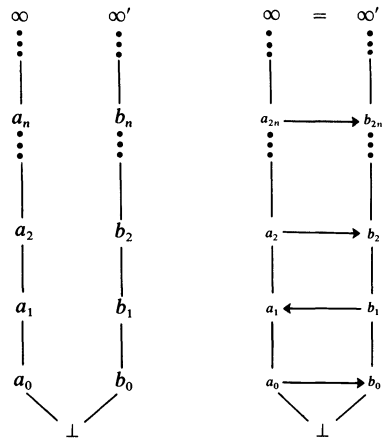


FIG. 5

of a set of axioms is what is needed to have a proof system, as indicated in the question above. We shall now show that, once Ax'' is provided, adding to it the induction rule for infinite terms gives a complete proof system to prove all equalities and inequalities between infinite terms that are satisfied by the class $\mathscr{C}(\omega)^E$. Proving such a completeness result amounts to showing that the class $\mathscr{C}(\omega)^E$ is *algebraic*. In general, a class $\mathscr{A}$ of continuous algebras on an arbitrary signature of operations is called *algebraic* iff whenever $T = \sqcup t_n$, $T' = \sqcup t'_n$ and $T \leqq^{\infty}_{\mathscr{A}} T'$ then for each n there is an m such that $t_n \leqq_{\mathscr{A}} t'_m$. We will also need the concept of absolute and relative first-order axiomatizability of a class $\mathscr{A}$ of continuous algebras. $\mathscr{A}$ is *relatively first-order axiomatizable* if it is the class of all continuous algebras that satisfy a set of first-order axioms. For instance, classes of continuous algebras satisfying a given set of equations or inequations between finite terms are (relatively) first-order axiomatizable. The class $\mathscr{C}(\omega)^E$ is also (relatively) first-order axiomatizable by adding to E the axioms

$$(\forall x\colon \textit{Boolean})(x = T) \vee (x = F) \vee (x = \bot).$$
$$(\forall x, y\colon \textit{Boolean})[T, x, y] = x, [F, x, y] = y, [\bot, x, y] = \bot.$$
$$(\forall x, y\colon \textit{Data})[T, x, y] = x, [F, x, y] = y, [\bot, x, y] = \bot.$$

A class of continuous algebras $\mathscr{A}$ is *absolutely first-order axiomatizable* if it is a class of ordered algebras satisfying a given set of first-order axioms. For instance, the class of all discrete algebras is the class of ordered algebras satisfying the axiom

$$(x < y) \Rightarrow (x = \bot).$$

In general a relatively first-order axiomatizable class of continuous algebras need not be algebraic ([6, Prop. 4.1], [13, Cor. 5.88]). However, an absolutely first-order axiomatizable class of continuous algebras is always algebraic ([6, Thm. 4.2], [13, Cor. 5.76]).

Assuming that an affirmative answer had been already given to the question of effectively axiomatizing by inequations the inequalities between finite terms satisfied by $\mathscr{C}(\omega)^E$, the following completeness result would provide a complete proof system for infinite terms:

THEOREM 26. *If Ax'' is an axiomatization by inequalities of the inequalities between finite terms satisfied by $\mathscr{C}(\omega)^E$, then Ax'' together with the induction rule for infinite terms will provide a complete proof system for all the inequalities $T \leqq T'$ between infinite terms satisfied by $\mathscr{C}(\omega)^E$.*

Proof. We have to show that $\mathscr{C}(\omega)^E$ is algebraic. Since $\mathscr{C}(d)^E$ is absolutely first order axiomatizable, it is algebraic. Thus, it will be enough to show that $\leqq^{\infty}_{\mathscr{C}(d)^E} = \leqq^{\infty}_{\mathscr{C}(\omega)^E}$. Since $\leqq^{\infty}_{\mathscr{C}(d)^E} \supseteq \leqq^{\infty}_{\mathscr{C}(\omega)^E}$, it is enough to show that whenever $T \leqq T'$ is satisfied in $\mathscr{C}(d)^E$, then it is also satisfied in $\mathscr{C}(\omega)^E$. The key observation is that for any continuous if-then-else interpretation $A \in \mathscr{C}(\omega)^E$, the discrete ordering on the underlying unordered algebra $V(A)$ gives a discrete if-then-else interpretation A^d and then the identity function gives a continuous homomorphism $h: A^d \to A$. Of course, an assignment α of values in A to a set X of variables gives rise to the same assignment α^d in A^d and vice versa. Since h is continuous we then have $\bar{\alpha} = h \circ \bar{\alpha}^d$ by initiality of $CT_\Omega(X)$. Thus, whenever $T \leqq T'$ is satisfied by A^d then it is also satisfied by A. Given that all discrete algebras in $\mathscr{C}(d)^E$ are of the form A^d, this proves $\leqq^{\infty}_{\mathscr{C}(d)^E} \subseteq \leqq^{\infty}_{\mathscr{C}(\omega)^E}$ as desired. □

5. Discussion and conclusion. The axiomatics of if-then-else can be traced back to McCarthy [18]. More recently, in [16], an axiom system, very close to Ax, is described, which simplifies if-then-else expressions, possibly many-sorted. This system is not complete, though we conjecture it might be completed by adding a commutative axiom; moreover the system contains axiom schemes, more powerful than mere axioms.

Manes [17] gives a complete set of equational axioms, describing a one-sorted if-then-else which allows for unary predicates: this axiom system uses $\vee$ (or) and $\sim$ (negation) in a very elegant algebraic style.

However, none of the above papers seems to be extendable to the continuous case, and none of them allows for $\perp$, i.e., dealing with possibly undefined terms.

Following ideas in a previous draft of this paper, Benoit [2] has given a completeness theorem for equality among finite terms in the case of one-sorted if-then-else interpretations with additional operations and equations but without continuity,[2] i.e., a one-sorted analogue of our Theorem 9. In the meantime, we have developed a many-sorted approach that makes possible the axiomatization of those contexts (the overwhelming majority in practice) in which different sorts of data other than the Booleans are present; in a one-sorted approach those contexts would be handled very awkwardly if at all. In addition, our approach was extended to continuous algebras in order to handle nonterminating computations.

In [12] slightly more general tests are investigated, corresponding to functions of the form $[p(x), y, z]$ where p is an arbitrary predicate. Formally, define a *test* g as a function $g: D^3 \to D$ such that

$$g(x, y, z) = \begin{cases} y & \text{if } p(x) = T, \\ z & \text{if } p(x) = F, \\ \perp & \text{otherwise.} \end{cases}$$

It was proved in [12] that $\mathscr{C}(g) = \{A | A$ is an ordered algebra such that g_A is a test$\}$ is neither equational, nor even $\mathscr{D}$-equational, i.e., of the form $\mathscr{C}(d)^{E'}$ for some set E' of equations. We however show that, by restricting the test g so that the predicate p is either everywhere defined or the ordering on the domain A is total, $\mathscr{C}(g)$ becomes $\mathscr{D}$-equational; and more precisely $\mathscr{C}(g) = \mathscr{C}(d)^{E'}$, where E' consists of axioms (8) and (9) of Ax together with axioms $g(u, v, v) = v$ and $g(\perp, v, v') = \perp$. This in turn implies that E', together with the axioms of discrete (or total) ordering, form a complete axiom

[2] *Note added February* 25, 1986. For the same case, see also the forthcoming *Equational Bases for If-Then-Else* by A. Mekler and E. Nelson, which corrects a gap in Benoit's proof and discusses decidability issues.

system for deducing all inequations between terms valid in $\mathscr{C}(g)$. Comparing these two approaches, the advantages of the [5] approach are the following:

(1) completeness results can be proven for wider classes of algebras (continuous, discrete, vs. linearly ordered algebras or discrete algebras with everywhere defined predicates for the [12] case);

(2) the [5] axioms are weaker than the [12] axioms, in that the latter allow for disjunction ($d < d' \Rightarrow d = \perp$ or $d = d'$).

On the other hand, the advantages of the [12] approach, besides dealing with a more general kind of test, are the following:

(1) we can prove inequations with the [12] axioms, whereas with [5] axioms we have only been able to prove equations;

(2) another advantage is that the axiomatization result holds, with the same proof, even when the signature contains operations other than g.

In this paper we have presented an algebraic axiomatization of the if-then-else operations that extends the completeness result of [5] to many-sorted algebras with additional operations and predicates and additional equations among those. We have also extended the completeness result for equality of finite terms to continuous algebras. In the case where there are no additional operations completeness has been proved for equality of infinite terms (program schemes). In this way, a complete proof system is obtained in each case. In addition we have shown that the class of continuous if-then-else interpretations with extra operations, predicates and equations among the additional operations and predicates is algebraic. This reduces the search for a complete proof system for infinite terms in the most general case of if-then-else interpretations with additional operations, predicates and equations to the problem of finding an effective axiomatization by inequations of the inequalities between finite terms satisfied by such a class.

Appendix. The following possible notions of if-then-else were introduced by Bloom and Tindell. In this Appendix, we compare them and consider their extensibility to the continuous case. We restrict the discussion to the one-sorted case. Let $[-,-,-]_i$, $i = 1, 3$ be ternary operations, $[-,-,-,-]_i$, $i = 2, 4$, be quaternary operations; let $\perp$, T, F be constants. Define the following classes of algebras:

(1) $\Sigma^1 = \{[-,-,-]_1, T\}$ and
$\mathscr{C}^1 = \{A \text{ in } \mathrm{Alg}_{\Sigma^1} \mid [-,-,-]_1 : A^3 \to A \text{ satisfies a.1}\}$.

(a.1) $$[x, y, z]_1 = \begin{cases} y & \text{if } x = T, \\ z & \text{if } x \neq T. \end{cases}$$

(2) $\Sigma^2 = \{[-,-,-,-]_2\}$ and
$\mathscr{C}_2 = \{A \text{ in } \mathrm{Alg}_{\Sigma^2} \mid [-,-,-,-]_2 : A^4 \to A \text{ satisfies a.2}\}$.

(a.2) $$[x, y, u, v]_2 = \begin{cases} u & \text{if } x = y, \\ v & \text{if } x \neq y. \end{cases}$$

(3) $\Sigma^3 = \{[-,-,-]_3, T, F, \perp\}$ and
$\mathscr{C}_3 = \{A \text{ in } \mathrm{Alg}_{\Sigma^3} \mid [-,-,-]_3 : A^3 \to A \text{ satisfies a.3}\}$.

(a.3) $$[x, y, z]_3 = \begin{cases} z & \text{if } x = F, \\ y & \text{if } x = T, \\ \perp & \text{if } x = \perp \text{ or otherwise.} \end{cases}$$

(4) $\Sigma^4 = \{[-,-,-,-]_4, \perp\}$ and

$$\mathscr{C}^4 = \{A \text{ in } \mathrm{Alg}_{\Sigma^4} \mid [-,-,-,-]_4 : A^4 \to A \text{ satisfies a.4}\}.$$

(a.4)
$$[x, y, u, v]_4 = \begin{cases} u & \text{if } \bot \neq x = y \neq \bot, \\ v & \text{if } \bot \neq x \neq y \neq \bot, \\ \bot & \text{otherwise.} \end{cases}$$

Notes.

(1) $[-,-,-]_i$ is called κ_i in [5] and $\mathscr{C}^1$ (resp. $\mathscr{C}^2$, $\mathscr{C}^3$, $\mathscr{C}^4$) is called $K(3)$ (resp. K, $K(3; \bot)$, $K^+(\bot)$).

(2) $[-,-,-]_3$ is the if-then-else that has been studied in this paper.

(3) $[-,-,-,-]_4$ can be expressed in terms of $[-,-,-]_3$ by adding to the signature Σ^4 the two constants T, F and an equality predicate:

$$p : A^2 \to \{\bot, T, F\} \quad \text{such that}$$
$$p(x, y) = \begin{cases} T & \text{if } \bot \neq x = y \neq \bot, \\ F & \text{if } \bot \neq x \neq y \neq \bot, \\ \bot & \text{otherwise.} \end{cases}$$

LEMMA A.1. *Suppose A is an ordered algebra with least element $\bot$, then:*

(1) *p defined as above is monotonic $\Leftrightarrow$ A has a flat ordering;*

(2) *if A is a flat domain then p is continuous.*

Proof. (1) $\Leftarrow$, and (2) are clear.

(1) $\Rightarrow$: assume $x \leqq y \neq \bot$ then either $x = \bot$ or ($x \neq \bot$ and $T = p(x, x) \leqq p(x, y) \leqq p(y, y) = T$); thus $p(x, y) = T$ and $x = y$. Hence $x \leqq y \neq \bot \Rightarrow x = y$ or $x = \bot$. □

Hence, in the continuous case, case 3 is in a sense more general than case 4, since case 3 admits also non flat domains (T and F have only to be maximal finite elements of the domain, cf., Lemma 15). □

LEMMA A.2. *Neither $[-,-,-]_1$ nor $[-,-,-,-]_2$ are monotone, even on flat domains:*

Proof. For $x \neq \bot$,

$$[\bot, \bot, x]_1 = x \not\leqq [T, \bot, x]_1 = \bot,$$
$$[\bot, \bot, x, x] = x \not\leqq [\bot, x, x, x] = \bot. \qquad \square$$

Therefore, only cases 3 and 4 are possible for if-then-else on ordered or continuous algebras.

We will sketch now how the case $\mathscr{C}^4$ of a quaternary if-then-else corresponding to a test for equality fits into our framework in a way entirely similar to the case $\mathscr{C}$ already studied. Although the case $\mathscr{C}$ is in a sense more general, tests for equality are usually very common, and the $\mathscr{C}^4$ case has the advantage of not requiring reference to the constants T and F.

LEMMA A.3. *Let $\Sigma^4 = \{\bot, [-,-,-,-]\}$, let $\mathscr{C}^4$ denote the class of algebras satisfying* (a.4), *and let $\mathscr{C}^4(\omega)$ be:*

$$\mathscr{C}^4(\omega) = \{I^*, \text{ continuous } \Sigma^4\text{-algebra } \mid V(I) \in \mathscr{C}^4\}.$$

Then the map $A \mapsto A^*$ sending each $A \in \mathscr{C}^4$ to the discrete Σ^4-algebra with same carrier

and operations is a bijection

$$\mathscr{C}^4 \to \mathscr{C}^4(\omega).$$

Proof. If a is an ω-complete c.p.o., a continuous map $[-,-,-,-]: A^4 \to A$ satisfies (a.4) if and only if A is flat (discrete). To see this, check that: (1) the passage $A \mapsto A^*$ described above is well defined, and notice that, (2) if $u > v > \perp$, then

$$[u, u, v, u] = v < [u, v, v, u] = u, \quad \text{a contradiction.} \qquad \square$$

COROLLARY A.4. (1) $\leqq_{\mathscr{C}^4(d)} = \leqq_{\mathscr{C}^4(\omega)}$. (2) *all inclusions in diagram* (xi) *are equalities.*

(xi)
$$\begin{array}{ccccc} \equiv_{\mathscr{C}^4(d)} & \supseteq & \equiv_{\mathscr{C}^4(\omega)} & \supseteq & \equiv_{\mathscr{C}^4} \\ \cup & & \cup & & \cup \\ \equiv_{\mathrm{Alg}(d)^{Ax^4}_{\Sigma^4}} & \supseteq & \equiv_{\mathrm{Alg}(w)^{Ax^4}_{\Sigma^4}} & \supseteq & \overline{Ax^4} \end{array}$$

where Ax^4 is the axiom system

(1) $[\perp, y, u, v] = \perp$,
(2) $[x, x, \perp, \perp] = \perp$,
(3) $[x, x, x, \perp] = x$,
(4) $[x, x, u, v] = [x, x, u, \perp]$,
(5) $[x, y, u, u] = [x, x, [y, y, u, \perp], \perp]$,
(6) $[x, y, u, v] = [x, y, u, [x, x, v, \perp]]$,
(7) $[x, y, u, v] = [y, x, u, v]$,
(8) $[x, y, x, u] = [x, y, y, u]$,
(9) $[x, y, [x, z, u, v], w] = [x, y, [y, z, u, v], w]$,
(10) $[x, y, [x, y, u, v], w] = [x, y, u, w]$,
(11) $[x, y, u, [x, y, v, w]] = [x, y, u, w]$,
(12) $[x, y, [u, v, a, b], [u, v, c, d]] = [u, v, [x, y, a, c], [x, y, b, d]]$,
(13) $[[x, y, u, v], w, b, z] = [x, y, [u, w, b, z], [v, w, b, z]]$.

Proof. The same as for Theorems 16 and 14. $\square$

This corollary extends the completeness result established in [5] for the class $\mathscr{C}^4(K^+(\perp)$ in their terminology) to a completeness result for the corresponding classes of discrete and continuous algebras. Notice that our axiom system is equivalent and almost identical to the one in [5]. Ours is slightly simpler, since their axiom $[x, y, \perp, \perp] = \perp$ has been replaced by our axiom (2).

Acknowledgments. We thank Stephen Bloom and Ralph Tindell for an elegant paper that stimulated our work. Thanks also to Stephen Bloom for his valuable comments. Special thanks to Joseph Goguen, with whom the second author has had many fruitful conversations on the treatment of conditionals that have influenced the approach of this paper, and who has encouraged our project at every stage. We have also benefited from the valuable detailed comments and corrections of Evelyn Nelson, that led to substantial improvements and revisions of a previous draft; we cordially thank her. We also thank Patrice Enjalbert, Emanuela Fachini, Klaus Indermark, Gordon Plotkin, and Martin Wirsing, with whom we had insightful discussions at various stages of the development of this work, and the referees for their helpful suggestions. Dikran Karagueuzian has excellently directed the task of producing the $\mathrm{T_EX}$ version of this paper incorporating changes and revisions of the original manuscript as they occurred with unfailing patience and good humour; we give him our sincere thanks as well as to Trudy Vizmanos and Arlette Dupont who typed the paper.

REFERENCES

[1] H. ANDREKA AND I. NEMETI, *Generalization of variety and quasivariety concepts to partial algebras through category theory*, Dissertationes Mathematicae (Rosprawy Mat.), 204 (1982).

[2] C. BENOIT, *Axiomatisation des tests*, TCS, 35 (1985), pp. 89-107.

[3] G. BIRKHOFF, *On the structure of abstract algebras*, Proc. Cambridge Philosophical Society, 31 (1935), pp. 433-454.

[4] S. BLOOM, *Varieties of ordered alegbras*, J. Comput. System Sci., 13 (1976), pp. 200-212.

[5] S. BLOOM AND R. TINDELL, *Varieties of if $\cdots$ then $\cdots$ else $\cdots$*, this Journal, 12 (1983), pp. 677-707.

[6] B. COURCELLE AND I. GUESSARIAN, *On some classes of interpretations*, J. Comput. System Sci., 17 (1978), pp. 388-413.

[7] B. COURCELLE AND M. NIVAT, *The algebraic semantics of recursive program schemes*, MFCS 78, Lecture Notes in Computer Science 64, Springer-Verlag, Berlin, 1978, pp. 16-30.

[8] J. GOGUEN AND J. MESEGUER, *Completeness of many-sorted equational logic*, SIGPLAN Notices, 16 (1981), pp. 24-32, extended version in Houston J. Math., 11 (1985), pp. 307-334.

[9] J. GOGUEN, J. THATCHER, E. WAGNER AND J. WRIGHT, *Initial algebra semantics and continuous algebras*, J. Assoc. Comput. Mach., 24 (1977), pp. 68-95.

[10] G. GRATZER, *Universal Algebra*, 2nd ed., Springer-Verlag, Berlin, 1979.

[11] I. GUESSARIAN, *Schémas récursifs polyadiques: équivalences et classes d'interprétations*, Thèse d'Etat, Université Paris, 7 (1975).

[12] ———, *Les tests et leur caractérisation syntaxique*, RAIRO Inform. Théor., 11 (1977), pp. 133-156.

[13] ———, *Algebraic Semantics*, Lecture Notes in Computer Science 99, Springer-Verlag, Berlin, 1981.

[14] ———, *Survey on classes of interpretations and some of their applications*, in Algebraic Methods in Semantics, M. Nivat and J. Reynolds, eds., Cambridge University Press, Cambridge, 1985, pp. 383-409.

[15] J. P. JOUANNAUD, ed., *Rewriting techniques and applications*, in Lecture Notes in Computer Science 202, Springer-Verlag, Berlin, 1985.

[16] D. LANKFORD AND D. MUSSER, *On semideciding first-order validity and invalidity*, USC Tech. Report, Univ. Southern Calif., 1978.

[17] E. MANES, *Guard modules*, Algebra Universalis, 21 (1985), pp. 103-110.

[18] J. MCCARTHY, *A basis for a mathematical theory of computation*, in Computer Programming and Formal Systems, Braffort and Hirschberg, eds., North-Holland, Amsterdam, 1963, pp. 33-70.

[19] J. MESEGUER, *On order-complete universal algebra and enriched functorial semantics*, FCT 77, Lecture Notes in Computer Science 56, Springer-Verlag, Berlin, 1977, pp. 294-301.

[20] ———, *Varieties of chain complete algebras*, J. Pure Appl. Algebra, 19 (1980), pp. 347-383.

[21] ———, *A Birkhoff like theorem for algebraic classes of Interpretations*, in Lecture Notes in Computer Science 107, Springer-Verlag, Berlin, 1981, pp. 152-168.

[22] J. MESEGUER AND J. GOGUEN, *Initiality, induction and computability*, in Algebraic Methods in Semantics, M. Nivat and J. Reynolds, eds., Cambridge University Press, Cambridge, 1985, pp. 459-540.

[23] R. MILNER, *Flowgraphs and flowalgebras*, J. Assoc. Comput. Mach., 26 (1979), pp. 794-818.

[24] M. NIVAT, *On the interpretation of recursive polyadic program schemes*, Symposia Mathematica 15, Rome, 1975, pp. 255-281.

[25] M. NIVAT AND J. REYNOLDS, eds., *Algebraic Methods in Semantics*, Cambridge University Press, Cambridge, 1985.

[26] P. PAPPINGHAUS AND M. WIRSING, *Nondeterministic three valued logic: Isotonic and guarded truth-functions*, Studia Logica, 42 (1983), pp. 1-22.

[27] D. SCOTT, *The lattice of flow diagrams*, in Symp. on Semantics of Algorithmic Languages, E. Engeler, ed., Lecture Notes in Mathematics 182, Springer-Verlag, Berlin, 1971, pp. 311-366.

[28] R. SHOSTAK, ed., *7th Intern. Conf. on Autom. Deduction*, Lecture Notes in Computer Science 170, Springer-Verlag, Berlin, 1984.

[29] R. SETHI, *Conditional expressions with equality tests*, J. Assoc. Comput. Mach., 25 (1978), pp. 667-674.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
009

A POLYNOMIAL-TIME ALGORITHM TO FIND THE SHORTEST CYCLE BASIS OF A GRAPH*

J. D. HORTON†

Abstract. Define the length of a basis of the cycle space of a graph to be the sum of the lengths of all cycles in the basis. An algorithm is given that finds a cycle basis with the shortest possible length in $O(m^3n)$ operations, where m is the number of edges and n is the number of vertices. This is the first known polynomial-time algorithm for this problem. Edges may be weighted or unweighted. Also, the shortest cycle basis is shown to have at most $3(n-1)(n-2)/2$ edges for the unweighted case. An $O(mn^2)$ algorithm to obtain a suboptimal cycle basis of length $O(n^2)$ for unweighted graphs is also given.

Key words. graph theory, cycle basis, minimization, polynomial-time algorithm, fundamental set of circuits

AMS(MOS) subject classifications. Primary 68Q25; secondary 05C38, 68R10, 05C35

1. Introduction. A cycle basis is used to examine the cyclic structure of a graph. For example, many algorithms use cycle bases to list all simple cycles in a graph (Mateti and Deo [11], Sysło [16]) or look for the longest cycle (Dixon and Goodman [6]). Cycle bases have been also used to solve electrical networks since the time of Kirchoff (see, for example, Chua and Chen [1]). Similar techniques have been used in the analysis of algorithms (Knuth [9]). Recently surveyors have also suggested using cycle bases to look for blunders in survey data (Steeves [13], Cribb et al. [2]). Some other uses are mentioned by Deo et al. [4].

In many of the above algorithms, the amount of work is dependent on the cycle basis chosen. A basis with shorter cycles in it may speed up the algorithm. Thus one would like to be able to calculate the shortest cycle basis possible, preferably quickly. The first references to this problem that I have found are Stepanec [14] and Zykov [17], which are in Russian. Hubicka and Sysło [8] showed that the Stepanec-Zykov algorithm does not work on all graphs, and proposed other algorithms. However, Kolasinska [10] gave an example on which the Hubicka-Sysło algorithm fails to obtain the shortest cycle basis. Both Cribb et al. [2] and Steeves [13] also asked the question of finding the shortest cycle basis. Steeves proposed a different algorithm, but examples have been found on which it did not find the shortest cycle basis either. Some other papers on the topic are Deo [3] and Sysło [15].

A second related problem first proposed in Hubicka and Sysło [8] has fared even worse. How can you find a spanning tree whose fundamental set of cycles has total shortest length? Deo, Prabhu and Krishnamoorthy [4] have shown that this question is *NP*-complete.

2. Definitions. In this paper, graphs are finite, undirected, without loops or multiple edges. Since the cycle space of a graph is the direct sum of the cycle spaces of its 2-connected components, we are concerned only with 2-connected graphs. An edge is denoted by the (unordered) pair of its endpoints. Each edge e of a graph is weighted with a real number $w(e)$, which extends to a weight function on all sets of edges. The unweighted case, or equivalently the case in which all weights are one, does not seem to lead to any simplification of the algorithm. Weights are not allowed to be negative.

* Received by the editors April 18, 1985; accepted for publication (in revised form) July 2, 1986. This research was supported by Natural Sciences and Engineering Research Council of Canada grant A5376.

† School of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada E3B 5A3.

A *cycle* is any set of edges of a graph in which each vertex of the graph is incident with an even number of edges. Two cycles, C and D, can be added to form their sum, $C+D$, using the set symmetric difference operation, $(C \cup D)-(C \cap D)$. Equivalently, one can identify a cycle with its incidence vector, and cycle-addition as vector addition over the integers modulo 2. The set of all cycles is closed under addition, forming the *cycle space*, and can be generated by the set of simple cycles. Subgraphs are identified with their edge-sets so that a simple circuit is also a cycle. The *weight* or length of a cycle basis is the sum of the weights of its cycles.

In much of the following we will be dealing with an arbitrary graph. Let the number of vertices in the graph be n; let the number of edges in the graph be m; let ν be the dimension of the cycle space. Then $\nu = m-n+1$, since we are assuming the graph to be 2-connected.

3. Theoretical results. A cycle in a shortest cycle basis of a graph must have certain properties, as will be shown in this section. Only simple cycles that have these properties need to be considered by the algorithm. We start by citing some known results.

THEOREM 1. *If* **B** *is a cycle basis for a graph, C is a cycle in* **B**, *and $C = C_1 + C_2$, then either* $\mathbf{B}-\{C\}\cup\{C_1\}$ *is a cycle basis or* $\mathbf{B}-\{C\}\cup\{C_2\}$ *is a cycle basis.*

Another result that is well known, and was used by Hubicka and Sysło [8], is that the shortest cycle through an edge is always in the cycle basis. Although I do not use this fact, I include it for completeness.

THEOREM 2. *If e is an edge, and C is a shortest cycle through e, then C is in some minimum cycle basis. Moreover, any minimum cycle basis must contain some shortest cycle through e.*

The major observation that I make is that in any cycle in a minimum cycle basis, any two vertices are connected by a shortest path.

THEOREM 3. *Let x and y be two vertices of G, and let* **B** *be a cycle basis of G. Let P be a path from x to y. Then any cycle C of* **B** *that contains x and y can be exchanged for either a cycle that includes the path P, or a cycle that excludes one of x and y.*

Proof. Let P_1 and P_2 be the two paths in C joining x and y. Assume $P \neq P_1$ and $P \neq P_2$, for otherwise C does not need to be exchanged. Define $C_1 = P + P_1$ and $C_2 = P + P_2$. Then C_1 and C_2 are cycles, and $C = C_1 + C_2$. By Theorem 1, C can be exchanged for either C_1 or C_2. If the exchanged C_i is not a simple cycle, then C_i can be exchanged for one of its simple cycles, and that simple cycle cannot include both x and y.

COROLLARY. *If* **B** *is a minimum cycle basis, and P is the unique shortest path from x to y, then every cycle of* **B** *that contains x and y must contain the path P.*

Proof. In the proof of Theorem 3, $w(C_i) \leqq w(P) + w(P_i) < w(P_1) + w(P_2) = w(C)$. Hence if C does not contain P, it can be exchanged for a shorter cycle, and so **B** would not be minimum.

Note that the corollary assumed that the shortest path from x to y was unique. In general, there can be several paths between any two vertices that are shortest. What Theorem 3 and its corollary imply is that in any minimum cycle basis, any two vertices of a cycle must be joined by a shortest path. Moreover, if there is a preferred shortest path from x to y, one can exchange all cycles in **B** for cycles that contain this preferred shortest path, and still retain a minimum cycle basis.

Define $d(x, y)$ to be the length of a shortest path from x to y.

THEOREM 4. *Let x be any vertex of any cycle C in a minimum cycle basis. Then there is an edge $\{y, z\}$ in C such that C consists of a shortest path from x to y, a shortest path from x to z, and the edge $\{y, z\}$.*

Proof. Let $C = (x_0, x_1, \cdots, x_{n-1}, x_n)$, where $x = x_0 = x_n$. Define $P_i = (x, x_1, x_2, \cdots, x_i)$ and $Q_i = (x, x_{n-1}, \cdots, x_i)$. By the corollary to Theorem 3, $d(x, x_i) = w(P_i)$ or $w(Q_i)$. Let $y = x_i$, where i is the largest subscript such that $d(x, x_i) = w(P_i)$. Let $z = x_{i+1}$. Then $C = P_i + \{x_i, x_{i+1}\} + Q_{i+1}$.

Theorem 4 gives a strong condition on the cycles in a minimum cycle basis, especially if all the shortest paths are unique. In the latter case all basis cycles can be found by considering for each vertex-edge pair $(x, \{y, z\})$ the cycle $C(x, y, z) = P(x, y) + P(x, z) + \{y, z\}$, where $P(v, u)$ is the shortest path from v to u. Obviously there are at most nm such cycles. The set of all cycles forms a matroid, so that the greedy algorithm can be used to extract the minimum cycle basis.

Unfortunately, this last greedy step is necessary. A counterexample to the converse of Theorem 4 is given in Fig. 1. The shortest cycle basis consists of the inner faces. However, the outer face also satisfies the condition of Theorem 4. Thus one cannot hope to eliminate the check for independence of the cycles unless another condition is found that cycles in a minimum cycle basis must satisfy.

4. The algorithm. The outline of the algorithm can be given as follows:

(1) Find a minimum path $P(x, y)$ between each pair of vertices x, y.
(2) For each vertex v and edge $\{x, y\}$ in the graph, create the cycle $C(v, x, y) = P(v, x) + P(v, y) + \{x, y\}$, and calculate its length. Degenerate cases in which $P(v, x)$ and $P(v, y)$ have vertices other than v in common can be omitted.
(3) Order the cycles by weight.
(4) Use the greedy algorithm to find the minimum cycle basis from this set of cycles.

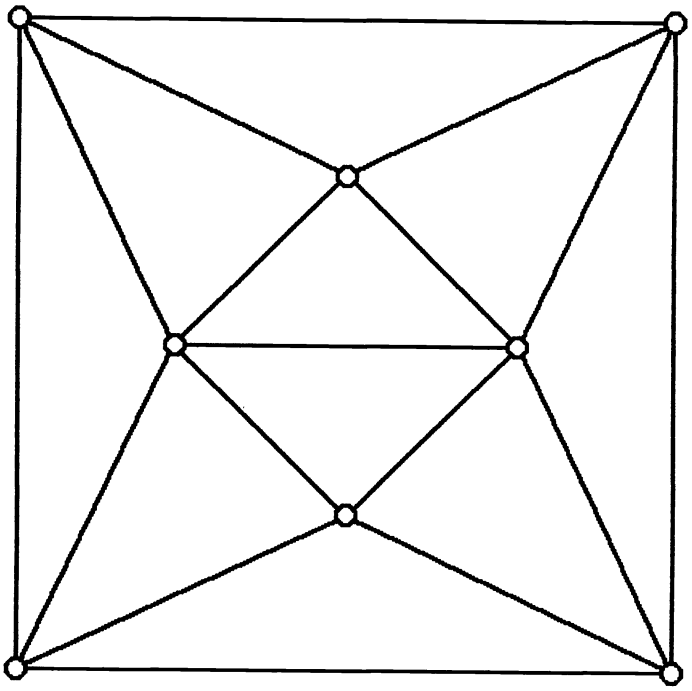


FIG. 1. *A counterexample to the converse of Theorem* 4.

That this algorithm works is not entirely obvious. It would be obvious if the shortest paths were all unique, or even if all subpaths of any chosen shortest path $P(x, y)$ were also a chosen subpath. The former could be guaranteed by perturbation or lexicographic techniques. The latter could be done by post-processing the output from the shortest path algorithm. However, these complications are unnecessary if all edges have positive weight.

THEOREM 5. *If all edges have nonnegative weight, the algorithm finds a minimum cycle basis no matter what shortest path $P(x, y)$ is chosen for each pair of vertices x and y.*

Proof. Let **B** be a minimum cycle basis. We shall change **B** by exchanging cycles in **B** with cycles containing the chosen shortest paths. First order the vertices $x_1, x_2, \cdots, x_n$. For each vertex x_i, order the $n-1$ paths $P_{ij} = P(x_i, y_{ij})$ in nondecreasing order of length. Now process all $n(n-1)$ paths in the lexicographic order of the subscripts, $P_{11}, P_{12}, P_{13}, \cdots, P_{21}, \cdots, P_{n\,n-1}$. For each P_{ij}, exchange each cycle in the basis that contains vertices x_i and y_{ij} with a cycle that contains the path P_{ij}. Note that only simple cycles can be formed, since otherwise we would obtain a basis of shorter length. By Theorem 3 and its corollary, a minimum cycle basis is obtained. We must show that the resultant basis $\mathbf{B}'$ obtained after processing all $n(n-1)$ paths is a subset of the cycles obtained from step (2) of the algorithm.

Let C be any cycle in $\mathbf{B}'$. Let x_i be the vertex of C with the largest subscript. Then C must have been in the basis at the end of the processing of $P_{i1}, P_{i2}, \cdots, P_{i\,n-1}$. By Theorem 4 there is an edge $\{y, z\}$ in C such that C consists of a shortest path from x_i to y, say P_1, a shortest path from x_i to z, say P_2, and the edge $\{y, z\}$. The last path P_{ij} that may have affected C must have been either $P(x_i, y)$ or $P(x_i, z)$, because all other vertices in C are closer to x. Assume that $P(x_i, y)$ was the last. Then either $P(x_i, y) = P_1$ or $P(x_1, y) = P_2 + \{y, z\}$.

First assume $P_1 = P(x_i, y)$. The only other paths that could have been processed after $P(x_i, z)$ was processed, and that changed the cycle C, are paths $P(x_i, w)$ where w is on a shortest path from y to x_i. But this could not have affected P_2, the path from z to x_i, since if it had, a nonsimple cycle would have been formed. Hence $P_2 = P(x_i, z)$. Thus C would have been found by the algorithm.

The alternative is that z is on $P(x_i, y)$. Thus C consists of two shortest paths from y to x_i. Let w be the other vertex neighboring y in the cycle C. Then the path from w to x_i is a shortest path, because it is a subpath of a shortest path. Thus w can replace z throughout the whole argument, and this second case reduces to the first case.

5. Analysis and implementation ideas. A quick analysis indicates that (4) is the critical step:

Step (1) takes $O(n^3)$ operations,

Step (2) takes $O(mn^2)$ operations,

Step (3) takes $O(a \log(a))$ operations,

Step (4) takes $O(nk)$ operations,

where a is the number of cycles found in Step (2), and k is the number of operations to decide whether a cycle is independent of another given set of independent cycles or not.

Step (1) can use your favorite shortest path algorithm, e.g., Floyd [7] or Dijkstra [5]. Step (2) is straightforward, but can be improved. The first factor m counts the number of edges, the factor n counts the number of vertices, and a second n is a bound on the time to write out the cycle. However, the cycles do not have to be calculated immediately but only when needed. It is necessary to calculate the length

of the cycle, which is the sum of the lengths of two shortest paths and an edge if the case is not degenerate. A degenerate case will never be needed in the processing later anyway. Thus Step (2) can be reduced to $O(nm)$ operations.

The bound of nm on the number of cycles can also be improved. If Dijkstra's shortest path algorithm is used [5], which requires each edge to have nonnegative length, then a shortest-path spanning tree is grown at each vertex. A vertex and an edge in its shortest-path spanning tree will never create a cycle. Hence $a \leqq n\nu$. This improvement is important only if the graph is very sparse. A larger practical gain may be made if the chosen shortest paths are such that all subpaths of a chosen path are also chosen. This will automatically occur if all shortest paths are unique. In this case, each cycle C will be generated once for each vertex, and hence the number of cycles is cut by at least a factor of three. Of course, a method of detecting duplicate circuits is then required, which at first glance seems to require calculating the circuits. One method to avoid calculating all the cycles is to first order the vertices, then remove any shortest path $P(x, y)$ that passes through a vertex z that precedes x in the ordering. Again, for this operation to allow the algorithm to work, it is sufficient that the set of chosen shortest paths satisfy the subpath property that all subpaths of chosen paths are also chosen paths. Each cycle will now be generated only once. The largest number of cycles that can be generated by the algorithm using this technique is $\sum_{k=2}^{n} k(k-1)/2 - k+1 = n(n-1)(n-2)/6$. This bound is realized by K_n, the complete graph on n vertices with unweighted edges. The algorithm must generate all triangles, and $n(n-1)(n-2)/6$ is precisely the number of triangles in K_n.

Another method to reduce the number of cycles is to test whether the cycles produced by Step (2) satisfy Theorem 4, which can be done in $O(k)$ operations, where k is the number of vertices in the cycle.

Step (3) is a noncritical step. Conceivably it would be made to run faster by taking into account the order in which the cycles are produced. However, its bound of $O(n\nu \log n)$ is certainly insignificant in comparison to Step (2) or (4).

Step (4) is critical. One method to perform this step is to consider the cycles as the rows of a 0-1 matrix. The columns correspond to the edges of the graph; the rows are the incidence vectors of the cycles. Gaussian elimination using elementary row operations over the integers modulo 2 can be applied to the matrix. Each row should be processed in turn, in the order of the weights of the cycles. When enough independent cycles have been found, the process can stop. Each row can be processed in $O(mr)$ operations, where m is the number of columns, and r is the number of rows above the row being processed. Since r is bounded by ν, since only a complete set of independent cycles are needed to be kept, and since the total number of cycles is $O(n\nu)$, Step (4) takes $O(m\nu^2 n)$ operations.

As the bound for Step (4) dominates the functions for the other steps, the algorithm as a whole has worst case upper bound of $O(m\nu^2 n) = O(m^3 n) = O(n^7)$ operations.

6. The length of the shortest cycle basis. Deo et al. [4] raised another question concerning fundamental sets of circuits in an unweighted graph. What is the maximum possible total length (weight) of the shortest (lightest) fundamental set of circuits of a graph? They conjectured that the total length is at most $3n^2/2$. I am able to show that they were correct at least for minimum cycle bases.

THEOREM 6. *The length of a minimum cycle basis of an unweighted graph with n vertices is at most* $3(n-1)(n-2)/2$.

Proof. The theorem is certainly true for $n = 1$, 2, or 3. Assume $n \geqq 3$. Let G be a graph with $n+1$ vertices. If G is not two-connected, then let v be a cut vertex of G.

Let H be a component of $G-v$, and let $G-H$ be the rest the graph, including the vertex v. Let H have k vertices; then $G-H$ has $n-k+1$ vertices. By an induction hypothesis, the subgraph of G induced by H and v combined has a cycle of length less than or equal to $3k(k-1)/2$, while $G-H$ has a cycle basis of length $3(n-k)\times(n-k-1)/2$ or less. Since the union of these two cycle bases forms a cycle basis for G, and $k(k-1)+(n-k)(n-k-1)\leqq n(n-1)$, the result is true for G. Therefore, we may assume G is two-connected. Let x be a vertex of minimal degree in G. As an induction hypothesis, assume that $G-x$ satisfies the theorem. Thus, if **B** is a minimum cycle basis of $G-x$, $w(G)\leqq 3(n-1)(n-2)/2$. I intend to add simple cycles that pass through x to **B** until a cycle basis for G is found. The total length of the cycles will be less than or equal to $3n-3$. This will complete the proof to the theorem, since $\sum_{k=3}^{n} 3k-6=3(n-1)(n-2)/2$. Let the degree of x be k. Then $k-1$ cycles have to be added to B to form a cycle basis for G and all these cycles must pass through x.

What other constraints are there on the choice of these cycles? Consider another graph which I shall call the x-network, in which vertices are called *nodes* and edges are called *branches*, to distinguish the original graph from the x-network. The edges adjacent to x are the nodes of the x-network, and two nodes are connected by branches if and only if a chosen cycle uses both edges. The set of chosen cycles together with **B** forms an independent set if and only if the network does not contain a cycle. As the x-network has k nodes and $k-1$ branches, the x-network must be a tree. Conversely, if the x-network is a tree then the set of chosen cycles together with **B** forms a cycle basis for G.

Call the other end points of the edges incident with x: $x_1, x_2, \cdots, x_k$. A chosen cycle will consist of a path in $G-x$, from some x_i to some other x_j, and two edges adjacent to x. I choose these $k-1$ paths from a spanning tree T of $G-x$.

Consider the special case when $k=n$. Then it is easy to pair off the x_i's, which are all the vertices of the tree. Pair any vertex of degree one in the spanning tree with its only neighbor, and remove the vertex of degree one from the tree. Continue in this way until all vertices are paired. All paths are of length 1. Then the $n-1$ cycles are all triangles, so the total weight to be added to **B** is $3n-3$.

The next case to be considered is for $k<n/2+1$. The paths joining the x_i in the tree T can be chosen so that no edge of T is in more than two paths. Consider, for example, an embedding of T in the plane, and also consider the closed path consisting of the face of T. Each edge occurs twice. Take the subpaths of this path that go from one x_i to the next x_j. In the corresponding network, these paths correspond to branches. The network is of course connected. In fact it contains at least k branches; and hence at least one cycle. A spanning tree in this network corresponds to a required set of paths in T. At least one of the paths have to be removed, and the total length of the chosen paths is at most $2(n-1)-1=2n-3$. Then the total length of the circuits to be added to **B** is at most $2n-3+2(k-1)<3n-3$.

The last case to be considered is for k such that $v/2+1\leqq k<v$. We use induction on k, the minimum degree of G. Since x, a vertex, is of minimum degree, the degree of every vertex in $G-x$ is at least $n/2$. Since there are $k\geqq n/2+1$ vertices adjacent to x, any vertex x_i adjacent to x is also adjacent to some other vertex x_j adjacent to x. Remove x_i from G. By an induction hypothesis, there are $k-2$ independent cycles in $G-x_i$ that pass through x, of total length at most $3n-6$. Adding the triangle (x, x_i, x_j) to this set satisfies the condition of the theorem.

Deo et al. [4] showed that the bound of Theorem 6 cannot be improved, by noting that for the complete graph on n vertices, the star tree has a set of fundamental circuits of length $3(n-1)(n-2)/2$.

The ideas used in the proof to Theorem 6 have some interesting extra implications, which are important enough to include as another theorem. Define a set of simple cycles through a vertex x to be *locally-independent-at-x* if the corresponding network, as developed in the proof to Theorem 6, does not contain a cycle. Thus, locally-independent-at-x implies independent, but not conversely. A maximal locally-independent-at-x set of cycles consists of degree $(x)-1$ cycles in a two-connected graph. In the corresponding x-network, the set corresponds with a spanning tree. Each cycle in a minimum-weight maximal locally-independent-at-x set consists of two edges $\{x, y\}$, and $\{x, z\}$, and the shortest path from y to z in $G-x$. If we label all the branches in the x-network with the length of the corresponding cycle, then a minimum-weight maximal locally-independent-at-x set of cycles corresponds to, and has the weight of, a minimum spanning tree in the network.

THEOREM 7. *Given a vertex x in a graph G, a minimum-weight maximal locally-independent-at-x set of circuits is a subset of a minimum cycle basis.*

Proof. Let B be a basis, and let C be a cycle, not in B, that is in the minimum-weight maximal locally-independent-at-x set of cycles. Let C correspond to the branch a in the network. C is the sum of cycles in a subset of B, say **A**. The cycles in **A** that contain x, together with C, sum to zero on the edges at x, and so must correspond to a cycle in the x-network. Hence there must be a set of cycles in **A** that, together with C, correspond to a cycle through a in the network. That cycle must contain a branch b that is not in the minimum spanning tree of the x-network. The branch b corresponds to a cycle D in the original graph. Then $B-\{D\}\cup\{C\}$ is also a cycle basis. But $w(C)=w(a)\leqq w(b)\leqq w(D)$, because a is in the minimum spanning tree, and b is not.

7. A fast algorithm for a suboptimal cycle basis. The proof to Theorem 6 also leads to an algorithm faster than the exact algorithm, that is guaranteed to give a cycle basis with no more than $3(n-1)(n-2)/2$ edges in the unweighted case.

(1) Choose a vertex x of minimum degree.
(2) Find all shortest paths in $G-x$ between neighbors of x.
(3) Find a minimal spanning tree in the corresponding weighted network.
(4) Form the shortest path cycles corresponding to the branches in the tree.
(5) Add to the set of cycles from Step 4 a cycle basis for $G-x$ (obtained by recursion).

A quick analysis of this algorithm is: Step (1), $O(1)$; Step (2), $O(dn^2)$, where $d=$ degree of x; Step (3), $O(n^2)$; Step (4), $O(dn)$; Step (5), multiplies by $O(n)$. The dominating term comes from Step (2), and is $O(dn^2)$. Given Step (5), the whole algorithm is $O(n^2$ (sum of degrees)$)=O(n^2m)$.

To find a minimum cycle basis, it is not sufficient to calculate the minimum-weight maximal locally-independent-at-x set of circuits at all vertices. Figure 2 gives a counter-

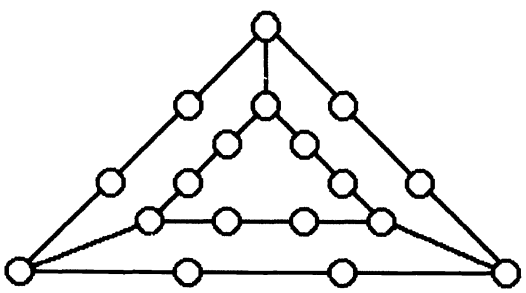


FIG. 2. *Shortest cycle basis with a circuit that is not locally independent.*

example. The three trapezoidal faces include all minimum-weight maximal locally-independent-at-x sets of circuits at any vertex x. But one of the triangular faces must be added to form a minimum-weight cycle basis.

Deo et al. [4] also suggest several heuristic algorithms to search for a spanning tree with a minimum fundamental set. In all cases, the algorithms may generate a basis of weight $O(n^3)$, as can be shown by graphs based on the graph used by Paton [12] for a similar purpose.

8. Conclusions. The algorithm that is outlined in §§ 4 and 5 is the first polynomial-time algorithm that actually generates the shortest possible cycle basis. However, it is a very expensive algorithm, $O(m^3n)$. Now a cycle basis is generally not wanted for its own sake, but to be used as input for a later algorithm. The importance of a short cycle basis is to reduce the amount of work that has to be done in this later algorithm. Other published algorithms, although much faster than my algorithm, may not produce a cycle basis that is even close in length. Much of the published work has concentrated on trying to generate a tree with a short fundamental set of circuits. These algorithms, although fast, are not guaranteed to arrive at a good answer. Indeed, the exact solution to the shortest fundamental set of circuits has been shown to be *NP*-complete. If the follow-up algorithm is an expensive one, such as listing all simple cycles, an exact algorithm may provide a much better starting point for it than other algorithms.

There is considerable room for improvement on this problem. It is quite likely that some more constraints on cycles in the shortest cycle basis can be found. Nor does the algorithm use the fact that certain cycles must be in the basis, as pointed out in Theorems 2 and 7, for example. Such a marriage between this algorithm and the earlier ones could make for a fast algorithm by eliminating checks for independence. Even better would be to find an exact set of conditions that a member of the shortest cycle basis must satisfy.

REFERENCES

[1] L. O. CHUA AND L. K. CHEN, *On optimally sparce cycle and coboundary basis for a linear graph*, IEEE Trans. Circuit Theory, CT-20 (1973), pp. 495-503.

[2] D. W. CRIBB, R. D. RINGEISEN AND D. R. SHIER, *On cycle bases of a graph*, Congr. Numer., 32 (1981), pp. 221-229.

[3] N. DEO, *Minimum length fundamental cycle set*, IEEE Trans. Circuits and Systems, 26 (1979), pp. 894-895.

[4] N. DEO, G. M. PRABHU AND M. S. KRISHNAMOORTHY, *Algorithms for generating fundamental cycles in a graph*, ACM Trans. Math. Software, 8 (1982), pp. 26-42.

[5] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269-271.

[6] E. T. DIXON AND S. E. GOODMAN, *An algorithm for the longest cycle problem*, Networks, 6 (1976), pp. 139-149.

[7] R. W. FLOYD, *Algorithm* 97: *shortest path*, Comm. ACM, 5 (1962), p. 345.

[8] E. HUBICKA AND M. M. SYSŁO, *Minimal bases of cycles of a graph*, in Recent Advances in Graph Theory, Proceedings of the symposium held in Prague, June 1974, M. Fiedler, ed., Academia Praha, Prague, 1975, pp. 283-293.

[9] D. E. KNUTH, *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, MA, 1968, pp. 363-370.

[10] E. KOLASINSKA, *On a minimum cycle basis of a graph*, Zastos. Mat., 16 (1980), pp. 631-639.

[11] P. MATETI AND N. DEO, *On algorithms for enumerating all circuits of a graph*, this Journal, 5 (1976), pp. 90-99.

[12] K. PATON, *An algorithm for finding a fundamental set of cycles of a graph*, Comm. ACM, 12 (1969), pp. 514-518.

[13] P. STEEVES, *Numerical processing of horizontal control data-economization by automation*, Ph.D. thesis, University of New Brunswick, 1984.

[14] G. F. STEPANEC, *Basis systems of vector cycles with extremal properties in graphs*, Uspekhi Mat. Nauk, 19 (1964), pp. 171-175. (In Russian.)
[15] M. M. SYSŁO, *On cycle bases of a graph*, Networks, 9 (1979), pp. 123-132.
[16] ———, *An efficient cycle vector space algorithm for listing all cycles of a planar graph*, this Journal, 10 (1981), pp. 797-808.
[17] A. A. ZYKOV, *Theory of Finite Graphs*, Nauka, Novosibirsk, 1969. (In Russian.)

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
010

ON EFFICIENT SIMULATIONS OF SYSTOLIC ARRAYS BY RANDOM-ACCESS MACHINES*

OSCAR H. IBARRA† AND MICHAEL A. PALIS‡

Abstract. We give efficient simulations of systolic arrays by unit-cost random-access machines (RAM's). For example, we show that a one-dimensional systolic array operating in linear time can be simulated by a RAM in $O(n^2/\log^2 n)$ time. For the case of a two-dimensional systolic array, the simulation time is $O(n^3/\log^{3/2} n)$.

Key words. systolic arrays, unit-cost random-access machines, Turing machines

AMS(MOS) subject classifications. 68Q05, 68Q10, 68Q80

1. Introduction. In the design of parallel algorithms, we are usually concerned with finding ways of parallelizing a known sequential algorithm so that it maps nicely onto a given parallel machine. However, it is also common to use sequential computers to simulate parallel machines and, therefore, it is interesting to analyze how efficiently this can be done.

In this paper, we give efficient simulations of systolic arrays [14], [15] by unit-cost random-access machines (RAM's) [1]. For example, we show that a one-dimensional systolic array operating in linear time can be simulated by a RAM in $O(n^2/\log^2 n)$ time. For the two-dimensional case, the simulation time is $O(n^3/\log^{3/2} n)$. As an application, we show that every language accepted by a one-way nondeterministic one-counter acceptor [6] can be recognized by an on-line RAM in time $O(n^2/\log^2 n)$, which improves the best previously known bound of $O(n^2/\log n)$. (The $O(n^2/\log n)$ time bound follows from the fact that a one-counter language can be recognized by an on-line multitape Turing machine in $O(n^2)$ time [6], which, in turn, can be simulated by an on-line RAM in $O(n^2/\log n)$ time [5].) For quasi-realtime two-counter languages, we give an on-line RAM recognition algorithm that takes $O(n^3/\log^{3/2} n)$ time.

We obtain our results using a technique that converts a systolic array into an equivalent single-tape Turing machine (TM) that makes unidirectional sweeps on the worktape [9]–[11]. We exploit this property of unidirectional sweeps to give an efficient simulation of the TM by a RAM using a variant of the 4-Russians method [2]. Without the characterization, it seems difficult to apply the 4-Russians method directly on the systolic array because of the two-way communication between processors.

The paper is organized as follows. In § 2, we define the sequential machines that characterize one- and two-dimensional systolic arrays. In § 3, we prove the RAM simulation results, and also mention simulations by other sequential models such as single-tape TM's and auxiliary push-down automata (APDA's). Section 4 gives some concluding remarks.

We end this section with some definitions concerning systolic arrays. To simplify the presentation, we only consider systolic arrays consisting of finite-state machines used as language recognizers (better known as cellular automata or iterative arrays) [3], [4], [17]. However, the results also apply to more general array models.

* Received by the editors February 28, 1985; accepted for publication (in revised form) June 24, 1986. This research was supported in part by National Science Foundation grants MCS-8304756 and DCR-8604603.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455. This author's research was also supported in part by a John Simon Guggenheim Memorial Foundation Fellowship.

‡ Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104.

A one-dimensional systolic array (1DSA) is shown in Fig. 1. It consists of a linear array of identical processors (nodes) that operate synchronously at discrete time steps by means of a common clock (not shown in the figure). The input string $a_1a_2 \cdots a_n$, where a_i is in the finite input alphabet Σ, is fed serially to the leftmost node, which is also called the accepting node (circled twice in the figure). Symbol a_i, $1 \leqq i \leqq n$, is received by the accepting node at time $i-1$; after time $n-1$, it receives the blank symbol λ which is not in Σ. The state and outputs of a node at time t are functions of its state and inputs at time $t-1$. At time 0, each node is in a distinguished start state q_0, with its outputs set to λ. It remains in this state until it receives a non-λ input from at least one of its neighbors.

If the 1DSA has $S(n)$ processors, where n is the length of the input string, then it is said to be $S(n)$-space bounded. Input $a_1a_2 \cdots a_n$ is accepted by the 1DSA in time $T(n)$ if and only if the accepting node enters an accepting state within $T(n)$ steps. We assume that $T(n) \geqq n$; otherwise, some suffix of the input cannot affect the outcome of the computation.

The two-dimensional counterpart (2DSA) of the array described above is shown in Fig. 2. In this case, we say that the array is $S(n)$-space bounded if it has $S(n)$ by $S(n)$ processors. (Thus, an $S(n)$-space bounded 2DSA has $S^2(n)$ processors.) The accepting node is the node at the origin. All other definitions are the same as in a 1DSA.

Notation. 1DSA($S(n)$, $T(n)$) will denote the class of $S(n)$-space bounded 1DSA's operating in $T(n)$ time. 2DSA($S(n)$, $T(n)$) denotes the corresponding two-dimensional class. 1DSA(∞, $T(n)$) and 2DSA(∞, $T(n)$) will denote unbounded 1DSA's and 2DSA's, respectively, operating in $T(n)$ time. (An unbounded 1DSA is one which is unbounded to the right; an unbounded 2DSA is one which is unbounded at the top and to the right.) Let C_1 and C_2 be classes of machines. $C_1 \subseteq C_2$ means that any machine in C_1 can be simulated by a machine in C_2.

For the purposes of this paper, we only consider arrays with the following space and time bounds:

(1) 1DSA(∞, $T(n)$) and 2DSA(∞, $T(n)$),

(2) 1DSA(n, $T(n)$), 1DSA(n^2, $T(n)$) and 2DSA(n, $T(n)$).

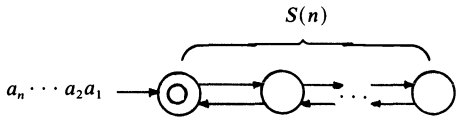


FIG. 1. *A* 1DSA.

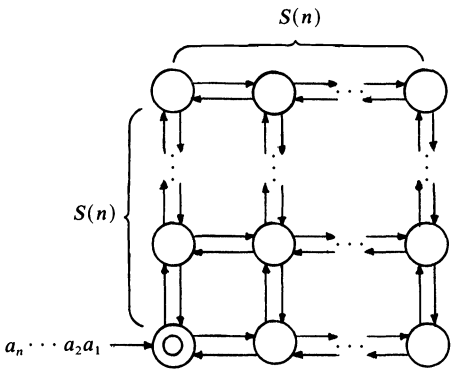


FIG. 2. *A* 2DSA.

2. Sequential machine characterizations of systolic arrays. We now define the sequential machines that characterize the systolic arrays. For one-dimensional systolic arrays, the basic sequential machine model (1DSM) is shown in Fig. 3. It consists of a semi-infinite worktape (bounded at the right by a special endmarker \$), and a finite-state control with an input terminal from which it receives the serial input $a_1 a_2 \cdots a_n$. We assume that after a_n is read the machine reads λ's from the input terminal. Each a_i, $1 \leqq i \leqq n$, comes from a finite input alphabet Σ, which does not include λ. A 1DSM operates as follows. Initially, all cells of the worktape contain λ's. The read-write head (RWH) makes successive (right-to-left) sweeps on the worktape, between the right boundary marker \$ and the first λ encountered. A sweep begins with the RWH scanning \$ and with the machine in a distinguished start state q_0. The machine then moves the RWH to the left and, as it does, rewrites the symbols scanned by the RWH by non-λ symbols and changes states (except into q_0) until the RWH scans a λ. It then reads an input symbol, rewrites λ by a non-λ symbol, returns to state q_0, and resets the RWH to the right boundary marker \$ to begin the next sweep. Figure 4 shows the worktape "profile" of a 1DSM for 5 sweeps. Each row of the profile represents the contents of the worktape after completing the sweep. Note that a 1DSM does not write λ's on its worktape.

The 1DSM accepts input $a_1 a_2 \cdots a_n$ if and only if it writes an accepting symbol on its worktape at the end of a right-to-left sweep. The 1DSM has sweep complexity $T(n)$ on an input of length n if and only if it accepts the input within $T(n)$ complete sweeps. We assume that $T(n) \geqq n$. 1DSM($T(n)$) denotes the class of 1DSM's with sweep complexity $T(n)$.

A variation of the basic 1DSM model called 1DSMV is shown in Fig. 5. A 1DSMV is similar to a 1DSM except that is enters a special state $t \neq q_0$ after $2n$ sweeps [10]. In succeeding sweeps, the right boundary marker \$ is propagated one cell to the left.

It is known that 1DSA's and 1DSM's are equivalent in the following sense [10]:

(1) 1DSA(∞, $T(n)$) = 1DSM($T(n)$),

(2) 1DSA(n, $T(n)$) = 1DSMV($T(n)$).

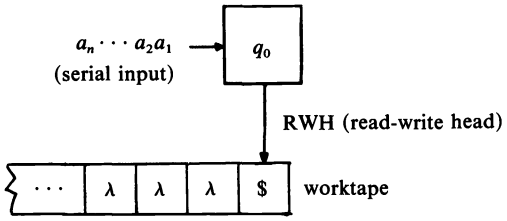


FIG. 3. *A* 1DSM.

Input	Worktape Profile						Sweep
						\$	0
a_1					Z_1^0	\$	1
a_2				Z_2^0	Z_1^1	\$	2
a_3			Z_3^0	Z_2^1	Z_1^2	\$	3
λ		Z_4^0	Z_3^1	Z_2^2	Z_1^3	\$	4
λ	Z_5^0	Z_4^1	Z_3^2	Z_2^3	Z_1^4	\$	5

FIG. 4. *The worktape profile of a* 1DSM ($n = 3$).

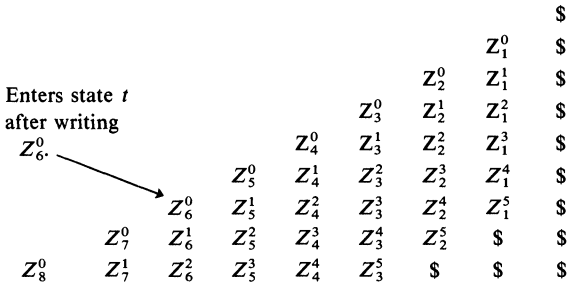


FIG. 5. *The worktape profile of a* 1DSMV ($n=3$).

Similar characterizations are known for two-dimensional systolic arrays. The basic machine model (2DSM) is shown in Fig. 6. It is similar to a 1DSM, except that it has a two-dimensional worktape, which is infinite to the left and to the bottom. The right end of each row is marked by a special endmarker \$. Initially, all cells contain λ's.

The 2DSM operates just like a 1DSM except that it sweeps the worktape in two dimensions. The worktape profile of the 2DSM for five complete sweeps is shown in Fig. 7. A sweep begins and ends with the machine in state q_0 and with the read-write head (RWH) scanning the \$ of the first row. During the sweep, the RWH scans the (non-λ) rows of the worktape from top to bottom and then creates a new bottom row. The 2DSM then returns to state q_0 and resets the RWH to the \$ of the first row to begin the next sweep. Figure 8 illustrates the movement of the RWH during one complete sweep. While scanning a row from right-to-left, the 2DSM can change states and can rewrite the symbols it scans (see the solid lines in the figure). When the RWH reaches the left end of the row, the machine returns to state q_0 and brings the RWH to the \$ of the next row, without rewriting the worktape nor changing states (see the dotted lines in the figure). When all previously created rows have been rewritten, the 2DSM creates a new bottom row which extends one cell to the left more than the previous row. An input symbol is read before rewriting the leftmost λ of the new bottom row (circled in Fig. 7). The 2DSM then returns to state q_0 and resets the RWH to the \$ of the first row. The RWH has the following added capability: when scanning a cell in some row, it can also read (but not rewrite) the content of the cell in the previous row directly above the scanned cell. (We may assume the existence of a zeroth row consisting of \$'s so that when the RWH is scanning a cell in the first row, it reads a \$ in the zeroth row.)

The 2DSM accepts input $a_1 a_2 \cdots a_n$ if and only if it writes an accepting symbol in the leftmost cell of the bottom row of some sweep (i.e., on a cell where an input symbol is read). The 2DSM has sweep complexity $T(n)$ on an input of length n if and only if it accepts the input within $T(n)$ complete sweeps. Again, we assume that $T(n) \geqq n$. 2DSM($T(n)$) denotes the class of 2DSM's with sweep complexity $T(n)$.

A variation of the 2DSM called 2DSMV is shown in Fig. 9. A 2DSMV is similar to a 2DSM except that it enters a special state $t \neq q_0$ after $2n$ sweeps. In succeeding sweeps, the topmost row is replaced by \$'s. Also, starting at the $(2n+1)$st row, \$ is propagated one cell to the left more than the previous row.

Again, the following relationship is known [11]:

(1) 2DSA(∞, $T(n)$) = 2DSM($T(n)$),

(2) 2DSA(n, $T(n)$) = 2DSMV($T(n)$).

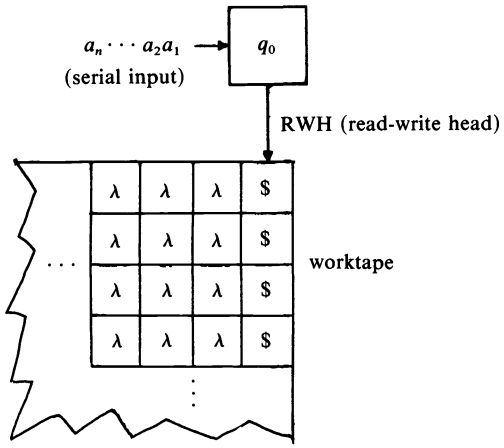


FIG. 6. *A* 2DSM.

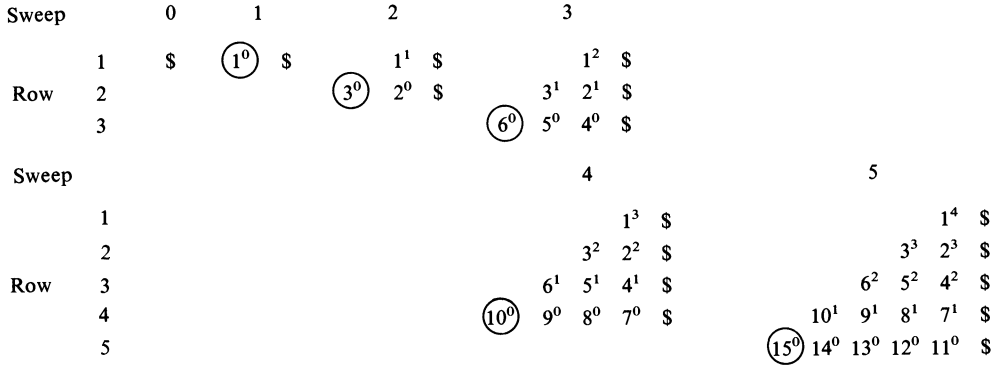


FIG. 7. *The worktape profile of a* 2DSM ($n = 3$).

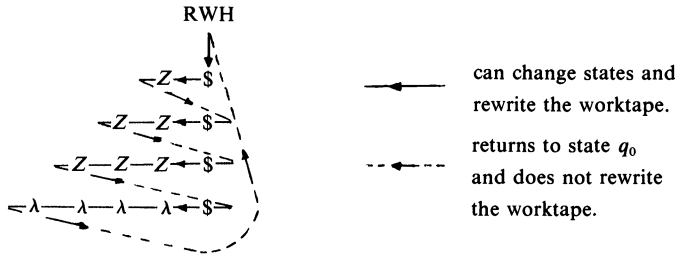


FIG. 8. *Movement of the* RWH *of a* 2DSM *during one complete sweep.*

3. RAM simulations of systolic arrays. Before proving the RAM results, we first analyze how systolic arrays can be simulated by other sequential models such as single-tape TM's and APDA's. We begin with the following lemma which states that two-dimensional systolic arrays can be efficiently simulated by their one-dimensional counterparts. (Part 1 was mentioned without proof in [16].)

LEMMA 1. (1) $2\text{DSA}(\infty, T(n)) \subseteq 1\text{DSA}(\infty, T^2(n))$,

(2) $2\text{DSA}(n, T(n)) \subseteq 1\text{DSA}(n^2, nT(n))$.

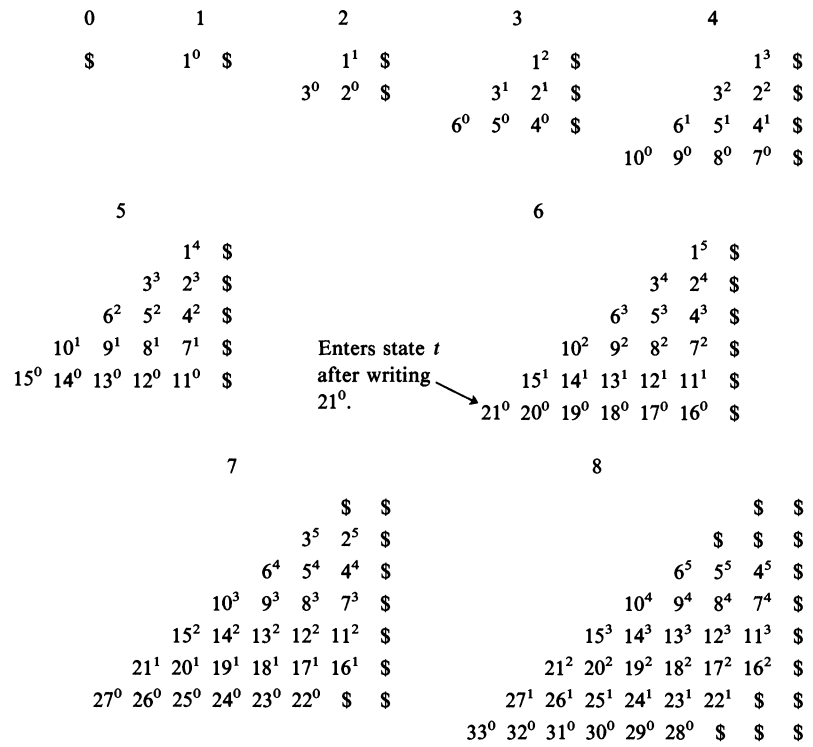


FIG. 9. *The worktape profile of a* 2DSMV ($n = 3$).

Sketch of Proof. The lemma is proved using the characterizations. Informally, the idea is to create blocks of size 1, 2, 3, $\cdots$ on the worktape of the 1DSM and simulate the rows of the 2DSM on these blocks. The details of the proof can be found in [12]. □

A single-tape TM is a TM with one read-write tape which initially contains the input string [8]. While it is obvious that any machine in 1DSA(n, $T(n)$) can be simulated by a single-tape TM in $nT(n)$ time, the characterizations reveal that the result holds even if the TM is oblivious and can only rewrite the worktape when scanning it in one direction, say right-to-left (henceforth, a *simple* TM). Thus, we have the following (the second part follows from Lemma 1).

PROPOSITION 1. (1) *Any machine in* 1DSA(∞, $T(n)$) [1DSA(n, $T(n)$)] *can be simulated by a simple* TM *in time* $T^2(n)$ [$nT(n)$].

(2) *Any machine in* 2DSA(∞, $T(n)$) [2DSA(n, $T(n)$)] *can be simulated by a simple* TM *in time* $T^4(n)$ [$n^3T(n)$].

Since context-free languages (CFL's) can be accepted by machines in 2DSA(n, $2n$) [9], [13], we get the following corollary. (A weaker result without the requirement that the TM is simple was earlier shown in [18].)

COROLLARY 1. *Every context-free language can be recognized by a simple* TM *in time* n^4.

An APDA is a single-tape TM augmented by a pushdown store [8]. It is interesting that with an extra tape which is operated as a pushdown store, the simulation time of Proposition 1-(2) can be reduced to $T^3(n)$ [$n^2T(n)$]. Moreover, the size of the pushdown store is bounded by $T(n)$ [n]. We omit the proofs, as they follow easily from the

two-dimensional characterizations. (The idea is to simulate the rows of the 2DSM or 2DSMV in successive blocks on the APDA worktape, and to use the pushdown store to move a copy of one block to the next block. See [12] for details.)

PROPOSITION 2. (1) *Any machine in* 2DSA(∞, $T(n)$) *can be simulated by an* APDA *in* $T^3(n)$ *time. Moreover, the space requirements of the worktape and the pushdown store are* $T^2(n)$ *and* $T(n)$, *respectively.*

(2) *Any machine in* 2DSA(n, $T(n)$) *can be simulated by an* APDA *in* $n^2T(n)$ *time. Moreover, the space requirements of the worktape and the pushdown store are* n^2 *and* n, *respectively.*

It is known that every context-free language (CFL) can be accepted by a multitape TM with a read-only input tape and two worktapes in n^3 time [8]. Both worktapes require n^2 space, and there does not seem to be a way of reducing the space by simply modifying the algorithm. However, from Proposition 2-(2) and the fact that CFL's can be accepted by machines in 2DSA(n, $2n$) [9], [13], we have the following stronger result.

COROLLARY 2. *Every* CFL *can be recognized by an* APDA *in time* n^3. *Moreover, the space requirements of the worktape and the pushdown store are* n^2 *and* n, *respectively.*

On random-access machines, the simulations are faster. In [7], it was shown that a multitape TM operating in time $T(n) \geqq n \log n$, where $\log T(n)$ is computable by a RAM in $O(T(n)/\log T(n))$ time, can be simulated by a RAM in $O(T(n)/\log T(n))$ time. It follows from Proposition 1-(1) that an unbounded 1DSA operating in $T(n)$ time can be simulated by a RAM in time $O(T^2(n)/\log T(n))$. For the two-dimensional case, the simulation time is $O(T^3(n)/\log T(n))$, by Proposition 2-(1). We show that, in fact, the simulations can be made faster: An unbounded 1DSA operating in $T(n)$ time can be simulated by a RAM in time $O(T^2(n)/\log^2 T(n))$. For the two-dimensional case, the simulation time is $O(T^3(n)/\log^{3/2} T(n))$.

We use the sequential machine characterizations to prove the RAM simulation results. The reader will find that the results would be much more difficult to prove without the characterizations.

THEOREM 1. (1) *Let* $\log T(n)$ *be computable by a* RAM *in* $O(T^2(n)/\log^2 T(n))$ *time. Then, any machine in* 1DSA(∞, $T(n)$) *can be simulated by a* RAM *in time* $O(T^2(n)/\log^2 T(n))$.

(2) *Let* $\log T(n)$ *be computable by a* RAM *in* $O(nT(n)/\log^2 T(n))$ *time. Then, any machine in* 1DSA(n, $T(n)$) *can be simulated by a* RAM *in time* $O(nT(n)/\log^2 T(n))$.

Proof. For (1), it is sufficient to show that a 1DSM M with sweep complexity $T(n)$ can be simulated by a RAM in $O(T^2(n)/\log^2 T(n))$ time. Let Q, Γ and Σ be the state set, worktape alphabet and input alphabet, respectively, of M. The computation profile G of M is shown pictorially in Fig. 10(a). (Ignore the dotted lines for the moment.) Let q_j^i be the state which M enters after writing Z_j^i. We shall refer to a pair (q_j^i, Z_j^i) as an *entry* of G.

We use a partitioning and precomputing technique similar to the 4-Russians method [2] (see also [7], [19]). Note that because M operates in sweeps and rewrites the worktape only in one direction, the computation profile can be easily partitioned; without the characterization, this would be difficult because of the two-way communication between processors. Let d be a positive integer (the value of d will be determined later). Suppose we divide G into square and triangular blocks of size d (i.e., d entries on each side or face), such that the entries lying on the face of a block overlap with those on the faces of the neighboring blocks (see dotted lines in Fig. 10(a)). Consider any square or triangular block and label its faces as an input or output face according to the labeling rule of Fig. 11. For a square block we can represent the contents of each of its faces by a string of length d consisting of pairs (q, Z), where $q \in Q$ and

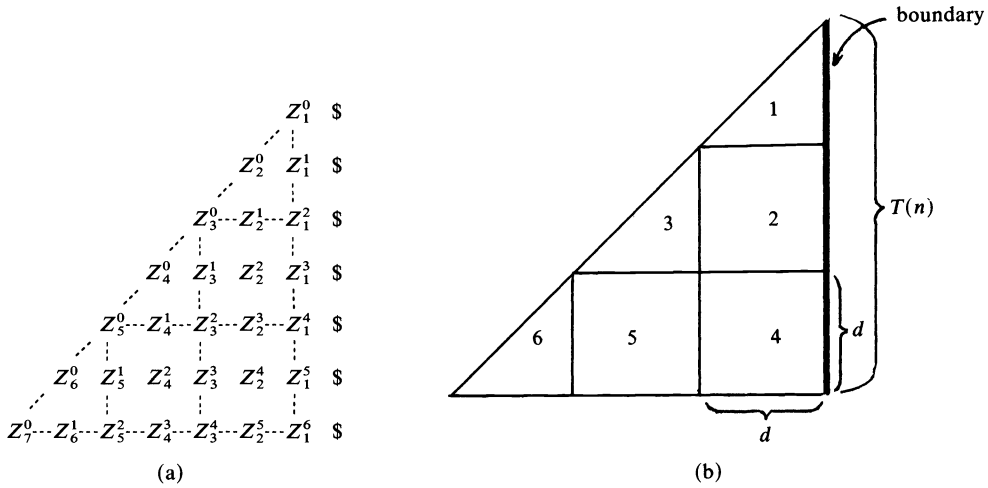


FIG. 10. *Partitioning the profile G of a* 1DSM *into blocks* ($d = 3$).

$Z \in \Gamma$. Denote the strings as s_1, s_2, s_3, and s_4, as shown in Fig. 11(a). We shall refer to $s_{in} = (s_1, s_2)$ and $s_{out} = (s_3, s_4)$ as the input and output arguments, respectively, of the square block. Clearly, once s_{in} is known, s_{out} can be determined by direct simulation of M. Similarly, for a triangular block, let t_1, t_2 and t_3 be strings of length d representing the contents of its faces, as shown in Fig. 11(b). In addition, let $w \in \Sigma^+$ be a string of length d representing a segment of the input string of M. We shall refer to $t_{in} = (t_1, w)$ and $t_{out} = (t_2, t_3)$ as the input and output arguments, respectively, of the triangular block. Again, given t_{in}, t_{out} can be determined. Now suppose that the strings along the "boundary" (see Fig. 10(b)) have been precomputed. Then, we can fill the blocks in the order shown in the figure, since the input arguments of a block to be filled are the output arguments of previously filled blocks.

We can save time in the computation if, for each type of block (square or triangular), we precompute the values of the output arguments for all possible values of the input arguments. That is, we precompute a mapping which maps input arguments of a block to its output arguments. A convenient way of storing this information is to encode the string arguments as integers, and build two trees (one for each type of block) with labeled edges, such that the labels on a path from the root to a leaf read as input arguments and pointers from the leaves refer to the corresponding output arguments. For both square and triangular blocks, the number of input arguments is two. Hence, a tree look-up takes constant time on a unit-cost RAM.

The total number of distinct s_{in}'s and t_{in}'s is $(|Q| \times |\Gamma|)^{2d} + (|Q| \times |\Gamma|)^d \times |\Sigma|^d$, which is $O(c^d)$ for some constant c. Since each block has $O(d^2)$ entries, the time to precompute the mapping is $O(c^d \times d^2)$.

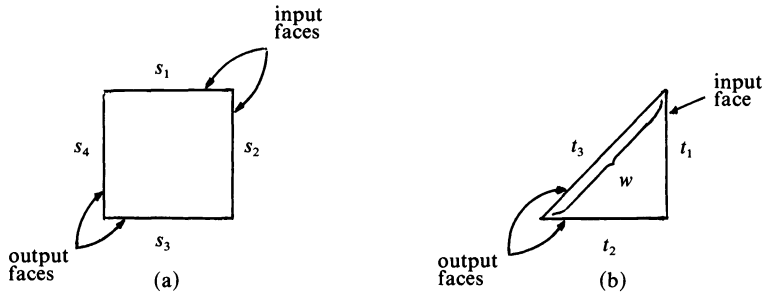


FIG. 11. *Labeling the faces of a square and a triangular block.*

The blocking scheme divides G into $O(T^2(n)/d^2)$ blocks. A tree look-up takes constant time. Hence, including precomputation, the total simulation time is $O(c^d \times d^2 + T^2(n)/d^2)$. Take $d = \log T(n)/\log c$. This yields a time complexity of $O(T^2(n)/\log^2 T(n))$.

The proof of (2) is similar. In this case, we need only show that a 1DSMV operating in $T(n)$ time can be simulated by a RAM in time $O(nT(n)/\log^2 T(n))$. The simulation is performed as in (1) except that the profile of the 1DSMV is divided into blocks of size d in the manner shown in Fig. 12. The time for the simulation is then $O(c^d \times d^2 + nT(n)/d^2)$. For $d = \log T(n)/2 \log c$, this is $O(T^{1/2}(n) \log^2 T(n) + nT(n)/\log^2 T(n))$. Hence, the simulation time is $O(nT(n)/\log^2 T(n))$. □

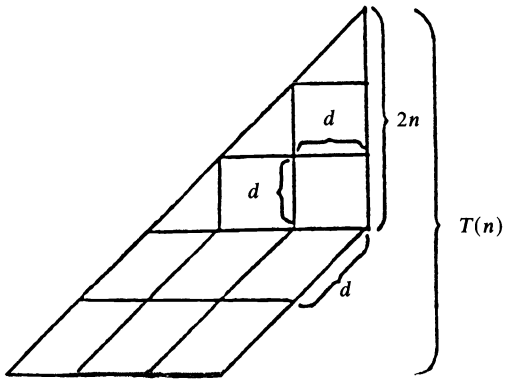


FIG. 12. *Partitioning the profile of a* 1DSMV *into blocks.*

For the two-dimensional case, we have the following.

THEOREM 2. (1) *Let* $\log T(n)$ *be computable by a* RAM *in* $O(T^3(n)/\log^{3/2} T(n))$ *time. Then, any machine in* 2DSA$(\infty, T(n))$ *can be simulated by a* RAM *in time* $O(T^3(n)/\log^{3/2} T(n))$.

(2) *Let* $\log T(n)$ *be computable by a* RAM *in* $O(n^2T(n)/\log^{3/2} T(n))$ *time. Then, any machine in* 2DSA$(n, T(n))$ *can be simulated by a* RAM *in time* $O(n^2T(n))/\log^{3/2} T(n))$.

Proof. The proofs are similar to that of Theorem 1. In this case, the profile of a 2DSM is divided into three-dimensional blocks of size d, where each block covers d rows and d columns of the tape, and d sweeps of the machine. Again, partitioning is easily accomplished because the dependencies of the entries of the computation profile are unidirectional in each dimension. In contrast, partitioning the computation of the 2DSA directly (e.g., using a space-time diagram) is quite difficult because of the two-way communication between processors. As in the one-dimensional case, there are two types of blocks: cube and pyramidal. The number of entries in each block is $O(d^3)$, and on each face the number of entries is $O(d^2)$. Thus, the total number of possible blocks is $O(c^{d^2})$ for some constant c, and the time needed for the precomputation is $O(c^{d^2} \times d^3)$. The profile is divided into $O(T^3(n)/d^3)$ blocks and a tree look-up takes constant time. Hence, including precomputation, the time needed by the simulation is $O(c^{d^2} \times d^3 + T^3(n)/d^3)$. Take $d = (\log T(n)/\log c)^{1/2}$. The time complexity would then be $O(T^3(n)/\log^{3/2} n)$. The same analysis can be used to prove (2). □

Theorems 1 and 2 can be extended to the case when the machines are *on-line.* An on-line systolic array is one which operates as follows: At each step, the node which receives the input string has the option to read or not to read an input symbol. Moreover,

for all i, $1 \leqq i \leqq n$, the array must accept or reject the segment $a_1 \cdots a_i$ of the input string before reading a_{i+1}. An on-line SM and an on-line RAM are similarly defined.

The proofs for the online machines are similar to those of Theorems 1 and 2. The only difficulty is that, since the machines are online, they cannot know the value of n (the length of the input string) in advance. This difficulty is overcome as follows. (We only illustrate this for the case of an unbounded 1DSA.) The RAM first assumes that $n=2$ and simulates the 1DSM as in the proof of Theorem 1. If the 1DSM reads the third input symbol, the RAM starts over with the assumption that $n=4$. In general, the RAM assumes that $n=2^k$ and if the (2^k+1)st input symbol is read, it simulates the 1DSM from the start with the assumption that $n=2^{k+1}$. Whenever the RAM restarts the simulation, the partitioning uses blocks of different size and a new tree of precomputed blocks is prepared. (A similar technique was used in [5].) Now, unlike the simulation in the proof of Theorem 1 where both square and triangular blocks were computed using tree look-ups, triangular blocks are computed by direct simulation of the 1DSM. This is necessary because the 1DSM is online, and hence, the RAM would not know in advance the number of input symbols the 1DSM would read when the triangular block is processed. For a given n, this will add an additional $O(T(n) \times \log T(n))$ time steps to the simulation time of $O(T^2(n)/\log^2 T(n))$. Hence, the time for one simulation remains $O(T^2(n)/\log^2 T(n))$. Let n be such that $2^k < n < 2^{k+1}$. Then the time spent by the RAM is $\sum_{i=1}^{k+1} O(T^2(2^i)/\log^2 T(2^i))$, which is $O(T^2(n)/\log^2 T(n))$, if $T(n)$ is well behaved and bounded by a polynomial.

The proofs for the other cases are similar. Thus, we have the following.

THEOREM 3. *Theorems* 1 *and* 2 *hold for the online versions of the machines, provided that* $T(n)$ *is well behaved and bounded by a polynomial.*

Theorem 3 can be used to obtain efficient recognition algorithms for nondeterministic k-counter languages [6] by on-line RAM's. A nondeterministic k-counter acceptor (k-NCA) is a one-way nondeterministic finite-state automaton with a read-only input tape (containing the input string $a_1 a_2 \cdots a_n$) and k counters, each of which can be incremented by -1, 0, or 1 on each atomic move. Each counter can be independently tested for a zero count at each atomic move. We assume that, at each step, at most one of the k counters is incremented or decremented. Initially, the machine is in a distinguished start state q_0, the input head is scanning a_1, and all counters are 0. The k-NCA accepts the input string if and only if it is in an accepting state, the input head is to the right of a_n, and all counters are 0. A k-NCA is said to be *quasi-realtime* if the input head makes at most a bounded number of consecutive ε-moves. It is said to be *realtime* if the input head moves one cell to the right at each atomic step (i.e., no ε-moves). It is easy to show that quasi-realtime k-NCA's can be made to operate in realtime. In fact, it is known that any 1-NCA (which is not necessarily realtime) can be made realtime [6].

Using SM's to implement a simulation by dynamic programming, the following result can be shown (see [12]).

PROPOSITION 3. (1) *Every one-counter language can be recognized by an on-line unbounded* 1DSA *with time complexity n.*

(2) *Every quasi-realtime two-counter language can be recognized by an on-line unbounded* 2DSA *with time complexity n.*

Thus, from Theorem 3 and the above proposition, we have the following corollary:

COROLLARY 3. (1) *Every one-counter language can be recognized by an on-line* RAM *in time* $O(n^2/\log^2 n)$.

(2) *Every quasi-realtime two-counter language can be recognized by an on-line* RAM *in time* $O(n^3/\log^{3/2} n)$.

4. Conclusion. We have given efficient simulations of one- and two-dimensional systolic arrays by RAM's, using a technique that converts the systolic array into an equivalent Turing machine that makes unidirectional sweeps on the worktape. The characterizations, as well as the simulation results, can be extended to other types of systolic arrays (e.g., arrays with parallel input), as well as to arrays of higher dimensions. For example, an unbounded k-dimensional systolic array can be simulated by a RAM in $O(T^{k+1}(n)/\log^{(k+1)/k} T(n))$ time. If the array is n-space bounded, the time is $O(n^k T(n)/\log^{(k+1)/k} T(n))$.

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] V. L. ARLAZAROV, E. A. DINIC, M. A. KRONROD AND I. A. FARADZEV, *On economical construction of the transitive closure of a directed graph*, Soviet Math. Dokl., 11 (1970), pp. 1209-1210.

[3] S. COLE, *Real-time computation by n-dimensional iterative arrays of finite-state machines*, IEEE Trans. Comput., C-18 (1969), pp. 346-365.

[4] C. DYER, *One-way bounded cellular automata*, Inform. Control, 44 (1980), pp. 261-281.

[5] Z. GALIL, *Two fast simulations which imply some fast string matching and palindrome-recognition algorithms*, Inform. Process. Lett., 4 (1976), pp. 85-87.

[6] S. GREIBACH, *A note on the recognition of one-counter languages*, Revue Francaise d-Automatique, Informatique et Recherche Operartionnelle, 1975, pp. 5-12.

[7] J. HOPCROFT, W. PAUL AND L. VALIANT, *On time versus space and related problems*, Proc. 16th Ann. IEEE Symp. on Foundations of Computer Science, 1975, pp. 57-64.

[8] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[9] O. H. IBARRA, S. M. KIM AND M. A. PALIS, *Designing systolic algorithms using sequential machines* (extended abstract), Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science, 1984, pp. 46-55; full paper in IEEE Trans. Comput., C-35 (1986), pp. 531-542.

[10] ———, *Some results concerning linear iterative (systolic) arrays*, J. Parallel Distributed Comput., 2 (1985), pp. 182-218.

[11] O. H. IBARRA AND M. A. PALIS, *Two-dimensional systolic arrays: characterizations and applications*, Technical Report 85-1, Department of Computer Science, University of Minnesota.

[12] ———, *On efficient simulations of systolic arrays by sequential machines*, Technical Report 85-3, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN.

[13] S. R. KOSARAJU, *Speed of recognition of context-free languages by array automata*, this Journal, 4 (1975), pp. 331-340.

[14] H.-T. KUNG, *Let's design algorithms for VLSI systems*, Proc. Caltech Conference on VLSI: Architecture, Design, Fabrication, 1979, pp. 65-90.

[15] ———, *Why systolic architectures?* Computer, 15 (1982), pp. 37-46.

[16] J. SEIFERAS, *Iterative arrays with direct central control*, Acta Inform., 8 (1977), pp. 177-190.

[17] A. SMITH, III, *Real-time language recognition by one-dimensional cellular automata*, J. Comput. System Sci., 6 (1972), pp. 233-253.

[18] K. TANIGUCHI AND T. KASAMI, *A note on computing time for the recognition of context-free languages by a single-tape Turing machine*, Inform. Control, 14 (1969), pp. 278-284.

[19] J. VAN LEEUWEN AND M. NIVAT, *Efficient recognition of rational relations*, Inform. Process. Lett., 14 (1982), pp. 34-38.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987
© 1987 Society for Industrial and Applied Mathematics
011

ASYMPTOTIC EXPANSIONS FOR A CLOSED MULTIPLE ACCESS SYSTEM*

C. KNESSL†‡, B. J. MATKOWSKY†, Z. SCHUSS§ AND C. TIER¶

Abstract. A closed multiple access system composed of M sources and a single service facility is analyzed. The closed network is modeled as a finite source $M/G/1$ queueing system. We consider systems with a large number of sources (i.e. $M \gg 1$) and we assume that the mean time required to process an individual request is short ($O(1/M)$). We then construct asymptotic approximations to the stationary distribution of the number of requests in the service facility by using the method of matched asymptotic expansions. We give formulas for the first and second moments of the number of requests, for all traffic intensities.

Key words. multiple access system, $M/G/1$ queue, asymptotic expansions, queueing theory, singular perturbations, performance evaluation

1. Introduction. We consider a closed multiple access system composed of M sources and a single service facility. When a source requests service, the request "enters" the service facility and receives service under some nonpreemptive scheduling algorithm (e.g., FCFS, etc.). While the request is in the service facility, the corresponding source is in a blocked state and cannot generate any new requests. When a response is received, i.e., when the request has been served, the source enters a "think" or request generation state. We assume that each source acts independently and that the think time for each source is exponentially distributed with mean $1/\lambda$. In addition, we assume that the service facility has one server and that the service time requested has a probability density function $b(x)$ with mean $1/\mu$. This closed network is a finite-source $M/G/1$ queueing system. When the model represents a closed computer system, the M sources represent terminals while the service facility represents some device or the flow-equivalent center of a subnetwork of devices [1], [2]. This model is also equivalent to the machine-servicing or machine interference model, where the M sources represent machines and the service facility represents a repairman [3]-[5].

The purpose of this paper is to compute approximations to the stationary probabilities for $N(t)$, the number of requests in the service facility, for $M \gg 1$. To motivate our results, we first consider the special case of an $M/M/1$ system in which the service times are exponentially distributed with mean $(M\mu)^{-1}$. One readily constructs the stationary probability $p(n)$ (i.e., $\Pr\{N(t)=n\}$), that n requests are in the service facility as (see [1], [2])

$$p(n) = p_0 \frac{M!}{(M-n)!}\left(\frac{\lambda}{M\mu}\right)^n, \qquad n = 0, 1, \cdots, M, \tag{1.1}$$

where p_0 is found by normalization to be

$$p_0 = 1 \Big/ \sum_{n=0}^{M} \frac{M!}{(M-n)!}\left(\frac{\lambda}{M\mu}\right)^n. \tag{1.2}$$

* Received by the editors June 17, 1985; accepted for publication (in revised form) July 8, 1986. This research was supported in part by National Science Foundation grants DMS-84-06110 and DMS-85-01535, the U.S. Israel Binational Science Foundation and the Israel Academy of Sciences.

† Department of Engineering Sciences and Applied Mathematics, The Technological Institute, Northwestern University, Evanston, Illinois 60201.

‡ The work of this author was partially supported by an IBM Fellowship.

§ Department of Mathematics, Tel Aviv University, Tel Aviv, Israel.

¶ Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, Chicago, Illinois 60680.

Note that we have scaled the service rate by the number of sources M (cf. § 2). When the number of sources is large, i.e., $M \gg 1$, simpler formulas are obtained by replacing (1.1) and (1.2) by their asymptotic approximations. An important aspect of the solution is that the asymptotic approximations behave differently for different values of n and for different values of λ/μ. Thus, we define three regions as follows:

$$\begin{aligned}
&\text{Region (i):} && n = O(1), \quad M \gg 1,\\
&\text{Region (ii):} && n = O(M), \quad M - n = O(M),\\
& && \begin{pmatrix} \xi = O(1), \quad 1 - \xi = O(1) \\ \text{where } \xi = n/M \end{pmatrix}, \quad M \gg 1,\\
&\text{Region (iii):} && M - n = O(1), \quad M \gg 1,
\end{aligned}$$

and expand $p(n)$ given by (1.1) for $M \gg 1$ in each of the three regions to obtain

$$p(n) \sim p_0 \begin{cases} \left(\dfrac{\lambda}{\mu}\right)^n, & \text{Region (i)},\\ \dfrac{\exp\left(-M[\xi \log(\mu/\lambda) + \xi + (1-\xi)\log(1-\xi)]\right)}{\sqrt{1-\xi}}, \quad \xi = \dfrac{n}{M}, & \text{Region (ii)},\\ \sqrt{2\pi M}\,\dfrac{M^{M-n}e^{-M}}{(M-n)!}\left(\dfrac{\lambda}{\mu}\right)^n, & \text{Region (iii)}. \end{cases} \tag{1.3}$$

Here we have used Stirling's formula to approximate factorial functions with large arguments. The approximations of the factorial functions change from region to region, thus leading to the different expansions.

The asymptotic expansion of p_0 for $M \gg 1$ depends on the ratio λ/μ. If we define the instantaneous traffic intensity by $r(\xi) = \lambda(M-n)/M\mu = (\lambda/\mu)(1-\xi)$, then the asymptotic expansion depends on whether $r(0) = \lambda/\mu < 1$, $r(0) = 1$, or $r(0) > 1$. The scaled number, $\xi(t) = (1/M)N(Mt)$, of requests in the service facility may be *qualitatively* described by the deterministic ("fluid") equation

$$\frac{d\xi(t)}{dt} = \lambda(1 - \xi(t)) - \mu, \qquad 0 \leqq \xi \leqq 1. \tag{1.4}$$

We call a point ξ^* which satisfies $\lambda(1-\xi^*) = \mu$, a rest point of the fluid approximation. If $\lambda/\mu < 1$ then $r(\xi) < 1$ for $0 \leqq \xi \leqq 1$ and there is no rest point of (1.4) in $0 \leqq \xi \leqq 1$. When $r(0) = \lambda/\mu = 1$, $\xi = 0$ is a rest point of (1.4). Finally, if $\lambda/\mu > 1$ then $r(1-\mu/\lambda) = 1$ and $\xi^* = 1 - \mu/\lambda$ is a rest point of (1.4). The asymptotic expansion of p_0 can be obtained using an integral representation of (1.2)

$$p_0 = \frac{1}{M\mu/\lambda} \sum_{k=0}^{M} \binom{M}{k} \int_0^\infty x^k e^{-M\mu x/\lambda}\, dx = \frac{1}{M\mu/\lambda} \int_0^\infty e^{-M[(\mu x/\lambda) - \log(1+x)]}\, dx,$$

which for $M \gg 1$ has the asymptotic behavior

$$p_0 \sim \begin{cases} 1 - \lambda/\mu, & \lambda/\mu < 1,\\ \sqrt{2/\pi M}, & \lambda/\mu = 1,\\ e^{M(1-\mu/\lambda + \log \mu/\lambda)}/\sqrt{2\pi M}, & \lambda/\mu > 1. \end{cases} \tag{1.5}$$

From (1.5), we see that p_0 is exponentially small for the case when $\lambda/\mu > 1$, which is reasonable since the instantaneous traffic intensity satisfies $r(\xi) > 1$, and hence the number in queue, on the average, grows with ξ in $0 \leqq \xi < 1 - \mu/\lambda$. We note that, even

for the simple finite-source $M/M/1$ queue, the asymptotic approximations are different in each of the three regions and for each of the different cases of λ/μ.

When the service time requests are not exponential, the number of requests $N(t)$ is no longer a Markov process. Thus, we introduce the supplementary variable $Y(t)$, which denotes the amount of service already received by the request currently being served (i.e., the age of service). The joint process $(N(t), Y(t))$ is then Markov. In § 2, we formulate the stationary forward Kolmogorov equations for the joint process $(N(t), Y(t))$, for a finite-source $M/G/1$ queueing system. An exact solution to this model is given in [4]. Unfortunately, as is often the case, the exact solution is sufficiently complicated that not much useful information is provided about the system. To obtain simpler formulas, we compute asymptotic expansions of this solution for $M \gg 1$. To do so, we introduce a scaled process. We scale the number of requests in the facility by the reciprocal of the number of sources M, which we assume is large, and also scale time by $t \to Mt$. The scaled process $(1/M)N(Mt)$ moves in small steps but on a fast time scale. The state space of the scaled process consists of the lattice points $(0, 1/M, 2/M, \cdots)$, but since $M \gg 1$, the scaled number of requests is treated as a continuous variable.

The main results of the paper are contained in §§ 3 and 4, where we construct asymptotic approximations to the stationary density $p(n, y)$ of the joint process $(N(t), Y(t))$ in each of Regions (i), (ii), (iii) and for each of the three cases of λ/μ, as indicated in the case of exponential service described above. Using these formulas, we obtain the marginal probabilities, $p(n) = \int_0^\infty p(n, y)\, dy$ that there are n requests in the service facility. We show that our results reduce to the asymptotic expansions of the exact solution (1.1) when specialized to the case of exponential service requests described above.

We construct asymptotic approximations to $p(n, y)$ using both the WKB method and the method of matched asymptotic expansions [8], [9]. Our asymptotic expansions are in the same spirit as the asymptotic expansions recently developed in [6], [7], [23]–[25] but are derived using a new and different approach. In [6], [7], [23]–[25], an integral representation of an exact solution to some queueing model (e.g., product form solution) is developed and then asymptotic expansions of the integral are constructed. In contrast, we apply perturbation methods directly to the equations for the quantity of interest, in this case the queue length distribution. Our approach has the advantage of being applicable even if no exact solution is available. Also, we need not assume any special form of the service density $b(z)$. The cost of this added generality is that it is more difficult to obtain error bounds using our approach. The justification for our results is that they reduce to the expansions of exact solutions in the special cases when such solutions are known. In addition the WKB method is widely used to solve problems in physics, e.g., in quantum mechanics [11] and wave propagation [12]. It was first employed in probability and statistics in an ad hoc manner by Daniels [13], [14]. Ludwig [15] and Matkowsky and Schuss [16] used the WKB (ray) method to obtain approximate stationary distributions for diffusion processes. Extensions of the method to general Markov jump processes appear in [17] and with applications to queueing models in [18], [19].

In § 3, we compute the approximation in Region (ii) by using a WKB technique [10]. In § 4, we construct the approximation in Region (i) and connect this to the WKB approximation of Region (ii) by the principle of asymptotic matching [8], [10]. In addition, we compute the first two moments of the queue length. We specialize our results to an $M/E_2/1$ system in § 5. The details of the computations in Region (iii) are relegated to two appendices.

2. Formulation. We now formulate our model for the queueing system described in § 1. We let $p(n, y, t) = \partial/\partial y \Pr[N(t) = n, Y(t) \leqq y]$ for $n = 0, 1, \cdots, M$ and $0 \leqq y < \infty$. The joint stationary density function is then $p(n, y) = \lim_{t\to\infty} p(n, y, t)$. We note that the solution for the $M/M/1$ system described in § 1 is the marginal density, i.e., $p(n) = \int_0^\infty p(n, y)\, dy$, which is a quantity of interest. The stationary density $p(n, y)$ satisfies the following set of equations (stationary forward Kolmogorov equation(s)):

$$p_y(n, y) = -[\lambda(M-n)+\mu(y)]p(n, y)+\lambda(M-n+1)p(n-1, y), \qquad 2 \leqq n \leqq M, \tag{2.1}$$

$$p(n, 0) = \int_0^\infty p(n+1, y)\mu(y)\, dy, \quad 2 \leqq n \leqq M-1, \quad p(M, 0) = 0, \tag{2.2}$$

$$p_y(1, y) = -[\lambda(M-1)+\mu(y)]p(1, y), \tag{2.3}$$

$$\lambda M p(0) = \int_0^\infty p(1, y)\mu(y)\, dy, \tag{2.4}$$

$$p(1, 0) = \lambda M p(0) + \int_0^\infty p(2, y)\mu(y)\, dy. \tag{2.5}$$

Here, we have used the fact that $p(0, y) = p(0)\delta(y)$, where $\delta(y)$ is the Dirac delta function, since the age $Y(t)$ must be zero if the facility is empty and $\mu(y) \equiv b(y)/\int_y^\infty b(t)\, dt$ is the instantaneous rate at which a request being executed is completed, given that it has age $Y(t) = y$. In addition, the state-dependent arrival rate is $\lambda(M-n)$ given that $N(t) = n$, since there are $M-n$ unblocked sources each with an exponential think time. The only restriction on the service density $b(x)$ is that $M(\theta) \equiv \int_0^\infty e^{x\theta} b(x)\, dx$ exists in some neighborhood of $\theta = 0$. In addition, we define the moments m_i of the service time as

$$m_i = \int_0^\infty x^i b(x)\, dx. \tag{2.6}$$

We define the scaled number and age as $\xi = n/M$ and $\eta = My$, respectively. The range of ξ is clearly $0 \leqq \xi \leqq 1$. In terms of the scaled variables the arrival rate of requests conditioned on $N(t) = n$ becomes $M\lambda(1-\xi)$. The scaled service density is defined as $b = Mb(Mx)$, so that the mean time to process an individual request is short ($O(1/M)$). Using this scaling we see that the instantaneous service rate is now $M\mu(\eta)$. In terms of ξ and η the probabilities of changes in $N(t)$ in a time interval of length Δt are given by

$$\Pr[N(t+\Delta t) - N(t) = 1 \mid Y(t) = \eta/M, N(t) = M\xi] = M\lambda(1-\xi)\Delta t + o(\Delta t)$$

and

$$\Pr[N(t+\Delta t) - N(t) = -1 \mid Y(t) = \eta/M, N(t) = M\xi] = M\mu(\eta)\Delta t + o(\Delta t).$$

The arrival and departure rates are proportional to M and are hence large. Thus, during a typical period of observation, the service facility receives and processes many requests. The scaling of the age $\eta = My$ reflects the fast time scale on which the process evolves.

The stationary forward equations (2.1)-(2.5) for the scaled $p(\xi, \eta)$ become

$$(2.7)\quad p_\eta(\xi,\eta) = -[\lambda(1-\xi)+\mu(\eta)]p(\xi,\eta)+\lambda\left(1-\xi+\frac{1}{M}\right)p\left(\xi-\frac{1}{M},\eta\right),\quad \frac{2}{M}\leqq\xi\leqq 1,$$

$$(2.8)\quad p(\xi,0) = \int_0^\infty p\left(\xi+\frac{1}{M},z\right)\mu(z)\,dz,\quad \frac{2}{M}\leqq\xi\leqq 1-\frac{1}{M},\quad p(1,0)=0,$$

$$(2.9)\quad p_\eta\left(\frac{1}{M},\eta\right) = -\left[\lambda\left(1-\frac{1}{M}\right)+\mu(\eta)\right]p\left(\frac{1}{M},\eta\right),$$

$$(2.10)\quad \lambda p(0) = \int_0^\infty p\left(\frac{1}{M},z\right)\mu(z)\,dz,$$

$$(2.11)\quad p\left(\frac{1}{M},0\right) = \lambda p(0)+\int_0^\infty p\left(\frac{2}{M},z\right)\mu(z)\,dz.$$

The function $p(\xi, \eta)$ is uniquely determined by (2.7)-(2.11) with the normalization requirement

$$(2.12)\qquad p(0)+\sum_{k=1}^{M}\int_0^\infty p\left(\frac{k}{M},\eta\right)d\eta = 1.$$

Jaiswal [4] obtained an exact expression for the marginal probabilities $p(n)$ using renewal-theoretic arguments. Unfortunately, the expression is complicated and difficult to interpret, even in limiting cases such as $M \gg 1$. We construct asymptotic expansions approximating $p(n)$ for $M \gg 1$ directly from (2.7)-(2.12). As indicated by the asymptotic expansion of the stationary probabilities $p(n)$ for exponential service times, the forms of the asymptotic expansions depend on the ratio λ/μ and on whether n is in Region (i), (ii) or (iii). Asymptotic expansions for $p(\xi, \eta)$ are constructed in the sections that follow.

3. Region (ii): The WKB approximation. We now construct a WKB type approximation [10] to the solution of the scaled forward equation (2.7)-(2.11) with the normalization condition (2.12). We seek an asymptotic representation of $p(\xi, \eta)$ for $M \gg 1$ in the form

$$(3.1)\qquad p_{\mathrm{WKB}}(\xi,\eta) \sim e^{-M[\psi_0+\psi_1/M+\psi_2/M^2+\cdots]}$$

where

$$\psi_j = \psi_j(\xi,\eta),\qquad j=0,1,2,\cdots.$$

We substitute (3.1) into (2.7) and equate the coefficient of each power of $1/M$ separately to zero. Thus (2.7) is transformed into a sequence of first order partial differential equations, the first of which is

$$(3.2)\qquad -\psi_{0,\eta} \equiv -\frac{\partial\psi_0}{\partial\eta} = 0.$$

We conclude that $\psi_0 = \psi_0(\xi)$. For the next term we obtain

$$(3.3)\qquad -\psi_{1,\eta}+\mu(\eta) = \lambda(1-\xi)[e^{\psi_0'(\xi)}-1].$$

The left side of (3.3) is thus independent of η and we may define

$$(3.4)\qquad K(\xi) = -\psi_{1,\eta}+\mu(\eta),$$

so that

$$(3.5)\qquad \psi_0'(\xi) = \log\left[1+\frac{K(\xi)}{\lambda(1-\xi)}\right]$$

and hence

$$\psi_0(\xi) = \int_{\xi^*}^{\xi} \log\left[1 + \frac{K(s)}{\lambda(1-s)}\right] ds. \tag{3.6}$$

Here ξ^* is chosen to be 0 if $\lambda/\mu \leqq 1$, and $1-(\mu/\lambda)$ if $\lambda/\mu > 1$, since $r(1-\mu/\lambda) = 1$. The point ξ^* is chosen so that $\psi_0(\xi)$ vanishes where the probability density peaks and is positive elsewhere.

The solution of (3.4) is

$$\psi_1(\xi, \eta) = \int_0^{\eta} \mu(t)\, dt - K(\xi)\eta + L(\xi), \tag{3.7}$$

where $K(\xi)$ and $L(\xi)$ are as yet undetermined. By similarly expanding the jump condition (2.8), using (3.6) and (3.7), we find that $K(\xi)$ satisfies

$$1 + \frac{K(\xi)}{\lambda(1-\xi)} = \int_0^{\infty} e^{zK(\xi)} b(z)\, dz. \tag{3.8}$$

A convexity argument shows that (3.8) has a unique solution $K(\xi)$ that is not identically zero. For the next term in the expansion of (2.7), we find

$$-\psi_{2,\eta} = \lambda(1-\xi)\, e^{\psi_0'(\xi)}[-\eta K'(\xi) + L'(\xi)] - \tfrac{1}{2}\lambda(1-\xi)\, e^{\psi_0'(\xi)}\psi_0''(\xi) + \lambda\, e^{\psi_0'(\xi)}. \tag{3.9}$$

Substitution of (3.9) into the expansion of the jump condition (2.8) implies that $L(\xi)$ satisfies

$$[1-\lambda(1-\xi)I_1(\xi)]L'(\xi) = \tfrac{1}{2}\psi_0''(\xi) + [\lambda - \tfrac{1}{2}\lambda(1-\xi)\psi_0'']I_1(\xi) - \tfrac{1}{2}K'(\xi)\lambda(1-\xi)I_2(\xi), \tag{3.10}$$

where

$$I_j(\xi) = \int_0^{\infty} z^j e^{zK(\xi)} b(z)\, dz, \qquad j = 1, 2 \cdots. \tag{3.11}$$

The calculation of the correction terms ψ_j, $j > 1$, is routine but tedious. The corrections are solutions of first order linear differential equations such as (3.10). The corrections can be expressed in terms of $K(\xi)$ and its derivatives.

We note that $\psi_0(\xi)$ and $\psi_1(\xi, \eta)$ are now determined in terms of $K(\xi)$ and $L(\xi)$, which satisfy (3.8) and (3.10), respectively. Thus, we have the leading term in the asymptotic expansion of $p(\xi, \eta)$

$$p_{\mathrm{WKB}}(\xi, \eta) \sim \hat{c} \exp\left(-M \int_{\xi^*}^{\xi} \log\left[1 + \frac{K(t)}{\lambda(1-t)}\right] dt\right) \cdot \exp\left(-\int_0^{\eta} \mu(t)\, dt\right) e^{K(\xi)\eta} \sqrt{\frac{(1-\xi)K'(\xi)}{\lambda(1-\xi) + K(\xi)}} \tag{3.12}$$

with the constant $\hat{c}$ as yet undetermined. This constant is found in § 4 along with the asymptotic expansions in Regions (i) and (iii).

The formula (3.12) is valid for any $b(z)$ that decays at least exponentially fast as $z \to \infty$.

We give $K(\xi)$ for the following special cases:

(i) exponential service: $b(x) = \mu\, e^{-\mu x}$,

$$K(\xi) = \mu - \lambda(1-\xi),$$

(ii) E_2-service (2-stage Erlang) (cf. § 5): $b(x) = (2\mu)^2 x\, e^{-2\mu x}$,

$$K(\xi) = 2\mu - \frac{\lambda(1-\xi)}{2} - \frac{1}{2}\lambda(1-\xi)\sqrt{1 + \frac{8\mu}{\lambda(1-\xi)}},$$

(iii) H_2 service (hyperexponential): $b(x)=\alpha_1 a_1 e^{-a_1 x}+\alpha_2 a_2 e^{-a_2 x}$, $\alpha_1+\alpha_2=1$,

$$K(\xi)=\frac{1}{2}[a_1+a_2-\lambda(1-\xi)-\sqrt{[a_1+a_2-\lambda(1-\xi)]^2-4a_1a_2(1-r(\xi))}],$$

$$\frac{1}{\mu}=\frac{\alpha_1}{a_1}+\frac{\alpha_2}{a_2}.$$

Furthermore, the transcendental equation (3.8) also applies to empirical service time distributions. If we observe N customers with respective service times $x_1, \cdots, x_N$ then

$$b(z)=\frac{1}{N}\sum_{i=1}^{N}\delta(z-x_i),$$

in which case (3.8) becomes

$$1+\frac{K(\xi)}{\lambda(1-\xi)}=\frac{1}{N}\sum_{i=1}^{N}e^{x_i K(\xi)} \qquad (K\not\equiv 0),$$

which can be readily solved numerically.

The WKB approximation (3.12) breaks down for small values of $\xi(\xi=O(1/M))$, as well as when ξ is close to unity $(1-\xi=O(1/M))$. In constructing the WKB approximation, we have used only (2.7) and (2.8) of the set (2.7)-(2.11), thus neglecting the boundary effects near $\xi=0$. Therefore, the WKB approximation is not valid when ξ is close to zero. Furthermore, from (3.8) and (3.12) we see that the function $K(\xi)$ develops singularities which cause the marginal density obtained from expression (3.12) to become infinite as $\xi\to 1$. Since the exact solution for $p(\xi)$ clearly cannot become infinite for $\xi=1$, the WKB approximation also breaks down when ξ is close to unity.

4. Asymptotic expansion in Region (i). We now construct the asymptotic expansion that is valid in Region (i) where $\xi=O(1/M)$ (i.e., $n=O(1)$). In analogy with the asymptotic expansion of the exact solution (cf. (1.3)), we take into account the boundary effects described in the last three equations (2.9)-(2.11). In Region (i), we introduce the "stretched" variable

$$k=M\xi \tag{4.1}$$

and the local function $Q_k(\eta; 1/M)=p(\xi, \eta; 1/M)$.

We expand $Q_k(\eta; 1/M)$ as

$$Q_k(\eta; 1/M)\sim Q_k^{(0)}(\eta)+\frac{1}{M}Q_k^{(1)}(\eta)+\cdots. \tag{4.2}$$

In terms of the stretched variable (4.1), the arrival rate is $\lambda(1-\xi)=\lambda-\lambda k/M$. Using this arrival rate and (4.2) in (2.7)-(2.13) we find that $Q_k^{(0)}(\eta)$ satisfies

$$\begin{aligned}
&\frac{d}{d\eta}Q_k^{(0)}(\eta)=\lambda Q_{k-1}^{(0)}(\eta)-[\lambda+\mu(\eta)]Q_k^{(0)}(\eta), \qquad k\geqq 2,\\
&Q_k^{(0)}(0)=\int_0^\infty Q_{k+1}^{(0)}(z)\mu(z)\,dz, \qquad k\geqq 2,\\
&\frac{d}{d\eta}Q_1^{(0)}(\eta)=-[\lambda+\mu(\eta)]Q_1^{(0)}(\eta),\\
&\lambda Q_0^{(0)}=\int_0^\infty Q_1^{(0)}(z)\mu(z)\,dz,\\
&Q_1^{(0)}(0)=\lambda Q_0^{(0)}+\int_0^\infty Q_2^{(0)}(z)\mu(z)\,dz.
\end{aligned} \tag{4.3}$$

Equations (4.3) are merely the equations of an infinite population $M/G/1$ queueing system with arrival rate λ and service density $b(z)$. The finite-source effect is asymptotically negligible when the process is observed only near $\xi = 0$. The solution of (4.3) can be constructed in a straightforward manner by introducing the generating function [1], [20]

$$G(\eta, s) = \sum_{n=1}^{\infty} s^n Q_n^{(0)}(\eta).$$

At this point it is necessary to distinguish the three cases. First, we assume that $\lambda/\mu < 1$, i.e., $r(\xi) = \lambda(1-\xi)/\mu < 1$ for all $\xi \geqq 0$. In this case, most of the dynamics of the queueing system occur in Region (i), where $\xi = O(1/M)$. Thus, it is appropriate to impose the normalization condition (2.12), at least to leading order in $1/M$, on the local solution (4.2). With the above conditions, (4.3) has the solution

$$Q_k^{(0)}(\eta) = \frac{1}{2\pi i}\left[\int_C \frac{\lambda Q_0^{(0)}(s-1)}{s^k[s-\hat{b}(\lambda-\lambda s)]} e^{\lambda(s-1)\eta}\, ds\right] \exp\left(-\int_0^{\eta} \mu(t)\, dt\right) \tag{4.4}$$

for $k \geqq 1$. Here $\hat{b}(s) = \int_0^{\infty} e^{-sz} b(z)\, dz$ and C is a closed contour in the complex s plane containing the origin but excluding any other singularity of the integrand. The constant $Q_0^{(0)}$, the stationary probability of finding the system empty, is determined to be

$$Q_0^{(0)} = 1 - \frac{\lambda}{\mu}. \tag{4.5}$$

We note that (4.5) represents the leading term in an asymptotic expansion of the exact solution. Higher order correction terms can be obtained in a straightforward manner. Each of the correction terms $Q_k^{(j)}$, $j \geqq 1$, satisfy an inhomogeneous version of (4.3) which can be solved using generating functions.

We now match the expansions in Regions (i) and (ii). This enables us to determine the constant $\hat{c}$ in the WKB expansion (3.12). We demand that the two expansions agree in an intermediate region $1/M \ll \xi \ll 1$. The above is accomplished by equating the large k behavior of (4.4) with the small ξ behavior of (3.12). The local solution (4.4), as $k \to \infty$, takes the form

$$Q_k^{(0)}(\eta) \sim \lambda\left(1-\frac{\lambda}{\mu}\right)\frac{(s^*-1)\, e^{-k\log s^*}}{\lambda I_1(0) - 1} e^{\lambda(s^*-1)\eta} \exp\left(-\int_0^{\eta} \mu(t)\, dt\right), \tag{4.6}$$

where $I_1(0)$ is defined by (3.11) and s^* is the unique real solution of

$$s = \int_0^{\infty} e^{\lambda(s-1)t} b(t)\, dt, \tag{4.7}$$

which satisfies $s^* \neq 1$. We note that $r(0) < 1$ implies that $s^* > 1$ so that the local solution, $Q_k^{(0)}(\eta)$, decays as $k \to \infty$. The WKB approximation, (3.12), expanded for $\xi \ll 1$ is

$$p(\xi, \eta) \sim \hat{c} \exp\left(-M\xi \log\left[1+\frac{K(0)}{\lambda}\right]\right) e^{K(0)\eta} \exp\left(-\int_0^{\eta} \mu(t)\, dt\right)\sqrt{\frac{K'(0)}{\lambda + K(0)}}. \tag{4.8}$$

Here we have chosen $\xi^* = 0$. Comparing (4.6) with (4.8) and noting that $K(0) = \lambda(s^*-1)$, we find that

$$\hat{c} = \frac{\lambda(1-\lambda/\mu)(s^*-1)}{\lambda I_1(0)-1}\sqrt{\frac{\lambda+K(0)}{K'(0)}} = \left(1-\frac{\lambda}{\mu}\right)\sqrt{K'(0)[\lambda+K(0)]}. \tag{4.9}$$

Thus, the expansions in Regions (i) and (ii) are now completely determined for this case. We summarize the results for $\lambda/\mu < 1$ as

$$p(\xi,\eta)\sim\left(1-\frac{\lambda}{\mu}\right)\exp\left(-\int_0^\eta \mu(t)\,dt\right)\begin{cases}\dfrac{\lambda}{2\pi i}\displaystyle\int_C \frac{s-1}{s^{M\xi}[s-\hat{b}(\lambda-\lambda s)]}\,e^{\lambda(s-1)\eta}\,ds,\\ \qquad \xi = O\left(\dfrac{1}{M}\right),\\ \left[\dfrac{(1-\xi)K'(0)K'(\xi)[\lambda+K(0)]}{\lambda(1-\xi)+K(\xi)}\right]^{1/2} e^{\eta K(\xi)}\\ \cdot\exp\left(-M\displaystyle\int_0^\xi \log\left[1+\frac{K(t)}{\lambda(1-t)}\right]dt\right),\\ \xi = O(1),\quad 1-\xi = O(1).\end{cases} \tag{4.10}$$

We also have shown that the probability of finding an empty system, $p(0)$, is approximated by

$$p(0)\sim Q_0^{(0)} = 1-\frac{\lambda}{\mu}, \tag{4.11}$$

which coincides with the exact solution for an infinite-source $M/G/1$ system. Using (4.10), we compute the marginal density in Region (ii), i.e., $\xi = O(1)$, $1-\xi = O(1)$, as

$$\begin{aligned} p(\xi) = \int_0^\infty p(\xi,\eta)\,d\eta &\sim \left(\frac{1}{\lambda}-\frac{1}{\mu}\right)\left|\frac{K'(0)K'(\xi)[\lambda+K(0)]}{(1-\xi)[\lambda(1-\xi)+K(\xi)]}\right|^{1/2}\\ &\quad\cdot\exp\left(-M\int_0^\xi \log\left(1+\frac{K(t)}{\lambda(1-t)}\right)dt\right).\end{aligned} \tag{4.12}$$

We note that $p(0)$ agrees with the $M/M/1$ result (1.5) obtained when $\lambda/\mu < 1$. If we specialize (4.12) to the case $b(z) = \mu e^{-\mu z}$, we find that $K(\xi) = \mu - \lambda(1-\xi)$ and

$$\begin{aligned} p(n) &\sim \frac{(1-\lambda/\mu)}{\sqrt{1-n/M}}\exp\left(-M\int_0^{n/M}\log\left[\frac{\mu}{\lambda(1-t)}\right]dt\right)\\ &= \left(1-\frac{\lambda}{\mu}\right)e^{-n}\left(\frac{\lambda}{\mu}\right)^n\left(1-\frac{n}{M}\right)^{n-M-1/2},\end{aligned} \tag{4.13}$$

which again agrees with the asymptotic expansion of the exact solution in Region (ii) given by (1.3).

We now consider the case $\lambda/\mu = 1$, where $r(0) = 1$ and $r(\xi) < 1$, $0 < \xi \leqq 1$. The dynamics now take place in an $O(1/\sqrt{M})$ neighborhood of $\xi = 0$ rather than the thinner, $O(1/M)$, neighborhood as in the case with $\lambda/\mu < 1$. Since for $\xi = O(1/\sqrt{M})$, the WKB solution (3.12) remains valid, we impose the normalization condition (2.14) directly on (3.12) to determine the unknown constant $\hat{c}$. We again choose $\xi^* = 0$. Using Laplace's method for the asymptotic evaluation of integrals [10], we find that

$$\begin{aligned} p(\xi,\eta) &\sim \sqrt{\frac{2\lambda\mu}{\pi M}}\sqrt{\left|\frac{(1-\xi)K'(\xi)}{\lambda(1-\xi)+K(\xi)}\right|}\exp\left(-\int_0^\eta \mu(t)\,dt\right)\\ &\quad\cdot e^{\eta K(\xi)}\exp\left(-M\int_0^\xi \log\left[1+\frac{K(t)}{\lambda(t)}\right]dt\right),\end{aligned} \tag{4.14}$$

where $\lambda = \mu$. The condition $r(0) = 1$ implies $K(0) = 0$ and hence one can readily show from the Euler–Maclaurin summation formula that it is irrelevant whether $p(\eta, \xi)$ is

normalized by using a sum over $n = M\xi$ or by integrating over ξ, up to a multiplicative factor of $1/M$.

The WKB approximation again fails to be valid for $\xi = O(1/M)$ so we again consider a local expansion in Region (i) by introducing the stretched variable (4.1) and the local function (4.2). Solving for the leading term in the expansion (4.2), we again obtain (4.4) but with Q_0, the probability that the system is empty, now an unknown constant. From (4.4), we obtain the large k expansion, as

$$Q_k^{(0)}(\eta) \sim \frac{2Q_0^{(0)}}{\lambda m_2} \exp\left(-\int_0^\eta \mu(t)\,dt\right), \qquad k \gg 1, \tag{4.15}$$

where m_2 is the second moment of the service time. Expanding (4.14) for $\xi \ll 1$ and matching the result to the expansion of $Q_k^{(0)}(\eta)$ for $k \gg 1$ given by (4.15), we find that for the two expansions to agree, we must have

$$Q_0^{(0)} = \lambda\sqrt{\frac{m_2}{M\pi}}. \tag{4.16}$$

Thus, we summarize the results for the case $\lambda/\mu = 1$ as

$$p(\xi,\eta) \sim \exp\left(-\int_0^\eta \mu(t)\,dt\right) \begin{cases} \dfrac{\lambda}{2\pi i}\displaystyle\int_C \frac{(s-1)Q_0}{s^{M\xi}[s-\hat{b}(\lambda-\lambda s)]} e^{\lambda(s-1)\eta}\,ds, \\ \qquad \xi = O\left(\dfrac{1}{M}\right), \\ \sqrt{\dfrac{2}{\pi M}}\,\mu\sqrt{\dfrac{(1-\xi)K'(\xi)}{\lambda(1-\xi)+K(\xi)}}\, e^{\eta K(\xi)} \\ \cdot \exp\left(-M\displaystyle\int_0^\xi \log\left[1+\frac{K(t)}{\lambda(1-t)}\right]dt\right), \\ \qquad \xi = O(1), \quad 1-\xi = O(1), \end{cases} \tag{4.17}$$

$$p(0) \sim Q_0^{(0)} = \lambda\sqrt{m_2/\pi M}. \tag{4.18}$$

The marginal density is now given by

$$p(\xi) \sim \sqrt{\frac{2}{\pi M}}\sqrt{\frac{K'(\xi)}{(1-\xi)[\lambda(1-\xi)+K(\xi)]}} \exp\left(-M\int_0^\xi \log\left[1+\frac{K(t)}{\lambda(1-t)}\right]dt\right). \tag{4.19}$$

Again specializing to the case of exponential service, we see that $p(0)$ given by (4.18) reduces to the result in (1.5) and the marginal density agrees with that given in (1.3).

Finally, we consider the case when $\lambda/\mu > 1$ in which there is an equilibrium point at $\xi = \xi^* = 1 - \mu/\lambda$. Since ξ^* is an attractor, the queueing system spends most of its time in an $O(1/\sqrt{M})$ region centered at ξ^*. As in the previous case, we normalize the WKB solution to obtain

$$\begin{aligned} p(\xi,\eta) \sim{}& \sqrt{\frac{\lambda\mu}{2\pi M}}\sqrt{\frac{(1-\xi)K'(\xi)}{\lambda(1-\xi)+K(\xi)}} \exp\left(-\int_0^\eta \mu(t)\,dt\right) \\ & \cdot e^{\eta K(\xi)} \exp\left(-M\int_{1-\mu/\lambda}^\xi \log\left[1+\frac{K(t)}{\lambda(1-t)}\right]dt\right). \end{aligned} \tag{4.20}$$

The local analysis in Region (i) follows the two previous cases and we again obtain the local solution (4.4). The asymptotic expansion of (4.4) for $k \gg 1$ and $\lambda/\mu > 1$ is now given by

$$Q_k^{(0)}(\eta) \sim \frac{\lambda Q_0(1-s^*)\, e^{-k\log s^*}}{1-\lambda I_1(0)}\, e^{\lambda(s^*-1)\eta} \exp\left(-\int_0^\eta \mu(t)\, dt\right), \tag{4.21}$$

where s^* is again defined by (4.7). Since $\lambda/\mu > 1$, we now have $0 < s^* < 1$ so that the local solution grows exponentially as k increases. Matching this expression to the WKB solution as $\xi \to 0$ yields

$$Q_0^{(0)} = \sqrt{\frac{\lambda\mu}{2\pi M}}\sqrt{\frac{1}{K'(0)[\lambda + K(0)]}} \exp\left(M\int_0^{1-\mu/\lambda} \log\left[1+\frac{K(t)}{\lambda(1-t)}\right] dt\right). \tag{4.22}$$

Here we have used $K(0) = \lambda(s^*-1)$. Since $K(\xi) < 0$ for $0 < \xi < 1-\mu/\lambda$, we see that the probability of having an empty system is exponentially small.

Finally, we give the marginal density as

$$\begin{aligned} p(\xi) \sim{}& \sqrt{\frac{\mu}{2\pi\lambda M}}\sqrt{\frac{K'(\xi)}{(1-\xi)[\lambda(1-\xi)+K(\xi)]}} \\ &\cdot \exp\left(-M\int_{1-\mu/\lambda}^{\xi} \log\left[1+\frac{K(t)}{\lambda(1-t)}\right] dt\right). \end{aligned} \tag{4.23}$$

Again, we see that when specialized to the case of exponential service, (4.22) agrees with the corresponding result in (1.5) and the marginal density (4.23) agrees with the result in (1.3).

At this point, we have computed the asymptotic expansions in Regions (i) and (ii) and can now obtain useful information about the system, specifically the first two moments of the process $N(t)$ in the stationary limit. The solution in Region (iii) represents only an exponentially small tail of the stationary solution. This solution is constructed in Appendix 1, where its significance is described.

Our analysis shows that Q_0, the probability of finding an empty system, differs radically for the three cases. If $\lambda/\mu < 1$, we showed that $Q_0 = 1-\lambda/\mu + O(1/M)$, where the $O(1/M)$ term involves derivatives of $r(\xi)$ and $\lambda(1-\xi)$ evaluated at $\xi = 0$. For the case with $\lambda/\mu = 1$, we found that $Q_0 = O(1/\sqrt{M})$. As in the case $\lambda/\mu < 1$, the system drifts toward $\xi = 0$, but the magnitude of the drift increases near $\xi = 0$ so the system resides most of the time in the neighborhood of $\xi = 0$ but not directly at $\xi = 0$. This explains why Q_0 is smaller if $\lambda/\mu = 1$. For the case when $\lambda/\mu > 1$, $Q_0 = O(e^{-M\psi_0(0)}/\sqrt{M})$ with $\psi_0(0) > 0$, indicating that an empty system is indeed a rare event.

We now compute the leading term in the asymptotic expansion of the first two moments of the number in the service facility. When $\lambda/\mu < 1$, the moments are computed using the probabilities (4.4) to yield

$$\begin{aligned} &\lim_{t\to\infty} E[N(t)] \sim \frac{\lambda}{\mu} + \frac{\lambda^2 m_2}{2(1-\lambda/\mu)}, \qquad M \gg 1, \\ &\lim_{t\to\infty} E[N^2(t)] \sim \frac{\lambda}{\mu} + \frac{3\lambda^2 m_2}{2(1-\lambda/\mu)} + \frac{\lambda^3 m_3}{3(1-\lambda/\mu)} + \frac{\lambda^4 m_2^2}{2(1-\lambda/\mu)^2}. \end{aligned} \tag{4.24}$$

Here m_2 and m_3 are the second and third moments of the service time, respectively. When $\lambda/\mu = 1$, the moments are computed using the WKB solution and evaluating

the resulting sums by Laplace's method. We obtain

$$\begin{aligned} &\lim_{t\to\infty} E[N(t)]\sim\sqrt{\lambda\mu m_2 M/\pi}, \qquad M\gg 1,\\ &\lim_{t\to\infty} E[N^2(t)]\sim\tfrac{1}{2}\lambda\mu m_2 M. \end{aligned} \tag{4.25}$$

Finally, if $\lambda/\mu>1$, we obtain

$$\begin{aligned} &\lim_{t\to\infty} E[N(t)]\sim M\left(1-\frac{\mu}{\lambda}\right), \qquad M\gg 1,\\ &\lim_{t\to\infty} E[N^2(t)]\sim M^2\left(1-\frac{\mu}{\lambda}\right)^2. \end{aligned} \tag{4.26}$$

5. E_2 service distribution. We now consider the special case in which the service times have a 2-stage Erlang distribution, E_2, with density function

$$b(x)=(2\mu)^2 x\, e^{-2\mu x}. \tag{5.1}$$

Thus, the closed system is equivalent to a finite population $M/E_2/1$ queueing system. We now specialize the formulas for $p(\xi,\eta)$ derived in §§ 3 and 4 and in addition determine explicitly the unknown constant c in the asymptotic expansion in Region (iii) (cf. Appendix 1).

We first note that the function $K(\xi)$ defined by (3.8) can be found explicitly as

$$K(\xi)=2\mu-\frac{\lambda(1-\xi)}{2}-\frac{1}{2}\lambda(1-\xi)\sqrt{1+\frac{8\mu}{\lambda(1-\xi)}}. \tag{5.2}$$

We find it convenient to define the functions

$$\gamma(\xi)\equiv\sqrt{1+\frac{8\mu}{\lambda(1-\xi)}} \tag{5.3}$$

and

$$\begin{aligned} \Delta(\xi)&\equiv\frac{\lambda}{4\mu}\int^{\xi}\log\left[1+\frac{K(t)}{\lambda(1-t)}\right]dt\\ &=\log\frac{\gamma(\xi)+1}{\gamma(\xi)-1}-\frac{2}{\gamma(\xi)-1}-\frac{4\log[\frac{1}{2}(\gamma(\xi)-1)]}{(\gamma(\xi)+1)(\gamma(\xi)-1)}. \end{aligned} \tag{5.4}$$

In terms of $\gamma(\xi)$, $K(\xi)$ is given by

$$K(\xi)=\frac{2\mu[\gamma(\xi)-3]}{\gamma(\xi)-1}. \tag{5.5}$$

We now summarize the results in Region (ii), the marginal density, and the probability, $p(0)$, of finding an empty service facility for the three cases of λ/μ.

1) $\lambda/\mu<1$. By using (5.1)-(5.5) in (4.10)-(4.12), we find that

$$\begin{aligned} p(\xi,\eta)\sim 2\left(1-\frac{\lambda}{\mu}\right)&\frac{\mu}{\sqrt{\gamma(0)\gamma(\xi)}}\left[\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\right]\exp\left(-\int_0^{\eta}\mu(t)\,dt\right)\\ &\cdot e^{\eta K(\xi)}\exp\left(-(4\mu M/\lambda)[\Delta(\xi)-\Delta(0)]\right), \qquad \xi=O(1), \qquad 1-\xi=O(1), \end{aligned} \tag{5.6}$$

$$p(0)\sim 1-\frac{\lambda}{\mu}, \tag{5.7}$$

and that the marginal distribution is

$$p(\xi)\sim\frac{1}{4}\left(1-\frac{\lambda}{\mu}\right)\frac{[\gamma(\xi)+1]^2}{\sqrt{\gamma(0)\gamma(\xi)}}\exp\left(-(4\mu M/\lambda)[\Delta(\xi)-\Delta(0)]\right). \tag{5.8}$$

We note that the actual state probabilities for $n\geqq 1$ are given by $(1/M)p(\xi)$.

2) $\lambda/\mu=1$. Again by using (5.1)-(5.5) in (4.17)-(4.19), we find that

$$p(\xi,\eta)\sim\sqrt{\frac{2}{\pi\gamma(\xi)M}}\mu\left[\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\right]\exp\left(-\int_0^\eta\mu(t)\,dt\right)\cdot e^{\eta K(\xi)}\exp\left(-4M[\Delta(\xi)+1-\log 2]\right), \tag{5.9}$$

$$p(0)\sim\sqrt{3/2\pi M}, \tag{5.10}$$

and that the marginal distribution is

$$p(\xi)\sim\frac{1}{4}\sqrt{\frac{1}{2\pi\gamma(\xi)M}}(\gamma(\xi)+1)^2\exp\left(-4M(\Delta(\xi)+1-\log 2)\right). \tag{5.11}$$

3) $\lambda/\mu>1$. Using $\Delta(1-(\mu/\lambda))=\log 2-1$, we obtain

$$p(\xi,\eta)\sim\sqrt{\frac{\lambda\mu}{2\pi\gamma(\xi)M}}\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\exp\left(-\int_0^\eta\mu(t)\,dt\right)\cdot e^{\eta K(\xi)}\exp\left(-(4\mu M/\lambda)[\Delta(\xi)+1-\log 2]\right), \tag{5.12}$$

$$p(0)\sim\frac{1}{2}\sqrt{\frac{\lambda\gamma(0)}{2\pi M\mu}}\exp\left(-(4\mu M/\lambda)[\Delta(0)+1-\log 2]\right), \tag{5.13}$$

and

$$p(\xi)\sim\frac{1}{8}\sqrt{\frac{\lambda}{2\pi\mu\gamma(\xi)M}}(\gamma(\xi)+1)^2\exp\left((4\mu M/\lambda)[\Delta(\xi)+1-\log 2]\right).$$

We note that the solution in Region (i) is given by (4.4) with $Q_0=p(0)$ defined by either (5.7), (5.10), or (5.13) for the three different cases of λ/μ.

We now complete the details of the matching to the local solution in Region (iii). The local solution in Region (iii) is constructed in Appendix 1 and is given by

$$S_m(\eta)=c\exp\left(-\int_0^\eta\mu(t)\,dt\right)\frac{\delta}{m!}\left(\frac{\delta}{2}\right)^m\left(\frac{2}{\delta}\right)^\delta\Gamma(\delta)\,e^{-\lambda\eta(m-\delta)/2M}\cdot\left[(1-e^{-\lambda\eta/m})J_{m+\delta}(\delta e^{-\lambda\eta/2M})+\frac{2m}{\delta}e^{-\lambda\eta/2M}J_{m+\delta-1}(\delta\,e^{-\lambda\eta/2M})\right]. \tag{5.14}$$

Here $m=M(1-\xi)$, $\delta=4\mu M/\lambda$, and $J_\nu(x)$ is the Bessel function of order ν. To determine c, we set $\eta=0$ in (5.14) and expand for $m\gg 1$, $\delta\gg 1$, and $m/\delta=O(1)$. The details of the expansion are in Appendix 2 and the final result is given by

$$S_m(0)\sim c\sqrt{\frac{2}{\pi\delta}}\left|\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\right|\frac{1}{\sqrt{\gamma(\xi)}}\cdot\exp\left(-\delta\left\{\log\left(\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\right)-\frac{2}{\gamma(\xi)-1}-\frac{4\log[(\gamma(\xi)-1)/2]}{[\gamma(\xi)+1][\gamma(\xi)-1]}+1-\log 2\right\}\right). \tag{5.15}$$

We next match (5.15) to the WKB approximation $p_{\mathrm{WKB}}(\xi,0)$ (i.e., $p(\xi,\eta)$ for ξ in Region (ii)). As we now show, (5.14) in fact contains the WKB expansion and is

uniformly valid for all $\xi = O(1)$. To see this, we rewrite $p_{\mathrm{WKB}}(\xi, 0)$ using (5.6), (5.9) and (5.12) as

$$p_{\mathrm{WKB}}(\xi,0) \sim A(M)\left|\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\right|\frac{1}{\sqrt{\gamma(\xi)}}\, e^{-\delta[\Delta(\xi)+1-\log 2]}, \tag{5.16}$$

where

$$A(M) \sim \frac{1}{M}\begin{cases} 2M\left(1-\dfrac{\lambda}{\mu}\right)\dfrac{\mu}{\sqrt{\gamma(0)}}\, e^{\delta[\Delta(0)+1-\log 2]}, & \dfrac{\lambda}{\mu}<1, \\ \sqrt{\dfrac{2M}{\pi}}\,\mu, & \dfrac{\lambda}{\mu}=1, \\ \sqrt{\dfrac{\lambda\mu M}{2\pi}}, & \dfrac{\lambda}{\mu}>1 \end{cases} \tag{5.17}$$

and $\Delta(\xi)$ is defined in (5.4). Comparing the expansions, we see that they match if

$$c = \sqrt{\frac{\pi\delta}{2}}A(M). \tag{5.18}$$

By the definitions of $A(M)$ and δ, (5.18) becomes

$$c = \begin{cases} \sqrt{\dfrac{\pi M}{\lambda\gamma(0)}}(2\mu)^{3/2}\left(1-\dfrac{\lambda}{\mu}\right) e^{\delta[\Delta(0)+1-\log 2]}, & \dfrac{\lambda}{\mu}<1, \\ 2\mu, & \dfrac{\lambda}{\mu}=1, \\ \mu, & \dfrac{\lambda}{\mu}>1. \end{cases} \tag{5.19}$$

This completes the matching and determines the asymptotic expansions of the solution in all three regions.

Finally, we compute the probability $p(M)$ that all sources are blocked:

$$p(M) \sim \int_0^\infty S_0(\eta)\, d\eta = \frac{c}{\mu} + \frac{cM}{\lambda}\sum_{l=1}^{\infty}\frac{(-1)^l}{l}\prod_{k=1}^{l-1}\frac{\beta_k}{1-\beta_k}. \tag{5.20}$$

For this case β_k is given by (A1.23), which when substituted into (5.20) yields

$$p(M) \sim \frac{c}{\mu}\Gamma(\delta)\left(\frac{2}{\delta}\right)^{\delta}\frac{\delta}{2}J_{\delta-1}(\delta). \tag{5.21}$$

Using the values of c given by (5.19) and expanding $\Gamma(\delta)$ and $J_{\delta-1}(\delta)$ for $\delta = 4\mu M/\lambda \gg 1$ we find

$$p(M) \sim \begin{cases} \dfrac{2^{2/3}\Gamma(1/3)}{3^{1/6}}\dfrac{(1-(\lambda/\mu))}{\sqrt{\gamma(0)}}\left(\dfrac{M\mu}{\lambda}\right)^{2/3} e^{-4\mu M|\Delta(0)|/\lambda}, & \dfrac{\lambda}{\mu}<1, \\ \left(\dfrac{2M}{3}\right)^{1/6}\dfrac{\Gamma(1/3)}{\sqrt{\pi}}\, e^{-4M(1-\log 2)}, & \dfrac{\lambda}{\mu}=1, \\ \dfrac{\Gamma(1/3)}{2^{5/6}3^{1/6}}\left(\dfrac{M\mu}{\lambda}\right)^{1/6}\dfrac{1}{\sqrt{\pi}}\, e^{-4M\mu(1-\log 2)/\lambda}, & \dfrac{\lambda}{\mu}>1. \end{cases} \tag{5.22}$$

We note that in all three cases, $p(M)$ is exponentially small for $M \gg 1$.

We now compare our results for the E_2 case to the exact solution (see [4, p. 45]), which we write as

$$p(m)=\frac{p(0)M!}{(M-m)!}\sum_{l=0}^{m-1}\frac{(-1)^l}{l!}\sum_{i=0}^{m-l-1}\left(\frac{\lambda}{2\mu M}\right)^{2(i+1)}$$
$$\cdot\frac{\Gamma(i+M-m+l+4\mu M/\lambda)}{(l-m-i-1)!\Gamma(M-m+l+4\mu M/\lambda)},\qquad m=1,2,\cdots,M,$$
$$p(0)=\left(1+\frac{\lambda}{\mu}I\right)^{-1},$$
$$I=\sum_{i=0}^{M-1}\frac{(M-1)!}{(M-1-i)!}\left(\frac{\lambda}{2\mu M}\right)^{2i}\frac{\Gamma(i+1+4\mu M/\lambda)}{\Gamma(1+4\mu M/\lambda)}.$$

To compare our formulas for $p(0)$ ((5.7), (5.10), (5.13)) to the above, we expand I for $M\gg 1$. Using the identities

$$\Gamma(i+1+4\mu M/\lambda)=\int_0^\infty t^{i+4\mu M/\lambda}e^{-t}\,dt,$$
$$\sum_{i=0}^{M-1}\frac{(M-1)!}{(M-1-i)!}x^{M-1-i}=\int_0^\infty e^{-t}(t+x)^{M-1}\,dt,$$

we obtain

$$I=\left(\frac{\lambda^2}{4\mu^2}\right)^{M-1}\frac{M^2M^{4\mu M/\lambda}}{\Gamma(1+4\mu M/\lambda)}\int_0^\infty\int_0^\infty\frac{e^{-MF(x,y)}}{xy+4\mu^2/\lambda^2}\,dx\,dy,$$
$$F(x,y)=x+y-\frac{4\mu}{\lambda}\log y-\log\left(xy+\frac{4\mu^2}{\lambda^2}\right).$$

Thus, I is expressed in terms of a two-dimensional Laplace type integral. The stationary points of the integral are defined by the conditions $\partial F/\partial x=\partial F/\partial y=0$. An elementary calculation shows that if $\lambda/\mu<1$, then there is no stationary point of F in $(0,\infty)\times(0,\infty)$. If $\lambda/\mu=1$, then there is a stationary point on the boundary of the first quadrant at $x=0$, $y=4$. If $\lambda/\mu>1$, then there is an interior stationary point at

$$x_0=\frac{1}{2}\left(1-\frac{4\mu}{\lambda}+\sqrt{1+\frac{8\mu}{\lambda}}\right),$$
$$y_0=\frac{1}{2}\left(1+\frac{4\mu}{\lambda}+\sqrt{1+\frac{8\mu}{\lambda}}\right).$$

If $\lambda/\mu<1$, the major contribution to the integral comes from a neighborhood of $x=0$, $y=4\mu/\lambda$ which is where $F(0,y)$ is minimal. Expanding F near $x=0$, $y=4\mu/\lambda$ and using Laplace's method we obtain

$$\int_0^\infty\int_0^\infty\frac{e^{-MF(x,y)}}{xy+4\mu^2/\lambda^2}\,dx\,dy$$
$$\sim\frac{\lambda^2}{4\mu^2}\sqrt{\frac{8\mu\pi}{M\lambda}}\,\frac{\exp\left(-M[(4\mu/\lambda)-(4\mu/\lambda)\log(4\mu/\lambda)-2\log(2\mu/\lambda)]\right)}{M(1-\lambda/\mu)}$$

for $M\gg 1$. Thus, $I\sim 1/(1-\lambda/\mu)$, $M\gg 1$, so that $p(0)\sim 1-\lambda/\mu$, $M\gg 1$, which agrees with our formula (5.7).

If $\lambda/\mu = 1$, then we have

$$\int_0^\infty \int_0^\infty \frac{e^{-MF(x,y)}}{xy+4}\,dx\,dy \sim e^{-4M}4^{5M}\frac{\pi}{M\sqrt{3}}, \qquad M \gg 1,$$

as the major contribution to the integral comes from a neighborhood of $x=0$, $y=4$. With above we have

$$I \sim \sqrt{\frac{2\pi M}{3}}, \qquad M \gg 1 \quad \text{and} \quad p(0) \sim \sqrt{\frac{3}{2M}}, \qquad M \gg 1,$$

which regains formula (5.10).

Finally, when $\lambda/\mu > 1$, we use the formula

$$\int_0^\infty \int_0^\infty \frac{e^{-MF(x,y)}}{xy+4\mu^2/\lambda^2}\,dx\,dy \sim \frac{2\pi\, e^{-MF(x_0,y_0)}}{(x_0y_0+4\mu^2/\lambda^2)M}(\det \mathbf{H})^{-1/2}, \qquad M \gg 1,$$

where

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x^2} & \dfrac{\partial^2 F}{\partial x\,\partial y} \\ \dfrac{\partial^2 F}{\partial y\,\partial x} & \dfrac{\partial^2 F}{\partial y^2} \end{bmatrix}(x_0, y_0),$$

to get

$$I \sim \sqrt{\frac{2\pi M}{\gamma(0)}}\frac{2\mu^{3/2}}{\lambda^{3/2}} \exp\left(M\left[\frac{4\mu}{\lambda}+2\log\frac{\lambda}{2\mu}+\frac{4\mu}{\lambda}\log\frac{\lambda}{4\mu}\right.\right.$$
$$\left.\left.-1-\gamma(0)+\left(1+\frac{4\mu}{\lambda}\right)\log\left(\left(1+\frac{4\mu}{\lambda}+\gamma(0)\right)\Big/2\right)\right]\right),$$
$$\gamma(0) = \sqrt{1+8\mu/\lambda}$$

and $p(0) \sim \mu/\lambda I$, $M \gg 1$.

This again agrees with our formula (5.13) if one notes that

$$\frac{8\mu}{\lambda}\cdot\frac{1}{\gamma(0)-1} = \gamma(0)+1,$$
$$\frac{\gamma(0)+1}{\gamma(0)-1} = \frac{\lambda}{4\mu}\left[1+\frac{4\mu}{\lambda}+\gamma(0)\right].$$

Thus, we have shown that for each case of λ/μ, the formulas we derived using our perturbation methods indeed coincide with the large M asymptotics of the exact expression for $p(0)$.

Appendix 1. Asymptotic expansion in Region (iii). We construct the local expansion in Region (iii) near $n = M$. This proves to be the most difficult region to analyze, but is the least interesting region since for $M \gg 1$ the system spends very little time there, and the probabilities in this region represent only an exponentially small tail of the full stationary solution regardless of the value of λ/μ. In addition the solution in Region (iii) is not needed to compute approximations to the moments of the queue length (cf. § 4). The local solution in Region (iii) is needed to construct uniform asymptotic approximations to the stationary distribution, e.g., to construct accurate approximations to quantities such as $p(M)$, the probability that all M sources are

busy. In constructing uniform approximations, we want our approximations $p_a(n)$ to be close to the exact solutions $p(n)$ in the sense that

$$\max_{0\leqq n\leqq M}\left|1-\frac{p_a(n)}{p(n)}\right| = o(1), \qquad M\to\infty.$$

To satisfy this criterion, the approximate solution in Region (iii) is needed, even though both $p(n)$ and $p_a(n)$ are exponentially small.

As before we introduce a "stretched" variable

$$m = M(1-\xi) \tag{A1.1}$$

and the local function $S_m(\eta)=p(\xi,\eta)$. Using (A1.1) and S_m in (2.7)-(2.11), we obtain the equation

$$S'_m(\eta) = -\mu(\eta)S_m(\eta)+\frac{\lambda(m+1)}{M}S_{m+1}(\eta)-\frac{\lambda m}{M}S_m(\eta), \qquad m\geqq 0 \tag{A1.2}$$

with the jump condition

$$S_m(0)=\int_0^\infty S_{m-1}(z)\mu(z)\,dz, \qquad m\geqq 1 \tag{A1.3}$$

and

$$S_0(0)=0. \tag{A1.4}$$

We note that equations (A1.2)-(A1.4) are those for a $G/M/\infty$ queue if we interchange the roles of the arrival and service processes.

If we let

$$S_m(\eta)=R_m(\eta)\exp\left(-\int_0^\eta \mu(t)\,dt\right) \tag{A1.5}$$

and $\tilde{\lambda}=\lambda/M$ in (A1.2), then we find that $R_m(\eta)$ satisfies

$$R'_m(\eta)=\tilde{\lambda}(m+1)R_{m+1}(\eta)-\tilde{\lambda}mR_m(\eta), \tag{A1.6}$$

$$R_m(0)=\int_0^\infty b(z)R_{m-1}(z)\,dz, \qquad m\geqq 1, \tag{A1.7}$$

$$R_0(0)=0.$$

We introduce the Laplace transformation $T_m(u)=\int_0^\infty e^{-u\eta}R_m(\eta)\,d\eta$ and define $s\equiv u/\tilde{\lambda}$ in (A1.6) to obtain

$$T_{m+1}(u)-\left(\frac{m+s}{m+1}\right)T_m(u)=-\frac{R_m(0)}{\tilde{\lambda}(m+1)}. \tag{A1.8}$$

The solution of (A1.8) is

$$T_m(u)=\frac{1}{m!}\frac{\Gamma(m+s)}{\Gamma(s)}T_0(u)-\frac{\Gamma(m+s)}{m!}\frac{1}{\tilde{\lambda}}\sum_{k=1}^{m-1}\frac{k!R_k(0)}{\Gamma(k+1+s)} \tag{A1.9}$$

where $T_0(u)$ is to be determined. The expression (A1.9) for $T_m(u)$ contains polynomials in s (and hence u). The inverse Laplace transform of a polynomial involves distributions (Dirac delta function and its derivatives) which lead to unacceptable solutions to the problem. To avoid this difficulty, we must choose

$$T_0(u)=\frac{\Gamma(s)}{\tilde{\lambda}}\sum_{k=1}^{\infty}\frac{k!R_k(0)}{\Gamma(k+1+s)}. \tag{A1.10}$$

Using (A1.10) in (A1.9) yields

$$T_m(u) = \frac{\Gamma(m+s)}{m!\tilde{\lambda}} \sum_{k=m}^{\infty} \frac{k!R_k(0)}{\Gamma(k+1+s)}. \tag{A1.11}$$

Inverting the Laplace transformation, we obtain an expression for $R_m(\eta)$ in terms of the $R_k(0)$ as

$$R_m(\eta) = \frac{1}{m!} \sum_{k=m}^{\infty} \frac{k!R_k(0)}{(k-m)!}(1-e^{-\tilde{\lambda}\eta})^{k-m}(e^{-m\tilde{\lambda}\eta}). \tag{A1.12}$$

We obtain an equation for the $R_k(0)$ by using (A1.12) in (A1.7) and defining $\gamma_j^m = \int_0^\infty e^{-m\tilde{\lambda}z}(1-e^{-\tilde{\lambda}z})^j b(z)\,dz$ as

$$R_{m+1}(0) = \sum_{j=0}^{\infty} \binom{m+j}{m} R_{m+j}(0)\gamma_j^m \tag{A1.13}$$

where $\binom{m}{k} = m!/k!(m-k)!$.

The solution of (A1.13) can be obtained by introducing the generating function $G(s) = \sum_{m=1}^{\infty} s^m R_m(0)$ to obtain

$$\frac{1}{s}G(s) = \int_0^\infty b(z)G((s-1)\,e^{-\tilde{\lambda}z}+1)\,dz. \tag{A1.14}$$

The solution of (A1.14) is given by

$$G(s) = c \sum_{k=0}^{\infty} (s-1)^k \left[\frac{1}{\beta_k} \prod_{j=1}^{k} \frac{\beta_j}{1-\beta_j}\right], \tag{A1.15}$$

where

$$\beta_k = \int_0^\infty e^{-k\tilde{\lambda}z} b(z)\,dz \tag{A1.16}$$

and c is an arbitrary constant. Inverting the generating function determines $R_m(0)$ as

$$R_m(0) = c \sum_{l=0}^{\infty} \binom{m+l}{m} \frac{(-1)^l}{\beta_{l+m}} \prod_{j=1}^{l+m} \frac{\beta_j}{1-\beta_j}. \tag{A1.17}$$

Using (A1.18) in (A1.13), we find that

$$R_m(\eta) = c \sum_{l=0}^{\infty} (-1)^l e^{-\lambda\eta(m+l)/M} \frac{(m+l)!}{m!l!}\left[\frac{1}{\beta_{m+l}} \prod_{k=1}^{m+l} \left(\frac{\beta_k}{1-\beta_k}\right)\right]. \tag{A1.18}$$

Thus, the local solution, $S_m(\eta)$, is given by

$$\begin{aligned} S_m(\eta) = c \exp\left(-\int_0^\eta \mu(t)\,dt\right) \sum_{l=0}^{\infty} (-1)^l e^{-\lambda\eta(m+l)/M} \\ \cdot \frac{(m+l)!}{m!l!}\left[\frac{1}{\beta_{m+l}} \prod_{k=1}^{m+l} \left(\frac{\beta_k}{1-\beta_k}\right)\right] \end{aligned} \tag{A1.19}$$

where

$$\beta_k = \int_0^\infty e^{-k\lambda z/M} b(z)\,dz \tag{A1.20}$$

and c is an as yet undetermined constant. This solution must be expanded for $m \gg 1$ and then matched with the WKB solution in order to determine the constant c.

We give the explicit solution for two special cases. First, for the exponential case, i.e., $b(x)=\mu e^{-\mu x}$, we find that $R_m(0)$ given by (A1.17) reduces to

$$R_m(0)=\frac{c}{(m-1)!}\left(\frac{M\mu}{\lambda}\right)^{m-1} e^{-M\mu/\lambda} \tag{A1.21}$$

and hence

$$S_m(\eta)=\frac{c}{m!}\left(\frac{\mu}{\tilde{\lambda}}\right)^m\left(\frac{m\tilde{\lambda}}{\mu}+1-e^{-\tilde{\lambda}\eta}\right)\exp\left(-\left[\frac{\mu}{\tilde{\lambda}}e^{-\tilde{\lambda}\eta}+(\mu+\tilde{\lambda}m)\eta\right]\right). \tag{A1.22}$$

The second case is the 2-stage Erlang, E_2, with $b(x)=(2\mu)^2 x e^{-2\mu x}$. From (A1.16) we obtain

$$\beta_k=\frac{(2\mu)^2}{(2\mu+k\tilde{\lambda})^2}. \tag{A1.23}$$

Using (A1.23) in (A1.17) and the following identity [21]

$$\frac{1}{(1+x)(1+2x)\cdots(1+kx)}=\frac{1}{k!x^{k+1}}\int_0^\infty e^{-y/x}(1-e^{-y})^k\,dy \tag{A1.24}$$

to replace the product $\prod_{k=1}^{m+l}1/(1+k\lambda/4\mu M)$, we obtain

$$R_{m+1}(0)=c\sum_{l=0}^{\infty}\frac{(-1)^l}{m!l!}\left(\frac{\delta}{2}\right)^{2(m+l)}\delta\int_0^\infty e^{-\delta y}\frac{(1-e^{-y})^{m+l}}{(m+l)!}\,dy, \tag{A1.25}$$

where $\delta=4\mu M/\lambda$. Next, we interchange the order of the summation and integration and use the definition of the Bessel function $J_m(2\sqrt{x})=x^{m/2}\sum_{l=0}^{\infty}(-x)^l/l!(m+l)!$ to obtain

$$R_{m+1}(0)=c\frac{\delta}{m!}\left(\frac{\delta}{2}\right)^m\int_0^\infty e^{-\delta y}(1-e^{-y})^{m/2}J_m(\delta\sqrt{1-e^{-y}})\,dy. \tag{A1.26}$$

After changing variables $t=\sqrt{1-e^{-y}}$ and using the identity [22, p. 688]

$$\int_0^1 x^{\nu+1}(1-x^2)^a J_\nu(bx)\,dx=\frac{2^a\Gamma(a+1)}{b^{a+1}}J_{\nu+a+1}(b)\quad\text{with } a=\delta-1,\ b=\delta \text{ and } \nu=m,$$

we obtain

$$R_m(0)=\frac{2c}{(m-1)!}\left(\frac{\delta}{2}\right)^m\left(\frac{2}{\delta}\right)^\delta\Gamma(\delta)J_{m+\delta-1}(\delta). \tag{A1.27}$$

The final expression for $S_m(\eta)$ is given in (5.14).

The final step in the construction of the asymptotic solution is to determine the constant c by matching (A1.19) for $m\gg 1$ and $\eta=O(1)$ to the WKB solution (3.12) for the three different cases of λ/μ. The matching is difficult for arbitrary service densities $b(z)$. There are two forms of b for which the expansion of $S_m(0)$ is easily obtained, specifically, the 2-stage Erlang (E_2) and the hyperexponential (H_2) densities. In the former case, $S_m(0)$ is expressible in terms of Bessel functions while in the hyperexponential case, $S_m(0)$ can be written in terms of confluent hypergeometric functions. The $M/E_2/1$ example was considered in detail in § 5.

Appendix 2. In this appendix, we give the details of the matching of Regions (ii) and (iii) in the example described in § 5.

1) Expansion of $S_m(0)$ for $m \gg 1$.

We first expand $S_m(0)$, given by (5.14) with $\eta = 0$, for $m \gg 1$, $\delta \gg 1$, and $m/\delta = O(1)$. Setting $\eta = 0$ in (5.14) we find that

$$S_m(0) = \frac{2c}{(m-1)!}\left(\frac{\delta}{2}\right)^m \left(\frac{2}{\delta}\right)^\delta \Gamma(\delta) J_{m+\delta-1}(\delta), \tag{A2.1}$$

where $\delta = 4\mu M/\lambda$. We next expand $J_{m-1+\delta}(\delta)$ for $m \gg 1$ and $\delta \gg 1$ with m/δ fixed and bounded away from one to obtain

$$\begin{aligned} J_{m+\delta-1}(\delta) \sim{}& \frac{1}{2}\frac{\gamma(\xi)+1}{\gamma(\xi)-1}\sqrt{\frac{\gamma^2(\xi)-1}{\pi\delta\gamma(\xi)}} \\ & \cdot \exp\left(-\delta\left[\frac{\gamma^2(\xi)+1}{\gamma^2(\xi)-1}\log\frac{\gamma(\xi)+1}{\gamma(\xi)-1} - \frac{2\gamma(\xi)}{\gamma^2(\xi)-1}\right]\right) \end{aligned} \tag{A2.2}$$

where $m = 2\delta/(\gamma^2(\xi)-1)$ and $\gamma = \sqrt{1+8\mu/\lambda(1-\xi)}$. In addition we make use of the following expansions:

$$\begin{aligned} &\frac{1}{(m-1)!}\left(\frac{\delta}{2}\right)^m \sim \frac{\sqrt{m}\, e^m}{\sqrt{2\pi}}\left(\frac{\delta}{2m}\right)^m, \qquad m \gg 1, \\ &\Gamma(\delta) \sim e^{-\delta}\delta^\delta\sqrt{\frac{2\pi}{\delta}}, \qquad \delta \gg 1. \end{aligned} \tag{A2.3}$$

Using $m = 2\delta/(\gamma^2(\xi)-1)$ in (A2.3) we obtain

$$\begin{aligned} \frac{1}{(m-1)!}\left(\frac{\delta}{2}\right)^m \sim{}& \frac{1}{\sqrt{\pi}}\sqrt{\frac{\delta}{\gamma^2(\xi)-1}}\exp\left(\frac{2\delta}{\gamma^2(\xi)-1}\right) \\ & \cdot \exp\left(\frac{2\delta}{\gamma^2(\xi)-1}\left[\log\frac{\gamma^2(\xi)-1}{4}\right]\right). \end{aligned} \tag{A2.4}$$

Now incorporating (A2.2), (A2.3) and (A2.4) in (A2.1) we obtain (5.15).

REFERENCES

[1] L. KLEINROCK, *Queueing Systems, Vols.* 1 *and* 2, John Wiley, New York, 1976.

[2] S. S. LAVENBERG, *Computer Performance Modeling Handbook*, Academic Press, New York, 1983.

[3] L. TAKACS, *Introduction to the Theory of Queues*, Oxford Univ. Press, Oxford, London, New York, 1962.

[4] N. K. JAISWAL, *Priority Queues*, Academic Press, New York, 1968.

[5] H. KOBAYASHI, *Stochastic modeling: queueing models*, in Probability Theory and Computer Science, G. Louchard and G. Latouche, eds., Academic Press, New York, 1983.

[6] J. MCKENNA AND D. MITRA, *Integral representations and asymptotic expansions for closed Markovian queueing networks: normal usage*, Bell Systems Tech. J., 61 (1982), pp. 661-683.

[7] D. MITRA AND J. A. MORRISON, *Asymptotic expansions of moments of the waiting time in a shared-processor of an interactive system*, this Journal, 12 (1983), pp. 789-802.

[8] R. E. O'MALLEY, JR., *Introduction to Singular Perturbations*, Academic Press, New York, 1974.

[9] J. KEVORKIAN AND J. D. COLE, *Perturbation Methods in Applied Mathematics*, Springer-Verlag, Berlin, New York, 1981.

[10] C. M. BENDER AND S. A. ORSZAG, *Advanced Mathematical Methods for Scientists and Engineers*, McGraw-Hill, New York, 1978.

[11] A. MESSIAH, *Quantum Mechanics*, John Wiley, New York, 1961.

[12] J. B. KELLER, *Rays, waves and asymptotics*, Bull. Amer. Math. Soc., 84 (1978), pp. 727-750.

[13] H. E. DANIELS, *Approximate solutions of Green's type for univariate stochastic processes*, J. Roy. Statist. Soc. Ser. B, 22 (1960), pp. 376-401.

[14] ———, *Saddle point approximations in statistics*, Ann. Math. Statist., 25 (1954), pp. 631-650.

[15] D. LUDWIG, *Persistence of dynamical systems under random perturbations*, SIAM Rev., 17 (1975), pp. 605-640.

[16] B. J. MATKOWSKY AND Z. SCHUSS, *The exit problem for randomly perturbed dynamical systems*, SIAM J. Appl. Math., 33 (1977), pp. 365-382.

[17] C. KNESSL, B. J. MATKOWSKY, Z. SCHUSS AND C. TIER, *An asymptotic theory of large deviations for Markov jump processes*, SIAM J. Appl. Math., 45 (1985), pp. 1006-1028.

[18] ———, *Asymptotic analysis of a state-dependent $M/G/1$ queueing system*, SIAM J. Appl. Math., 46 (1986), pp. 483-505.

[19] ———, *On the performance of state-dependent single server queues*, SIAM J. Appl. Math., 46 (1986), pp. 657-697.

[20] J. W. COHEN, *The Single Server Queue*, North-Holland, Amsterdam, New York, 1982.

[21] N. G. DE BRUIJN, *Asymptotic Methods in Analysis*, North-Holland, Amsterdam, New York, 1958.

[22] I. S. GRADSHTEYN AND I. M. RYZHIK, *Tables of Integrals, Series, and Products*, Academic Press, New York, 1980.

[23] D. MITRA, *Waiting time distributions from closed queueing network models of shared processor systems*, in Performance '81, Kylstra, ed., North-Nolland, Amsterdam, New York, 1981, pp. 113-131.

[24] D. MITRA AND J. A. MORRISON, *Asymptotic expansions of moments of the waiting time in closed and open processor sharing systems with multiple job classes*, Adv. in Appl. Probab., 15 (1983), pp. 813-839.

[25] J. A. MORRISON AND D. MITRA, *Heavy traffic asymptotic expansions for the waiting time in closed processor sharing systems with multiple classes*, Adv. in Appl. Probab., 17 (1985), pp. 163-185.

SIAM J. COMPUT.
Vol. 16, No. 2, April 1987

© 1987 Society for Industrial and Applied Mathematics
012

ON THE OPTIMAL CONTROL OF TWO QUEUES WITH SERVER SETUP TIMES AND ITS ANALYSIS*

MICHA HOFRI† AND KEITH W. ROSS‡

Abstract. Two queues are fed by independent, time-homogeneous Poisson arrival processes. One server is available to handle both. All service durations, in both queues, are drawn independently from the same distribution. A setup time is incurred whenever the server moves (switches) from one queue to the other. We prove that in order to minimize the sum of discounted setup charges and holdings costs, assumed linear in queue length and having the same rate at the two queues, the service at each queue should be exhaustive. A "threshold policy" is defined as a policy under which the server switches (from an empty queue) only when the other reaches a critical size. It is shown to be a likely candidate for the optimal policy, both for the discounted version and for the long-time average criterion.

The steady-state performance of this policy (under somewhat more general distributional assumptions) and the optimal thresholds are determined for a number of cases.

Key words. two-queue system, server, setup times, optimal control, MDP, queueing analysis, exhaustive service, threshold switching

AMS(MOS) subject classification. 60

I. Introduction and mathematical model.

1. We consider a system of two service stations. Each is fed from a separate queue, for which there is an unlimited waiting room. Both stations share a single server. Customers arrive independently to the two queues, according to Poisson processes; alternatively, one may view the situation as arising from one stream which is split with fixed probabilities. All customers receive services with durations drawn independently from the same distribution, which is otherwise arbitrary. Service, once started, may not be interrupted till conclusion. When the server wants to change from treating one waiting line to the other, it incurs a setup time, and possibly some other costs as well.

Our goal is to determine an optimal nonpreemptive server assignment policy, that minimizes the sum of customer holding costs and switching charges.

2. Such models arise naturally in numerous contexts. One could cite the following examples of (sub)systems where models of this nature may be suitable:

(a) A printing output device which accommodates two widths of continuous paper forms. Print orders come at random for the two paper types. Changing the paper, the necessary alignment operations, etc. need to be done manually and consume a time which is considerable compared with producing the output of a typical job.

(b) Manufacturing systems, flexible and otherwise, where a machine can perform several functions but requires a setup every time its operation changes, to change tool, clear chuck, etc. Very similar are the models for hospital operating rooms, batch processing computer systems and more.

(c) Computer communications networks, using token protocols, polling protocols and others. There are several elements in such a nontrivial network which the model would fit, though for some purposes the number of *two* stations is too small to be really useful.

* Received by the editors April 7, 1986; accepted for publication (in revised form) July 21, 1986. This research was initiated at the Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France.

† Department of Computer Science, Technion Haifa, 32000, Israel.

‡ Department of Systems Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104.

(d) A signalized intersection of two one-way streets, equipped with detectors that can count the number of arriving vehicles at each street. Here the setup time is a natural consequence of the time required for cars to start passing the intersection once they have the green light.

It is not accidental that the above examples consist all of automatic (computerized) elements. When humans operate under such conditions one sees additional control elements, such as the acceleration of service in the presence of long waiting lines, and reneging (and jockeying) on the part of the customers, and more. In contrast, we only consider the control actions of nonpreemptive server assignment decisions.

3. Correspondingly, such models are no newcomers to the performance evaluation literature. The control issues—namely, the server assignment policy—do not seem to have been addressed in such a context so far. There is an extensive understanding of the optimal assignment policy under similar conditions, except that the server incurs *no time loss* between serving different types of customers, those enqued in distinct queues: see for example Harrison [1975a], where the different classes are allowed to demand differently distributed services, and also Klimov [1974] and Meilijson and Weiss [1977], where customers are allowed to feed-back to the waiting line instantaneously after service completion. In Buyukkoc et al. [1985] the optimal control when the service is preemptible is considered. In Stidham and Prabhu [1973] a fairly complete survey of the available results on the control of queues is given. The paper by Baras et al. [1985] updates this survey to some extent. The unpublished survey by Stidham [1969] is the only where we found a mention of threshold switching ("nonzero switch rule" is the term he uses). The advisability of such a policy is briefly discussed, but no results are reported, and their lack lamented.

4. The analysis of these models presents a rather similar picture: fairly complete and elaborate analyses of similar situations, several of the works not very recent, but with one feature missing. See for example Eisenberg [1971, 1972], Ferguson and Aminetzah [1985], Konheim and Meister [1974], Watson [1984], Boxma [1984]. The missing feature is that in all these works the itinerary of the server, the sequence of the types to which successive services (or busy-periods of services, where the server exhausts each queue before departing) is pre-determined; in particular, it does not depend on the state of the various queues. The last work cited above goes beyond this limitation in a certain sense (after each busy period there is a rest period, at the end of which, if only one queue is nonempty it is served exhaustively; if both are, there is a contention period, which one of the queues "wins" ultimately and is served; the outcome of the contention does not depend however on the queue lengths beyond the fact that they nonzero).

5. In contrast, we shall show that the optimal policy in the setup described herein dictates to the server a switching sequence that *does* depend on the lengths of the queues. Obtaining then the distributions of the queue lengths and waiting times involves a rather different approach, even when only the *marginal* distributions are needed, not the joint ones.

6. We present now the mathematical model to be used below, and then further discuss our results. The arrivals to the queues are assumed to be governed by independent time-homogeneous Poisson processes with rates λ_1 and λ_2, to queue-1 and queue-2 respectively. The service time distribution is the *same* for both queues, but otherwise arbitrary. (This restriction will be relaxed in the analytical part.) Service durations are denoted by S and the Laplace-Stieltjes transform (LST) of its distribution by $S(\cdot)$. We also assume that a switching time is required for the server to move from one queue to the other. The switching-time distributions in general depend on the direction of

the switch; we denote by C_i and $C_i(\cdot)$ the duration and the LST of its distribution for the switch time to queue i.

7. The decision epochs for the server are service completions, switch completions, and, when the server is idling at one of the queues, arrival instants. It will transpire from the analysis that no additional decision epochs need be considered. At each decision epoch the server may either serve, change (i.e. switch) or idle; the action space is therefore $\mathscr{A} = \{S, C, I\}$. The state of the system at any given decision epoch is determined by the position of the server, and the number of customers in each of the two queues; the state space is therefore $\mathbf{S} = \{1, 2\} \times I \times I$, where I is the set of nonnegative integers. A policy π for this semi-Markov decision problem (S. Ross [1970, Chap. 7]) is a sequence $\pi = (\pi_0, \pi_1, \pi_2 \cdots)$ where π_n is applied at decision epoch number n. Note that π_n may be regarded as a mapping from $(\mathbf{S} \times \mathscr{A})^n \times \mathbf{S}$ into $\mathscr{A}$.

8. Let $X_i^{\pi,z}(t)$ (resp. $X_i^{\pi,z}(n)$) be the number of customers in queue-i at time t (resp. at decision epoch n), given that the initial state is z, when policy π is applied. Similarly, let $U^{\pi,z}(n)$ be the server position at epoch n. The state of the system at epoch n under π with initial state z is given by

$$Z^{\pi}(n) = (U^{\pi,z}(n), X_1^{\pi,z}(n), X_2^{\pi,z}(n)).$$

We consider the problem of minimizing the discounted holding cost and switching charges given by

$$V_{\pi}(z) = E\left[\int_{t=0}^{\infty} e^{-\beta t} |X^{\pi,z}(t)|\, dt + a \int_{t=0}^{\infty} e^{-\beta t}\, dR^{\pi,z}(t)\right], \tag{1.1}$$

where $|x| \overset{\text{def}}{=} x_1 + x_2$ for any vector $x = (x_1, x_2)$, $R^{\pi,z}(t)$ is the number of switches up to time t, and a is the one-time charge levied at the start of each switch. Note that the holding costs are assumed to be charged at the same rate in both queues, and the switch charge is also the same in both directions (although the latter is not a material assumption). The holding cost rate is assumed to be 1, thus normalizing the cost units.

A policy π' is said to be optimal if it attains

$$V(z) = V_{\pi'}(z) = \inf_{\pi} V_{\pi}(z) \quad \text{for all } z \in S. \tag{1.2}$$

We will also consider the problem of minimizing the long-run average cost:

$$\bar{V}_{\pi} = \limsup_{t} \frac{1}{t} E\left[\int_{s=0}^{t} |X^{\pi,z}(s)|\, ds + aR^{\pi,z}(t)\right]. \tag{1.3}$$

9. In § II we show, for the discounted criterion, that the optimal policy is exhaustive; i.e., if the server is at a given queue, it is best to remain there at least until that queue is emptied. It is this result that requires the sameness of the service distribution at the two queues (indeed, it is not difficult to see that with these durations far from equal, exhaustive service can be a *very* poor policy). It then only remains to specify the assignment policy when the server is attached to an empty queue. In § III it is argued that the server should remain in this queue, either idling or serving newly arrived customers, and should not switch unless, when it idles, the other queue length achieves or exceeds a predetermined threshold. Such a policy is then completely specified by the two thresholds m_1 and m_2 for switching to queue-1 and queue-2, respectively. Note that each of these thresholds may turn out to be 0, in which case an idle server begins to switch immediately. In § IV it is further argued that there exists such a double-threshold policy for the long-run average criterion. Finally, in § V, a

queueing-theoretical analysis of the system under such a policy is described, and then employed to find the optimal threshold values. We remark that this analysis, and in particular the numerical analysis it involves, has some novel features.

II. Exhaustive service is optimal.

1. We begin by showing, after Harrison [1975], that the optimization criterion (1.1) is equivalent to another criterion that is more amenable to analysis. Since our proofs are essentailly computational, this is of importance. Observe that for $z = (u, x_1, x_2)$

$$X_i^{\pi,z}(t) = x_i + A_i^{\pi,z}(t) - D_i^{\pi,z}(t), \tag{2.1}$$

where $A_i^{\pi,z}(t)$ and $D_i^{\pi,z}(t)$ are the arrival and departure processes for queue i.

Combining (1.1) and (2.1), we obtain (see the above reference)

$$\begin{aligned} V_\pi(z) &= \frac{\lambda_1+\lambda_2}{\beta^2} + \frac{x_1+x_2}{\beta} - E\left[\int_{t=0}^{\infty} e^{-\beta t}|D^{\pi,z}(t)|\,dt - aR^{\pi,z}(\beta)\right] \\ &= \frac{\lambda_1+\lambda_2}{\beta^2} + \frac{x_1+x_2}{\beta} - \frac{1}{\beta}E\left[\sum_{k\geqq 1} e^{-\beta T_k^{\pi,z}} - a\beta R^{\pi,z}(\beta)\right], \end{aligned}$$

where $T_k^{\pi,z}$ is the kth departure instant from the queueing system, and $R^{\pi,z}(\beta)$ is the LST of $R^{\pi,z}(t)$. Our optimization criterion is therefore equivalent to maximizing

$$J_\pi(z) \equiv E\left[\sum_{k\geqq 1} e^{-\beta T_k^{\pi,z}} - \beta a R^{\pi,z}(\beta)\right]. \tag{2.2}$$

Write $J(z)$ for the supremum of $J_\pi(z)$ over all policies. It follows immediately from Lippman [1973] that there is an optimal pure policy $g: \mathbf{S} \to \mathscr{A}$ that achieves $J(z)$.

2. A pure policy g is said to be an exhaustive service policy if, at any given decision epoch, it satisfies the following property: if the server is at a nonempty queue, then the server serves a customer from that queue. In other words, an exhaustive service policy never idles a server at a nonempty queue, nor does it ever allow such a queue to be abandoned. We show that if g is nonexhaustive it is suboptimal: there exists a $z \in S$ such that $J_g(z) < J(z)$.

3. Let g be nonexhaustive. Then there is some state z such that g either idles at a nonempty queue or lets the server abandon it. Without loss of generality, suppose this occurs at queue-1. Then $g(1, x_1, x_2) \neq S$ with $x_1 > 0$. For the remainder of this section we shall suppose the initial state to be given by this $z_0 = (1, x_1, x_2)$, and shall therefore write $Z^\pi(t)$ for $Z^{\pi,z_0}(t)$.

In order to show that g is suboptimal, we need to introduce a probability space to support the participating processes. Let (Ω, f, P) support

(i) The independent Poisson processes $N_1(t)$, $N_2(t)$, $\bar{N}_1(t)$, $\bar{N}_2(t)$, where $N_1(t)$, $\bar{N}_1(t)$ have rate λ_1 and $N_2(t)$, $\bar{N}_2(t)$ have rate λ_2. We shall write $N(t) = (N_1(t), N_2(t))$ and $\bar{N}(t) = (\bar{N}_1(t), \bar{N}_2(t))$.

(ii) The sequences of i.i.d. random variables $\{S_n\}$ and $\{T_n\}$, where each S_n corresponds to a service duration for a queue-1 customer and each T_n similarly for one from queue-2. Note that *both* sequence elements are drawn from the same distribution.

(iii) The sequences of i.i.d. random variables $\{C_{1,n}\}$ and $\{C_{2,n}\}$, where each $C_{1,n}$ (resp. $C_{2,n}$) corresponds to a switching time to queue-1 (resp. to queue-2).

4. Consider the queueing system described in the Introduction with the following modification. Let the arrival process be $N(t)$ up to the first instant that the server begins to serve a customer from queue-1. While serving that customer, let the arrival

process be $\bar{N}(t)$. After completing that service at queue-1 let the arrival process revert to $N(t)$ indefinitely. In other words, when the system is governed by policy π, denote by τ^π the first instant that service is offered to a customer from queue-1, then the arrival process $A^\pi(t)$ is given by

$$A^\pi(t)=\begin{cases} N(t), & 0\leqq t\leqq \tau^\pi, \\ N(\tau^\pi)+\bar{N}(t-\tau^\pi), & \tau^\pi\leqq t\leqq \tau^\pi+S_1, \\ \bar{N}(S_1)+N(t-S_1), & t\geqq \tau^\pi+S_1. \end{cases} \tag{2.3}$$

It is clear that for any fixed π, the multivariate process $A^\pi(t)$ is a Poisson process with rates (λ_1, λ_2). Moreover, it is simple to show that $A^\pi(t)$ is independent of $\{S_n\}$, $\{T_n\}$, $\{C_{1,n}\}$, and $\{C_{2,n}\}$. Therefore the system described in the Introduction is equivalent to one driven by the arrival process $A^\pi(t)$, service processes $\{S_n\}$, $\{T_n\}$, and switching processes $\{C_{1,n}\}$ and $\{C_{2,n}\}$.

The seemingly peculiar fabrication of $A^\pi(t)$, as well as the artificial distinction between $\{S_n\}$ and $\{T_n\}$ are for technical reasons in the following proof.

5. PROPOSITION 2.1. *Let at least one of the three values a, $E(C_1)$ and $E(C_2)$ be greater than zero; then any nonexhaustive policy g is suboptimal.*

Proof. Consider the policy π, beginning in state z_0, that behaves as follows. First, π provides a service for a customer from queue-1; then the actions of π follow those of g, with a lag of S_1, just until the completion of the *first* queue-1 customer under g, which occurs at time τ^g+S_1. From τ^g+S_1 onward, the actions under π are those of g (without the delay). We shall show that π is feasible and improves on g (though for nearly all g it would still be nonoptimal as well). Observe by (2.3) that

$$A^\pi(t)=\begin{cases} A^g(t+\tau^g)-A^g(\tau^g), & 0\leqq t\leqq S_1, \\ A^g(t-S_1)+A^\pi(S_1), & S_1\leqq t\leqq \tau^g+S_1, \\ A^g(t), & t\geqq \tau^g+S_1, \end{cases} \tag{2.4}$$

so that $A^\pi(t)-A^\pi(S_1)$ lags behind $A^g(t)$ for $S_1\leqq t\leqq S_1+\tau^g$. Moreover, the sequence of switching times and service times for g when $0\leqq t\leqq \tau^g$ corresponds exactly to those used by π during $S_1\leqq t\leqq \tau^g+S_1$. Therefore π as defined is feasible. Due to the stochastic coupling between π and g we have

$$D^\pi(t)=\begin{cases} 0, & 0\leqq t<S_1, \\ 1+D^g(t-S_1), & S_1\leqq t<\tau^g+S_1, \\ D^g(t), & t\geqq \tau^g+S_1. \end{cases} \tag{2.5}$$

It follows from (2.4) and (2.5) that

$$Z^\pi(t)=Z^g(t) \quad \text{for } t\geqq \tau^g+S_1.$$

The rest of the proof is computational, as we calculate the difference $J_\pi(z_0)-J_g(z_0)$:

$$\begin{aligned} J_\pi(z_0)-J_g(z_0)&=\beta E\int_{t\geqq 0} e^{-\beta t}[D^\pi(t)-D^g(t)]\,dt \\ &\quad -a\beta E\int_{t\geqq 0} e^{-\beta t}[dR^\pi(t)-dR^g(t)]. \end{aligned} \tag{2.6}$$

6. Let σ^g be the first time that g, following the state z_0, switches from queue-1 to queue-2. We shall show that (2.6) is strictly positive over each of the three sets

$$\Lambda_1=\{\tau^g<\sigma^g\}, \quad \Lambda_2=\{\tau^g=\sigma^g=\infty\}, \quad \Lambda_3=\{\tau^g\geqq\sigma^g<\infty\}.$$

Clearly, the union of these three sets is the entire sample space. In the following we assume that either $E(C_2)>0$ or $a>0$. The case corresponding to either $E(C_1)>0$ or $a>0$ can be proved in a similar manner.

7. Suppose $\tau^g<\sigma^g$, i.e., g has a customer from queue-1 served before switching over to queue-2 (which implies $g(z_0)=I$). In this case

$$R^\pi(t)=R^g(t), \qquad t\geqq 0$$

and also

$$D^\pi(t)=\begin{cases} D^g(t)+1, & t\in[S_1,\ \tau^g+S_1], \\ D^g(t), & t\notin[S_1,\ \tau^g+S_1]. \end{cases}$$

Therefore

$$a\beta E\left[\int_{t\geqq 0} e^{-\beta t}[dR^\pi(t)-dR^g(t)]\,|\,\Lambda_1\right]=0$$

and

$$\begin{aligned}\beta E\left[\int_{t\geqq 0} e^{-\beta t}[D^\pi(t)-D^g(t)]\,dt\,|\,\Lambda_1\right] &= E[e^{-\beta S_1}(1-e^{-\beta\tau^g})\,|\,\Lambda_1] \\ &= E[e^{-\beta S_1}]E[1-e^{-\beta\tau^g}\,|\,\Lambda_1] \\ &\geqq S(\beta)\frac{\beta}{\beta+\lambda_1+\lambda_2},\end{aligned}$$

where the last inequality follows from the fact that τ^g must be at least as great as the time of first arrival. Thus (2.6) is strictly positive over Λ_1.

8. Now assume $\tau^g=\sigma^g=\infty$. In this case a customer is never served and a switch never made under the uninteresting policy g. Therefore

$$a\beta E\left[\int_{t\geqq 0} e^{-\beta t}[dR^\pi(t)-dR^g(t)]\,|\,\Lambda_2\right]=0$$

and

$$\beta E\left[\int_{t\geqq 0} e^{-\beta t}[D^\pi(t)-D^g(t)]\,dt\,|\,\Lambda_2\right]=E[e^{-\beta S_1}]=S(\beta).$$

Thus (2.6) is strictly positive over Λ_2 as well.

9. Suppose $\tau^g\geqq\sigma^g<\infty$. This implies that g switches to queue-2 before serving a customer at queue-1 (quite a few different behavior patterns for g are subsumed here). In this case up to time τ^g+S_1 the jumps in $R^\pi(t)$ lag those of $R^g(t)$ by S_1; thereafter $R^\pi(t)=R^g(t)$. Therefore

$$\begin{aligned}a\beta E\left[\int_{t\geqq 0} e^{-\beta t}[dR^\pi(t)-dR^g(t)]\,|\,\Lambda_3\right] &\geqq a\beta E[e^{-\beta\sigma^g}(1-e^{-\beta S_1})\,|\,\Lambda_3] \\ &\geqq a\beta[1-S(\beta)]E[e^{-\beta\sigma^g}\,|\,\Lambda_3],\end{aligned} \tag{2.7}$$

which in turn is always nonnegative, and is strictly positive if $a>0$. Consider now the second term of (2.6), over Λ_3. Using (2.5), we can write

$$\begin{aligned}&\beta E\left[\int_{t\geqq 0} e^{-\beta t}[D^\pi(t)-D^g(t)]\,dt\,|\,\Lambda_3\right] \\ &=\beta E\left[\int_{t=0}^{S_1+\tau^g} e^{-\beta t}[D^\pi(t)-D^g(t)]\,dt\,|\,\Lambda_3\right] \\ &=\beta\int_{t\geqq 0} e^{-\beta t}\sum_{k\geqq 1}[P[D^\pi(t)\geqq k,\Psi\,|\,\Lambda_3]-P[D^g(t)\geqq k,\Psi\,|\,\Lambda_3]]\,dt,\end{aligned} \tag{2.8}$$

where the second equality arises from replacing the expectation operator by summation over the complementary distribution function and enjoining the event $\Psi = \{t \leqq S_1 + \tau^g\}$. It remains to show that (2.8) is nonnegative, and strictly positive when $E(C_2) > 0$. This will follow upon showing

$$\text{(a)}\quad P[D^\pi(t) \geqq k, \Psi \mid \Lambda_3] \geqq P[D^g(t) \geqq k, \Psi \mid \Lambda_3] \text{ for all } t \geqq 0,\ k \geqq 0;$$

(2.9) (b) If $E(C_2) > 0$, then there is an $\varepsilon > 0$ such that

$$P[D^\pi(t) \geqq 1, \Psi \mid \Lambda_3] \geqq P[D^g(t) \geqq 1, \Psi \mid \Lambda_3] + \varepsilon$$

for all t in some open interval. To establish (2.9), introduce the notation $U_i \equiv T_i^g - T_{i-1}^g$, with $T_0^g \equiv 0$. Then (2.9b) would be implied by

$$P(S_1 \leqq t, \Psi \mid \Lambda_3) \geqq P(U_1 \leqq t, \Psi \mid \Lambda_3) + \varepsilon$$

for some $\varepsilon > 0$ and t in an open interval. This clearly holds, since U_1 includes the service time of a request as well as an idling or switching time.

Moreover, (2.9a) is equivalent to

$$\text{(2.10)}\qquad P(U_1 + U_2 + \cdots + U_{k-1} + S_1 \leqq t) \geqq P(U_1 + U_2 + \cdots + U_k \leqq t).$$

Since, as above, U_k is stochastically larger than S_1, this inequality indeed holds.

Note that the equality of the rates of the holding costs (or equivalently, service completion awards) for the two queues is crucial for the proof.

Remark. The above result generalizes directly to the case of m queues.

III. The optimal double-threshold policy.

1. For any pure policy g, we know from the previous section that optimality requires $g(1, x_1, x_2) = S$ whenever $x_1 > 0$, and $g(2, x_1, x_2) = S$ whenever $x_2 > 0$. It remains to specify the actions $g(1, 0, x_2)$ and $g(2, x_1, 0)$ for all $x_1, x_2 \in I$.

DEFINITION 3.1. A policy g is said to be a *double-threshold policy* if it is exhaustive and if there exist two nonnegative integers (possibly infinite) such that

$$\text{(3.1)}\qquad g(1, 0, x_2) = \begin{cases} I, & x_2 < m_2, \\ C, & x_2 \geqq m_2, \end{cases}$$

and

$$\text{(3.2)}\qquad g(2, x_1, 0) = \begin{cases} I, & x_1 < m_1, \\ C, & x_1 \geqq m_1. \end{cases}$$

2. In order to prove the optimality of a double-threshold policy, we need to establish some properties of the optimal objective function $J(z)$. The fact that income (and charges) are taken at customer departure and switch beginning epochs only, and the convexity of the discount function provide immediately the following properties:

(1) $J(u, x, y)$ is strictly increasing in both x and y for $u = 1, 2$.

(2) $J(1, x+1, y) - J(1, x, y) \geqq J(2, x+1, y) - J(2, x, y)$,

$J(2, x, y+1) - J(2, x, y) \geqq J(1, x, y+1) - J(1, x, y)$.

$$(3)\quad J(u,x,y)-J(u,x-1,y)\geqq J(u,x+1,y)-J(u,x,y),\qquad u=1,2,$$
$$J(u,x,y)-J(u,x,y-1)\geqq J(u,x,y+1)-J(u,x,y),\qquad u=1,2.$$

Property (1) follows from the observation that a policy may "disregard" a queued request, e.g., operate on the state (u,x,y) as if it were $(u,x-1,y)$, and thereby forgo the advantage of possible service (and earning income). The contribution of R clearly decreases as well when the waiting lines increase, for any exhaustive-service policy. Arguing differently, it would be nonoptimal by the proposition of II.5.

Property (1) implies that the right- and left-hand sides of (2) and (3) are all positive. In both (2) and (3) the left-hand sides represent the same kind of advantage as the right-hand sides do, but earlier in time, and for $\beta>0$ the present value it provides is higher.

Remark 3.2. A computational proof of the properties (2) and (3) is very involved, and except for several particular cases we do not know a complete one.

3. The concavity property (3) provides a proof for a less obvious concavity-type property:

PROPOSITION 3.3. *For all* $x, y\geqq 0$ *we have*

$$(3.3)\qquad J(1,x,y+1)-J(1,x,y)\geqq J(1,x+1,y+1)-J(1,x+1,y),$$

$$(3.4)\qquad J(2,x+1,y)-J(2,x,y)\geqq J(2,x+1,y+1)-J(1,x,y+1).$$

Proof. It suffices to consider (3.3). Since we know that $J(\cdot)$ is realized by an exhaustive service policy, we have

$$J(1,x,y)=E\,e^{-\beta B_1^{*x}}J(1,0,y+B_{12}^{*x})+\text{discounted income during these busy periods,}$$

where B_1 is a random variable with the distribution of a busy-period of queue-1, B_{12} is a random variable with a distribution of the number of arrivals to queue-2 during a B_1 duration, and a $*r$ superscript denotes an r-fold convolution. Evaluating thus separately the sides of (3.3) we find

$$\text{LHS}=E\,e^{-\beta B_1^{*x}}(J(1,0,y+1+B_{12}^{*x})-J(1,0,y+B_{12}^{*x})),$$
$$\text{RHS}=E\,e^{-\beta B_1^{*(x+1)}}(J(1,0,y+1+B_{12}^{*(x+1)})-J(1,0,y+B_{12}^{*(x+1)})).$$

Now we condition the above values on the history of the arrival and service processes just up to the end of the x'th busy period at queue-1. The rest of the computation is under this conditioning; thus B_1^{*x} in the exponent and the B_{12}^{*x} in the $J(\cdot)$ arguments are independent. We cancel $e^{-\beta B_1^{*x}}$ on both sides, call B_{12}^{*x} under this conditioning b_{12} and find that we need to show

$$J(1,0,y+b_{12}+1)-J(1,0,y+b_{12})$$
$$\geqq E_{B_1}[e^{-\beta B_1}(J(1,0,y+1+b_{12}+B_{12})-J(1,0,y+b_{12}+B_{12}))].$$

Since $e^{-\beta B_1}<1$ with probability 1, it suffices to prove the above inequality with this factor removed, which results in the same inequality as in the second part of property (3) above (perhaps iterated) with $x=0$.

THEOREM 3.4. *There exists an optimal double-threshold policy.*

Proof. Suppose not. Then for any optimal policy g there is an integer $m\geqq 0$ that satisfies either

$$(3.5)\qquad g(1,0,m)=C\quad\text{and}\quad g(1,0,m+1)=I$$

or

$$(3.6)\qquad g(2,m,0)=C\quad\text{and}\quad g(2,m+1,0)=I.$$

It suffices to show that (3.5) leads to a contradiction. Define

$$\tau = \inf\{t: Z^{g,z'}(t) = (2, x, 1) \text{ for some } x \geqq 0\}, \tag{3.7}$$

where we define the states $z = (1, 0, m)$ and $z' = (1, 0, m+1)$.

For the initial state z, let π be the policy that behaves as follows: Up to time τ, π pretends that there is an extra customer in queue-2 and follows g; after τ it will still ape g but with no pretending. Note that π is thus feasible.

A straightforward calculation leads to

$$J_\pi(z) = J_g(z') - E[e^{-\beta\tau}(J_g(W') - J_g(W))], \tag{3.8}$$

where

$$W = (2, X_1^{g,z'}(\tau), 0), \tag{3.9}$$

$$W' = (2, X_1^{g,z'}(\tau), 1). \tag{3.10}$$

For the initial state z', let $\tilde{\pi}$ be the policy that immediately switches to queue-2, and then follows g. Define now

$$\eta = \inf\{t: Z^{g,z}(t) = (2, x, 0) \text{ for some } x \geqq 0\}.$$

Note that by (3.5) η is stochastically smaller than τ. Another straightforward calculation provides

$$J_{\tilde{\pi}}(z') = J_g(z) + E[e^{-\beta\eta}(J_g(U') - J_g(U))], \tag{3.11}$$

where

$$U = (2, X_1^{g,z}(\eta), 0), \tag{3.12}$$

$$U' = (2, X_1^{g,z}(\eta), 1). \tag{3.13}$$

Combining (3.8) and (3.11), we obtain

$$\begin{aligned} J_{\tilde{\pi}}(z') - J_g(z') = J_g(z) - J_\pi(z) + E[e^{-\beta\eta}(J_g(U') - J_g(U))] \\ - E[e^{-\beta\tau}(J_g(W') - J_g(W))]. \end{aligned} \tag{3.14}$$

Combining (3.4) in a rearranged form with (3.14), we obtain

$$J_{\tilde{\pi}}(z') - J_g(z') \geqq J_g(z) - J_\pi(z) + E[(e^{-\beta\eta} - e^{-\beta\tau})(J_g(W') - J_g(W))]. \tag{3.15}$$

Now an argument like the one we used for property (1) above provides

$$J_g(2, x, 1) - J_g(2, x, 0) \geqq \varepsilon > 0. \tag{3.16}$$

Therefore

$$J_{\tilde{\pi}}(z') - J_g(z') \geqq \varepsilon E(e^{-\beta\eta} - e^{-\beta\tau}), \tag{3.17}$$

which is strictly positive, and provides the contradiction the proof required.

Note that the switching cost part of the object function makes no explicit appearance here.

IV. Average-cost case.

1. The following two lemmata will be needed in our discussion of the optimality of the double-threshold policy for the long-run average cost:

LEMMA 4.1. *For any given* a, λ_1, λ_2, $S(\cdot)$, $C_1(\cdot)$, $C_2(\cdot)$, *with* $E(C_1) + E(C_2) > 0$ *there exists a* β *sufficiently large such that a* β*-optimal policy* g *shall have* $g(1, 0, y) = g(2, x, 0) = I$ *for all* x, $y \geqq 0$. *If* $E(C_i) = 0$ *the claim only holds for queue-i actions.*

Proof. Consider a policy f that satisfies $f(1, 0, y) = C$. It can certainly do no better thereafter than serve continuously, which implies

$$J_f^\beta(1,0,y) \leqq C_2(\beta) \sum_{k\geqq 1} S^k(\beta) - a \tag{4.1}$$
$$= \frac{C_2(\beta)S(\beta)}{1-S(\beta)} - a.$$

Now consider an exhaustive service policy g that satisfies the condition in the lemma: $g(1, 0, y) = I$ for all $y \geqq 0$. In the proof of the next lemma we shall present the exact earning of this policy, but here it suffices to observe that it earns at least the income from serving one customer, and incurs no switching costs:

$$J_g^\beta(1,0,y) \geqq \frac{\lambda_1}{\lambda_1+\beta} S(\beta). \tag{4.2}$$

The proof will be complete a fortiori if we can show the existence of a β sufficiently large that

$$\frac{\lambda_1}{\lambda_1+\beta} > \frac{C_2(\beta)}{1-S(\beta)}$$

or equivalently

$$\lambda_1 > \lambda_1[S(\beta) - C_2(\beta)] + \beta C_2(\beta). \tag{4.3}$$

But as the right-hand side of (4.3), when $E(C_2) > 0$ vanishes in the limit as $\beta \to \infty$, the required inequality is obtained. Note that it vanishes when $\beta \to 0$ as well.

2. LEMMA 4.2. *For any given a, λ_1, λ_2, $S(\cdot)$, $C_1(\cdot)$, $C_2(\cdot)$, such that $(\lambda_1+\lambda_2)E(S) < 1$, there exists a β sufficiently small and a y sufficiently large such that any β-optimal policy g shall satisfy $g(1, 0, y) = C$.*

Proof. Let f be an exhaustive policy that never switches: $f(1, 0, y) = I$ for all $y \geqq 0$. In Harrison [1975] we find that

$$J_f^\beta \equiv J_f^\beta(1,0,y) = \frac{S(\beta)}{1-S(\beta)} \frac{\lambda_1(1-B_1(\beta))}{\lambda_1(1-B_1(\beta))+\beta}. \tag{4.4}$$

Now consider the policy g under which the server at state $(1, 0, y)$ immediately switches to queue-2, serves the y customers initially queued there (abandoning any additional possible customers waiting then there), returns to queue-1 and stays there forever, earning at least an additional J_f^β (disregarding any arrivals to queue-1 up to the completion of the return switch). Clearly then

$$J_g^\beta \equiv J_g^\beta(1,0,y) \geqq -a + C_2(\beta) \sum_{k=1}^{y} S^k(\beta) - aC_2(\beta)S^y(\beta) + C_1(\beta)C_2(\beta)S^y(\beta)J_f^\beta. \tag{4.5}$$

To show the possibility of having $J_g^\beta > J_f^\beta$, we only consider values of β small enough so that we can replace the LST's by their first order development in β, denoting $E(S)$ by $1/\mu$:

$$C_2(\beta) = 1 - \beta E(C_2), \qquad S(\beta) = 1 - \beta/\mu,$$
$$B_1(\beta) = 1 - \frac{\beta}{\mu - \lambda_1}, \qquad J_f^\beta = \frac{\lambda_1}{\mu} \frac{\mu-\beta}{\beta}.$$

Simplifying in (4.5), it remains to show that there exists a y such that

$$-a\left(2 - \beta E(C_2) - \beta\frac{y}{\mu}\right) + y(1-\beta E(C_2)) - \frac{\lambda_1}{\mu}(\mu-\beta)\left(E(C_1) + E(C_2) + \frac{y}{\mu}\right) > 0. \tag{4.6}$$

This will hold when

$$y(\beta) > \frac{a(2-\beta E(C_2))+(\lambda_1/\mu)(E(C_1)+E(C_2))(\mu-\beta)}{1-(\lambda_1/\mu)+\beta[(\lambda_1/\mu^2)+(a/\mu)-E(C_2)]} \tag{4.7}$$

and such a $y(\beta)$ clearly exists. Note that when $\beta \to 0$ the right-hand side of (4.7) converges simply to

$$y(0) = \frac{2a+\lambda_1(E(C_1)+E(C_2))}{1-(\lambda_1/\mu)}. \tag{4.8}$$

The denominator in (4.8) is positive by the lemma conditions. If we let policy g take advantage of additional service opportunities in queues 1 and 2, on customers that arrive during the server's initial peregrinations, we could make do with a lower limit for $y(0)$ than (4.8) indicates.

3. Let $g(\beta)$ be the optimal double-threshold policy for the discount factor β with the optimal thresholds $m(\beta) = (m_1(\beta), m_2(\beta))$. The previous two lemmata lead to the following conjecture.

PROPOSITION 4.3. *If* $(\lambda_1+\lambda_2)E(S)<1$ *then as* β *approaches zero* $m_i(\beta)$ *decreases monotonically to an integer* m_i, *for* $i = 1, 2$.

Remark 4.4. We have not been able to construct a complete proof of the above proposition.

In the next section we show that under the condition of the proposition the average number of the customers in the system is finite. This, combined with the proposition, enables us to invoke the Tauberian theorem (Widder [1946]):

$$\bar{V}_\pi^{(z)} \geqq \lim_{\beta\to 0} \beta V_\pi^\beta(z) \geqq \lim_{\beta\to 0} \beta V_{g(\beta)}^\beta(z) = \bar{V}_g(z), \tag{4.9}$$

where g is the double threshold policy with threshold $m = (m_1, m_2)$.

It follows from (4.9) that this policy g is optimal for the average criterion, i.e., there exists an optimal double threshold policy for the long-run average cost criterion.

In the next section we analyze the steady-state performance of such a double threshold policy.

V. The optimal threshold values.

1. We present in this section a queueing-theoretical analysis of the two queues under a double-threshold policy. The analysis will provide a procedure to determine the operational characteristics of the system, as a function of the parameters. In particular, the long-run average queue lengths and switching frequency can be computed, and used to locate the optimal threshold values.

We say "locate" rather than "compute," since, as we shall see, it is unlikely that an expression for the values of these optimal values can be produced, save such that indirectly specifies a search over all values.

2. The analysis treats a slightly more general set-up: we shall not require in this section that the services accorded to customers in the two queues have durations drawn from the same distribution. Assuming two different ones does not increase the complexity (or, the assumption of uniformity would not help).

A more detailed (and somewhat extended) analysis may be found in Hofri [1986]. The numerical procedures are outlined there as well. We only present here the components needed to determine the optimal thresholds.

V.1. Specification of the model.

1. Thus we assume a customer from the waiting line i occupies the server for a duration S_i, with distribution $F_i(\cdot)$ and LST $S_i(\cdot)$. Switching between the lines is only

done as defined in § III.1. As in § III.3, the symbol U_{ij} will denote the number of arrivals to line j during a period U_i; $U_{ij}(\cdot)$ denotes its probability generating function (pgf), and we have $U_{ij}(z) = U_i(\lambda_j(1-z))$.

The following symbols will also be employed:

X_i—the number of customers in line i (including the one in service, if there is one).

$r_i(\cdot)$—pmf for X_i at departure epochs from line i.

$G_i(\cdot)$—generating function for the $r_i(\cdot)$.

I_i—the value of X_i when the server starts there a busy period, with pgf $I_i(\cdot)$.

B_i—duration of a busy-period for line i, initiated by one customer, with LST $B_i(\cdot)$.

V_i—interarrival time for line i, with LST $V_i(s) = \lambda_i/(\lambda_i + s)$.

$H_i = B_i + V_i$—a busy-cycle for queue-i.

W_i—Sojourn time for a customer in line i, with LST $W_i(\cdot)$.

2. The analysis of this set-up follows the traditional route of observing the system at suitable epochs, when the states form a Markov chain. The natural epochs are departures, from both queues. It is not hard to write transition probabilities for this chain, but the resultant matrix is quite unwieldy, since the above thresholds induce a complicated interaction between the queues-length processes. Thus we elected a somewhat different approach: the interaction is captured by a "coarser" chain, i.e., one with rarer transitions (essentially at ends of busy periods). This t-chain is defined and analyzed in § V.3. It carries the brunt of the work. Section V.2 shows the marginal distributions of the queue-lengths, which are naturally quite close to those of $M/G/1$ systems.

V.2. The marginal distributions.

1. Consider the system as defined above, and view queue i in isolation. Proceeding precisely as one does for the standard $M/G/1$ queue we observe the chain formed by X_i at departure epochs from that queue.

Assume for the moment the stability of this chain (we shall discuss it in § V.3), we may write for the steady-state probabilities

$$\begin{aligned}\text{Prob}\,(X_i \text{ at departure has length } x) \equiv r_i(x) \\ = \sum_{j=1}^{x+1} r_i(j) P(S_{ii} = x-j+1) \\ + r_i(0) \sum_{j=1}^{x+1} P(I_i = j) P(S_{ii} = x-j+1), \qquad x \geqq 0\end{aligned} \tag{5.1}$$

and the pgf is immediate

$$\begin{aligned}G_i(z) \equiv \sum_{x\geqq 0} r_i(x) z^x = \frac{1}{z} S_{ii}(z)[G_i(z) - r_i(0)] + r_i(0)\frac{S_{ii}(z)}{z} I_i(z) \\ = r_i(0) S_{ii}(z) \frac{I_i(z)-1}{z - S_{ii}(z)}.\end{aligned} \tag{5.2}$$

2. Note that by setting $I_i(z) = z$ the familiar expression for the pgf of the queue length in an $M/G/1$ system reappears. Setting $z = 1$ in (5.2) provides $r_i(0)$ as the reciprocal of the mean number of services performed in a busy-period:

$$r_i(0) = (1-\rho_i)/E(I_i), \qquad \rho_i = \lambda_i E(S_i). \tag{5.3}$$

3. The following usual properties of $M/G/1$ hold here as well:

The sojourn-time distribution LST is given in terms of the above $G_i(\cdot)$:

$$W_i(u) = G_i(1 - u/\lambda_i). \tag{5.4}$$

$G_i(\cdot)$ is also the pgf for the queue length observed by customers arriving to queue i, and hence also the "random time" line length pgf, which is intuitively surprising since $G_i(\cdot)$ is based on relatively sparse information.

4. It only remains to determine $I_i(z)$ to completely specify the marginal distribution of X_i. I_i, however, depends on the evolution of *both* queues, and thus we have to consider now their interaction.

V.3. The two-queue interaction.

1. Since the interaction between the two queues is rather complicated when viewed at successive departure epochs, we use a more parsimonious description that only encodes the states when the queues actually interact. These epochs we call t-points, and there are two types of such points:

End of a busy period (type b point);

Beginning of switching by an idling server (type c point).

A type c point occurs following the event that a server finishes a busy period (itself a type b point), observes that the queue at the other position is below the switching threshold, and therefore idles. However, arrivals to that other queue let it reach the threshold before there is an arrival to the queue where the idling server is stationed, and it starts to switch over. A state of the t-chain will be denoted by (i, x, b or c) with $i \in (1, 2)$ the position of the server, and x—the length of the other queue ($X_i = 0$ then). Note that for a type c point only $x = m_{i'}$ is possible,[1] with i' being the alternate of i. The transition probabilities for this chain may be read off the following equations, which we write for clarity for the specific value of $i = 1$:

$$p(1, x, b) = \sum_{k=0}^{x} p(1, k, b) P(H_{12} = x - k) + p(2, m_1, c) P(C_{12} + B_{12}^{*(m_1 + C_{11})} = x) + \sum_{j \geqq m_1} p(2, j, b) P(C_{12} + B_{12}^{*(j + C_{11})} = x), \qquad 0 \leqq x < m_2, \tag{5.5}$$

where a superscript $*r$ denotes r-fold convolution, and

$$p(1, x, b) = \sum_{i_1=0}^{m_2-1} p(1, i_1, b) \sum_{i_2=0}^{m_2-i_1-1} P(V_{12} = i_2) P(B_{12} = x - i_1 - i_2) + p(2, m_1, c) P(C_{12} + B_{12}^{*(m_i + C_{11})} = x) + \sum_{j \geqq m_1} p(2, j, b) P(C_{12} + B_{12}^{*(j + C_{11})} = x), \qquad x \geqq m_2 \tag{5.6}$$

and lastly

$$p(1, x, c) = \delta(x - m_2) \sum_{j=0}^{m_2-1} p(1, j, b) P(V_{21}^{*(m_2 - j)} = 0). \tag{5.7}$$

To simplify the notation we shall use $p_i(x)$ instead of $p(i, x, b)$.

Note that the C_{11} and C_{12} in (5.5) or (5.6) are dependent through the same realization of C_1. The corresponding equations and transition probabilities for $i = 2$ are obtained from the above by completely symmetrizing over the indices 1 and 2.

2. Define

$$g_1(z) = \sum_{x \geqq 0} p_1(x) z^x \tag{5.8}$$

[1] This chain is "minimal" in the sense that for low enough arrival rates it has no transient states.

and observe that

$$P(V_{12}=k)=\frac{\lambda_1\lambda_2^k}{(\lambda_1+\lambda_2)^{k+1}}, \qquad P(V_{21}^{*k}=0)=\left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)^k. \tag{5.9}$$

Substituting (5.5)-(5.6) and (5.9) into (5.8) and rearranging, we obtain

$$\begin{aligned} g_1(z) &= C_1(\lambda_1(1-B_{12}(z))+\lambda_2(1-z)) \\ &\cdot\left[p(2,m_1,c)B_{12}^{m_1}(z)+g_2(B_{12}(z))-\sum_{j=0}^{m_1-1}p_2(j)B_{12}^j(z)\right] \\ &+\frac{\lambda_1 B_{12}(z)}{\lambda_1+\lambda_2-\lambda_2 z}\sum_{i=0}^{m_2-1}z^i p_1(i)\left[1-\left(\frac{\lambda_2 z}{\lambda_1+\lambda_2}\right)^{m_2-i}\right] \end{aligned} \tag{5.10a}$$

and the dual equation

$$\begin{aligned} g_2(z) &= C_2(\lambda_2(1-B_{21}(z))+\lambda_1(1-z)) \\ &\cdot\left[p(1,m_2,c)B_{21}^{m_2}(z)+g_1(B_{21}(z))-\sum_{j=0}^{m_2-1}p_1(j)B_{21}^j(z)\right] \\ &+\frac{\lambda_2 B_{21}(z)}{\lambda_2+\lambda_1-\lambda_1 z}\sum_{i=0}^{m_1-1}z^i p_2(i)\left[1-\left(\frac{\lambda_1 z}{\lambda_1+\lambda_2}\right)^{m_1-i}\right]. \end{aligned} \tag{5.10b}$$

Denote by $h_i(z)$ the low-order part of $g_i(z)$. Specifically,

$$h_i(z)=\sum_{x=0}^{m_i-1}p_i(x)z^x.$$

Note that substituting $z=1$ in (5.10a) (or (5.10b)) yields

$$p(1,m_2,c)+g_1(1)-h_1(1)=p(2,m_1,c)+g_2(1)-h_2(1), \tag{5.11}$$

which merely states that under steady-state conditions, the frequency of switching from queue 1 to queue 2 is equal to the frequency of switching in the reverse direction. This obvious observation will provide us with an equation for the "boundary" probabilities $p_i(x)$, $x<m_{i'}$. Another equation is supplied by the natural normalization

$$p(1,m_2,c)+p(2,m_1,c)+g_1(1)+g_2(1)=1. \tag{5.12}$$

3. We turn now to the solution of (5.10). To keep the size of the expressions manageable, we shall introduce a raft of functions as we go along.

Define

$$\alpha_i(z)=C_i(\lambda_i(1-B_{ii'}(z))+\lambda_{i'}(1-z)).$$

Substituting (5.7) into (5.10a), we obtain

$$g_1(z)=\alpha_1(z)g_2(B_{12}(z))+\sum_{i=0}^{m_2-1}p_1(i)\beta_{1,i}(z)+\sum_{i=0}^{m_1-1}p_2(i)\gamma_{1,i}(z), \tag{5.13a}$$

with

$$\begin{aligned} \beta_{1,i}(z) &= \frac{\lambda_i z^i B_{12}(z)}{\lambda_1+\lambda_2-\lambda_2 z}\left[1-\left(\frac{\lambda_2 z}{\lambda_1+\lambda_2}\right)^{m_2-i}\right], \\ \gamma_{1,i} &= \alpha_1(z)\left[B_{12}^{m_1}(z)\left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)^{m_1-i}-B_{12}^i(z)\right]. \end{aligned} \tag{5.14}$$

A dual equation is likewise obtained from (5.10b):

$$g_2(z) = \alpha_2(z) g_1(B_{21}(z)) + \sum_{i=0}^{m_1-1} p_2(i)\beta_{2,i}(z) + \sum_{i=0}^{m_2-1} p_1(i)\gamma_{2,i}(z), \tag{5.13b}$$

with the $\beta_{2,i}(z)$ and $\gamma_{2,i}(z)$ being obtained from their correspondents in (5.14) by the exchange $1 \Leftrightarrow 2$. Using (5.13b) to evaluate $g_2(B_{12}(z))$, we get from (5.13a) an equation involving $g_1(\cdot)$ only, with $m_1 + m_2$ "boundary" state probabilities:

$$g_1(z) = \delta_1(z) g_1(\eta_1(z)) + \sum_{i=0}^{m_2-1} p_1(i)\theta_{1,i}(z) + \sum_{i=0}^{m_1-1} p_2(i)\zeta_{1,i}(z), \tag{5.15}$$

with

$$\begin{aligned} &\delta_1(z) = \alpha_1(z)\alpha_2(B_{12}(z)), \qquad \eta_1(z) = B_{21}(B_{12}(z)), \\ &\theta_{1,i}(z) = \beta_{1,i}(z) + \alpha_1(z)\gamma_{2,i}(B_{12}(z)), \qquad \zeta_{1,i}(z) = \gamma_{1,i}(z) + \alpha_1(z)\beta_{2,i}(B_{12}(z)). \end{aligned} \tag{5.16}$$

Clearly a dual equation for $g_2(\cdot)$ is produced when one symmetrizes over 1 and 2 in (5.15)-(5.16).

4. So far we have proceeded with blithe disregard to the question of stability of stationarity of the two-queue system, expecting that analysis under the assumption of ergodicity will produce equations that inter alia display, or at least determine, the conditions under which stationarity will in fact prevail.

We have not been disappointed. Considering (5.15) one may ask: under what conditions does this equation define a function $g_1(z)$ which is analytic for $|z| < 1$ and continuous for $|z| \leqq 1$, as a proper pgf should be? Since when $g_1(\cdot)$ is analytic we may use (5.15) for recursive substitution, one may ask instead what conditions are required to guarantee that such a procedure, continued ad infinitum, yields an analytic result. To this question we have an answer. Let $\eta_1^{(j)}(z)$ denote the jth functional iterate of $\eta_1(z)$, i.e.

$$\eta_1^{(0)}(z) = z, \qquad \eta_1^{(j)}(z) = \eta_1(\eta_1^{(j-1)}(z)). \tag{5.17}$$

Carrying out the recursive substitution procedure, we formally obtain

$$\begin{aligned} g_1(z) = &\lim_{j\to\infty} g_1(\eta_1^{(j)}(z)) \prod_{j\geqq 0} \delta_1(\eta_1^{(j)}(z)) \\ &+ \sum_{i=0}^{m_2-1} p_1(i) \sum_{j\geqq 0} \theta_{1,i}(\eta_1^{(j)}(z)) \prod_{k=0}^{j-1} \delta_1(\eta_1^{(k)}(z)) \\ &+ \sum_{i=0}^{m_1-1} p_2(i) \sum_{j\geqq 0} \zeta_{1,i}(\eta_1^{(j)}(z)) \prod_{k=0}^{j-1} \delta_1(\eta_1^{(k)}(z)). \end{aligned} \tag{5.18}$$

Now the conditions can be stated explicitly: (5.15) is viable iff the following hold:

(a) $\eta_1^{(j)}(z)$, for $|z| \leqq 1$ and $j \to \infty$ converges to a point in the analyticity region of $g_1(z)$. Call this point a. Then we also need

(b) $\delta_1(a) \leqq 1$;

(c) $\theta_{1,i}(a)$, $\zeta_{1,i}(a) < \infty$, and if δ_1 (a) $= 1$, then must both vanish there.

(d) Finally, the limits must be approached fast enough to assure convergence to a continuous function for $|z| \leqq 1$. The last requirement will turn out to be the crucial one.

In the Appendix we show that the above requirements indeed hold when $\eta_1'(1)$ (or $\eta_2'(1)$) is less than 1, which is equivalent to $\rho \equiv \lambda_1 E(S_1) + \lambda_2 E(S_2) < 1$. Then $a = 1$ and the convergence is geometrical.

The condition has a simple intuitive meaning. The definition in (5.16) of $\eta_1(z)$ can be read as

$$\eta_1(z) = E(z^{B_{12}{}^{*B_{21}}});$$

this translates to the verbal statement that $\eta_1(z)$ is the pgf of the number of arrivals to queue-2, during a sequence of busy periods of queue-1 with multiplicity equal to a random variable that has the distribution of the number of arrivals to queue-1 during a (simple) busy period of queue-2. Clearly the expected value derived from this pgf must be less than one to assure stability. Note that the switching times play no role in the stability condition, which is natural for the case of exhaustive service. We may therefore now state the following theorem.

THEOREM 5.1. *When $\rho = \lambda_1 E(S_1) + \lambda_2 E(S_2) < 1$, the t-chain reaches a steady state with the pgf*

$$\begin{aligned} g_1(z) = g_1(1) \prod_{j\geqq 0} \delta_1(\eta_1^{(j)}(z)) + \sum_{i=0}^{m_2-1} p_1(i) \sum_{j\geqq 0} \theta_{1,i}(\eta_1^{(j)}(z)) \prod_{k=0}^{j-1} \delta_1(\eta_1^{(k)}(z)) \\ + \sum_{i=0}^{m_1-1} p_2(i) \sum_{j\geqq 0} \zeta_{1,i}(\eta_1^{(j)}(z)) \prod_{k=0}^{j-1} \delta_1(\eta_1^{(k)}(z)). \end{aligned} \tag{5.19}$$

The dual $g_2(z)$ is obtained by symmetrizing over the indices 1 and 2.

5. To complete the computation, two tasks remain: relating $g_i(\cdot)$ to $I_i(\cdot)$ as required in (5.2), and the evaluation of the $2+m_1+m_2$ unknowns appearing in (5.19) and its dual. The first task is simple, the other is not. We shall tackle them in order.

6. The fraction of queue i busy periods that begin when an idling server receives an arrival—and then $I_i = 1$—is $(h_i(1) - p(i, m_{i'}, c))/g_i(1)$; the fraction that begins after the server completed a switch it started when idling—and then $I_i = m_i + C_{ii}$—is $p(i', m_i, c)/g_i(1)$, and finally there are the busy periods that follow a switch started immediately following a busy period of queue i'; the value of I_i is then $j + C_{ii}$ and their frequency $p_{i'}(j)/g_i(1)$ for $j \geqq m_i$. Thus for a given value of C_{ii}

$$I_i(z) = \frac{1}{g_i(1)}\left[z(h_i(1) - p(i, m_{i'}, c)) + z^{m_i + C_{ii}} p(i', m_i, c) + \sum_{j\geqq m_i} z^{j+C_{ii}} p_{i'}(j) \right]. \tag{5.20}$$

Removing the conditioning on C_{ii} yields

$$I_i(z) = \frac{1}{g_i(1)}[z(h_i(1) - p(i, m_{i'}, c)) + C_{ii}(z)\{p(i', m_i, c) z^{m_i} + g_{i'}(z) - h_{i'}(z)\}]. \tag{5.21}$$

When $m_1 = 0$, the state $(2, 0, b)$ does not necessarily lead to a 1-busy-period, but only with probability $1 - C_{11}(0)$; thus instead of (5.21) we obtain:

$$I_i(z) = \frac{1}{\tau_i}\left[z[h_i(1) - p(i, m_{i'}, c)] + C_{ii}(z) g_{i'}(z) - p_{i'}(0) C_{ii}(0) \right], \qquad m_i = 0, \tag{5.22}$$

with the denominator given by

$$\tau_i = g_i(1) - p_{i'}(0) C_{ii}(0).$$

Note that when $m_i = 0$, $p_{i'}(0)$ is not one of the boundary probabilities computed through the procedure outlined below but has to be evaluated from (5.19).

7. The determination of $g_i(1)$ and $p_i(j)$, $0 \leqq j < m_{i'}$ is is in principle quite straightforward: (5.11) and (5.12), (5.19) (resp., its dual), its first $m_2 - 1$ derivatives (respectively, $m_1 - 1$), all the derivatives evaluated at $z = 0$, provide the required number of

linear equations in those unknowns. The difficulty resides in differentiating anything as unattractive as the right-hand side of (5.19) several times.

Two obstacles had to be overcome, beyond the sheer complexity of the expressions:

(a) The expressions contain infinite sums and products. As mentioned above, these converge exponentially. In practice we iterated $\eta_i(z)$, starting at $z=0$ until it approached $z=1$ to within $\varepsilon = 10^{-8}$, and used the order of the iteration thus determined as the upper limit to all the (infinite) sums and products. It was found that changing $\log_{10}\varepsilon$ in the range -7 to -13 had no effect on the first six significant decimal digits in all the numbers of interest. The required number of iterations rarely exceeded 30, and never reached 100.

(b) The basic functions, $\eta_i(z)$, are available in terms of the LSTs of the busy-period length distributions. These are determined by the equations

$$B_i(u) = S_i(u+\lambda_i(1-B_i(u))). \tag{5.23}$$

In the general case there is no closed-form solution to (5.23). A formal solution is given by

$$B_i(u) = \sum_{j\geqq 1} \frac{(-\lambda)^{j-1}}{j!}(S_i^j(u+\lambda_i))^{[j]}, \tag{5.24}$$

where a bracketed superscript $[j]$ denotes jth order differentiation. From the numerical point of view our experience with (5.24) was disappointing. Although the series converges nicely in principle, the higher order terms—at least when $S(u)$ is a rational or an exponential function—are made of very large and extremely small factors, and overflow or underflow usually intervened before satisfactory convergence was achieved. This can probably be overcome by careful programming, specific for each type of $S(u)$, but it turned out that the following iterative scheme provided a stable and as prompt convergence. Shedding the subscript i, we write

$$B_0(u) = S(u+\lambda), \qquad B_{j+1}(u) = S(u+\lambda(1+B_j(u))). \tag{5.25}$$

Note that (for a fixed u) $B_j(u)$ is monotonically increasing in j.

8. To obtain derivatives of $B(u)$ and the other functions that abound here, one uses the Faa-di-Bruno formula:

$$f(g(z))^{[r]} = \sum_{j=1}^{r} f^{[j]}(g(z)) \sum_{\mathbf{k}} F_{j,r}^{\mathbf{k}} \prod_{l=1}^{r} [g^{[l]}(z)]^{k_l}, \tag{5.26}$$

with the coefficients F given by

$$F_{j,r}^{\mathbf{k}} = \frac{r!}{\prod_{i=1}^{r} k_i!(i!)^{k_i}}$$

and the sum over $\mathbf{k}$ ranges on integer-valued r-dimensional vectors that satisfy

$$\sum_{i=1}^{r} k_i = j, \qquad \sum_{i=1}^{r} ik_i = r, \quad k_i \geqq 0.$$

Applying (5.26) to (5.23) iteratively, and noting that $B^{[r]}(u)$ appears just once in the right-hand side (with coefficient 1), the derivatives of $B(u)$ to any order were easy to compute. Equation (5.26) was used extensively throughout the computation. The entire set of the required vectors $\mathbf{k}$ and coefficients F was prepared in a first phase, to order of $\max(m_1, m_2) - 1$.

9. The analysis was organized to evaluate this policy under a cost structure where the system incurs constant-rate holding charges for any customer in the queues or being serviced, as well as a one-time charge for every switch. The first component averages out to $c_1E(X_1)+c_2E(X_2)$ which is available from the first derivative of $G_i(z)$, as given in (5.2) at $z=1$. To adapt the second component to this scale of average cost per time unit, we need the mean frequency of switching. This can be approached in a number of ways, the simplest being probably as follows:

Consider the t-chain transitions. Let π_i be the probability that the state at a transition point of the chain, selected at random, leads to an immediate switch to queue i (*every* switch starts at a transition point of this chain). Such a switch occurs if the selected point found the chain in the state (i', m_i, c) or (i', x, b) with $x \geqq m_i$. Otherwise one or more t-transitions will be realized before such a switch takes place. Thus

$$\pi_i = p(i', m_i, c) + g_{i'}(1) - h_{i'}(1). \tag{5.27}$$

To convert this to frequency, we need the mean time between transition points of the chain, L. This is straightforward to compute by counting the possible transitions: at queue i, for a c-point, the expected time to the next point (necessarily a b-point in queue i') is $E(C_i)+[m_{i'}+\lambda_{i'}E(C_{i'})]E(B_{i'})$; similar is the case at a b-point when $X_{i'} > m_{i'}$, with $m_{i'}$ in the above expression replaced by the value of $X_{i'}$; at a b-point where the server idles, we have to find the mean time to the first of the two possible transitions: to a b-point in the same queue, or to a c-point there. One obtains, using the symbol α_i for $\lambda_{i'}/(\lambda_1+\lambda_2)$

$$\begin{aligned} L = \sum_{i=1}^{2} \Bigg\{ & (g_i(1)-h_i(1))E(C_{i'})[1+\lambda_{i'}E(B_{i'})]+[g_i'(1)-h_i'(1)]E(B_{i'}) \\ & + \sum_{j=0}^{m_{i'}-1} p_i(j)\Bigg[\alpha_i^{m_{i'}-j}\left(E(C_i)+\frac{m_{i'}-j}{\lambda_1+\lambda_2}+(m_{i'}+\lambda_{i'}E(C_{i'}))E(B_{i'})\right) \\ & + E(B_i)(1-\alpha_i^{m_{i'}-j})+\frac{1}{\lambda_i}[1-(m_{i'}-j+1)\alpha_i^{m_{i'}-j}+(m_{i'}-j)\alpha_i^{m_{i'}-j+1}]\Bigg]\Bigg\}. \end{aligned} \tag{5.28}$$

Using (5.11), the required frequency is clearly given by $2\pi_i/L$, for which all the necessary quantities are given parameters or have been computed.

VI. Discussion and numerical results.

1. The preceding analysis raises some interesting as well as vexing questions. The first concerns the way we chose to analyze the interaction between the two queues. The method works, but only at the expense of a substantial and highly involved amount of software, in order to obtain the boundary probabilities and get any numbers out of the equations we could write. (Note that the computation times were entirely acceptable—the appended tables are extracted from a much larger set that required less than ten minutes on the minicomputer VAX-780.) Furthermore, some 3–4 percent of the software are distribution-specific, and need be redone when the distributions of the service of switching durations are changed. Is there any other approach that would be—if less direct—easier to implement? We have not found one and confess to being highly intrigued by such a possibility.

2. The above analysis handles *two* queues in interaction. What about more? It is easy to see which way our approach generalizes: if the *sequence* of switches the server performs is predetermined, the above method applies. That limitation implies the

following on the part of the server: if it is to serve in the order of stations $(\cdots i, i', i'', \cdots)$, and when it empties queue i it finds $X_{i'} < m_{i'}$, no switching is done, even if $X_{i''} \geqq m_{i''}$. Admittedly this appears to be a peculiar constraint. Even then, however, while the preceding analysis holds, the task of grinding out numbers looks even more daunting. Consider that now, for a mere three queues, the function we need to iterate to solve for $g_1(z)$ has the following representation:

$$\delta_1(z) = \alpha_1(z)\alpha_2(B_{12}(z))\alpha_3(B_{23}(B_{12}(z))).$$

Abandoning the assumption of a deterministic switching sequence (e.g., in the above example letting the server switch directly to i'') leads to a different structure that we have not yet examined, but believe to be considerably harder. We remark that a well-known problem, of doing exact analysis of the policy SSTF used sometimes to manage computer disks may be cast within such a model.

With respect to the optimal control, the optimality of exhaustive service at each queue, as well as the proof we produced, extend easily to a multi-queue environment. The optimal switching sequences are clearly more involved, and we doubt their properties could be captured in a simple characterization.

3. We show some of the results in Tables 1 and 2, giving the sum of the expected queue lengths at departure epochs, and ten times the switching frequency as a function of the thresholds. These are given by the upper and lower numbers in each box. It was interesting to observe that it is not always obvious a priori which thresholds would provide the shorter queues, though the overall behavior is quite intuitive. Thus in the first table the total queue length is minimized with the thresholds $(1, 3)$, whereas in Table 2 at $(0, 4)$. As the switching frequency is monotonically decreasing in both thresholds, higher values for a may shift the optimal values up considerably. In both tables the means of the service times are the same, but in Table 1 the distributions are slightly different. Note the surprisingly pronounced effect that a rather small change in the variance of one of the service distributions produced in the mean switching frequency, compared with the mild effect it has on the mean queue lengths.

4. The proofs of the optimality of the threshold property both in the discounted case and in the long run limit are incomplete, though we managed to reduce the lacunae to dependence on properties of the object function that are highly appealing to our intuition, and thus the remaining doubt in the correctness of the proposition is virtually nil.

We also devised a scheme to compute $J(z)$ under this policy for the discounted version. Naturally it is quite similar to the analysis presented in § 5, and naturally too it is considerably more intricate, and leads to no insight of the dependence of the optimal thresholds on the system parameters. We have no hope to duplicate the feat in Courcoubetis and Reiman [1985], where a closed form equation was found for the critical parameter of the problem there presented. This difference is directly related to the "memory span" of the interaction mechanism underlying the two systems.

Appendix. We wish to prove the conditions specified in § V.3.4 for the existence of a limiting distribution for the t-chain. Consider first the behavior of $\eta_1^{(j)}(z)$. (We need not treat the dual function separately.)

When $\rho_i = \lambda_i E(S_i) < 1$ the busy period in queue i has a proper limiting distribution with the LST $B_i(\cdot)$, and $B_{ii'}$ is well defined too. Thus $\eta_1(z)$ is an analytic function for $|z| < 1$ (at least) and continuous for $|z| \leqq 1$. Observe that, being a pgf, it is completely monotonic—all its derivatives are positive for $0 \leqq z \leqq 1$. When $\eta_1(z)$ is iterated, we get a sequence of analytic functions, which could converge to any solution of the equation $z = \eta_1(z)$. This equation need not have a unique solution. Indeed, if $\eta_1'(1) > 1$ there

TABLE 1

Expected values for constant switching times $C_1 = 10$, $C_2 = 5$, $\lambda_1 = 0.5$, $\lambda_2 = 0.2$ *and services are* $S_i = \alpha_i + U(0, \psi_i)$, $\alpha_1 = 0.7$, $\psi_1 = 0.6$, $\alpha_2 = 0.25$, $\psi_2 = 1.5$. *Upper figure in each box is* $E(X_1 + X_2)$, *the lower is* 10* *mean switching frequency.*

m_1/m_2	0	1	2	3	4	5	6	7	8	9	10
0	7.2865	7.2779	7.2521	7.2250	7.2391	7.3356	7.5303	7.8158	8.1738	8.5854	9.0352
	0.6133	0.6115	0.6029	0.5825	0.5496	0.5082	0.4642	0.4220	0.3837	0.3501	0.3210
1	7.2792	7.2711	7.2465	7.2206	7.2356	7.3325	7.5277	7.8137	8.1721	8.5841	9.0343
	0.6091	0.6074	0.5993	0.5798	0.5478	0.5071	0.4636	0.4216	0.3835	0.3500	0.3210
2	7.2783	7.2711	7.2485	7.2240	7.2378	7.3321	7.5248	7.8095	8.1677	8.5800	9.0307
	0.5940	0.5927	0.5860	0.5691	0.5401	0.5020	0.4604	0.4197	0.3823	0.3493	0.3205
3	7.3253	7.3192	7.2987	7.2735	7.2801	7.3617	7.5413	7.8160	8.1679	8.5769	9.0261
	0.5656	0.5647	0.5598	0.5467	0.5229	0.4900	0.4524	0.4144	0.3790	0.3471	0.3191
4	7.4581	7.4531	7.4349	7.4082	7.4025	7.4606	7.6121	7.8612	8.1936	8.5894	9.0303
	0.5277	0.5271	0.5237	0.5144	0.4964	0.4700	0.4382	0.4047	0.3723	0.3425	0.3159
5	7.6878	7.6840	7.6684	7.6413	7.6239	7.6545	7.7673	7.9758	8.2728	8.6410	9.0619
	0.4860	0.4856	0.4834	0.4770	0.4641	0.4440	0.4185	0.3902	0.3619	0.3351	0.3106
6	8.0033	8.0004	7.9876	7.9620	7.9369	7.9436	8.0167	8.1773	8.7272	8.7545	9.1424
	0.4452	0.4450	0.4436	0.4392	0.4301	0.4153	0.3954	0.3724	0.3484	0.3249	0.3029
7	8.3852	8.3831	8.3729	8.3500	8.3216	8.3113	8.3506	8.4649	8.6638	8.9423	9.2875
	0.4078	0.4076	0.4067	0.4038	0.3974	0.3865	0.3713	0.3529	0.3329	0.3127	0.2933
8	8.8153	8.8137	8.8057	8.7861	8.7575	8.7367	8.7506	8.8254	8.9762	9.2046	9.5024
	0.3745	0.3744	0.3738	0.3719	0.3674	0.3594	0.3477	0.3331	0.3166	0.2994	0.2826
9	9.2791	9.2780	9.2718	9.2553	9.2284	9.2022	9.1984	9.2420	9.3511	9.5327	9.7834
	0.3454	0.3453	0.3449	0.3436	0.3404	0.3345	0.3256	0.3140	0.3004	0.2859	0.2713
10	9.7666	9.7658	9.7610	9.7474	9.7231	9.6947	9.6793	9.6993	9.7741	9.9144	10.1212
	0.3199	0.3199	0.3196	0.3187	0.3165	0.3121	0.3053	0.2960	0.2849	0.2726	0.2599

must be a real solution in (0, 1), since $\eta_1(0) > 0$, $\eta_1(1) = 1$, in addition to $z = 1$ which is a solution as well.

Observe however that if $\eta_1^{(j)}(z) \to a < 1$, then also $\delta_1(\eta_1^{(j)}(z))$, being a pgf, would converge to a number smaller than 1. Since for $z = 1$ the "convergence" is to the value 1, (5.18) would define a function which is discontinuous at $z = 1$. We must reject this possibility, and disallow $\eta_1'(1) > 1$, and demand that $z = 1$ be the unique fixed point for the iteration of $\eta_1^{(j)}(z)$.

This would be satisfied if $\eta_1'(1) \leqq 1$, but in order to have the sums and products in (5.18) converge we must impose a stronger condition: we have, developing around $z = 1$

$$\eta_1^{(j)}(z) = \eta_1^{(j)}(1) + (z-1)\eta_1^{(j)'}(1) + O(z-1)^2$$

or

$$\begin{aligned} |\xi_j| \overset{\text{def}}{=} |\eta_1^{(j)}(z) - 1| &\leqq |z-1| \eta_1^{(j)'}(1) \\ &= |z-1| (\eta_1'(1))^j \end{aligned}$$

TABLE 2

Expected values for constant switching times $C_1 = 10$, $C_2 = 5$, $\lambda_1 = 0.5$, $\lambda_2 = 0.2$ *and services are* $S_i = \alpha_i + U(0, \psi_i)$, $\alpha_i = 0.7$, $\psi_i = 0.6$. *Upper figure in each box is* $E(X_1 + X_2)$, *the lower is* 10^* *mean switching frequency.*

m_1/m_2	0	1	2	3	4	5	6	7	8	9	10
0	7.4385	7.3294	7.1026	6.9047	6.8517	6.9582	7.1902	7.5100	7.8893	8.3089	8.7565
	0.5333	0.5188	0.4780	0.4198	0.3606	0.3093	0.2678	0.2348	0.2084	0.1871	0.1696
1	7.4398	7.3315	7.1051	6.9066	6.8528	6.9587	7.1904	7.5102	7.8893	8.3089	8.7565
	0.5319	0.5177	0.4773	0.4195	0.3605	0.3093	0.2678	0.2348	0.2084	0.1871	0.1696
2	7.4569	7.3510	7.1247	6.9215	6.8620	6.9638	7.1932	7.5116	7.8901	8.3093	8.7567
	0.5259	0.5127	0.4743	0.4180	0.3598	0.3090	0.2677	0.2347	0.2084	0.1871	0.1696
3	7.5217	7.4201	7.1927	6.9758	6.8984	6.9859	7.2059	7.5189	7.8942	8.3116	8.7580
	0.5122	0.5009	0.4665	0.4139	0.3579	0.3081	0.2672	0.2345	0.2082	0.1870	0.1696
4	7.6667	7.5718	7.3439	7.1040	6.9915	7.0472	7.2445	7.5425	7.9086	8.3202	8.7632
	0.4907	0.4817	0.4527	0.4058	0.3537	0.3059	0.2661	0.2339	0.2079	0.1868	0.1695
5	7.9053	7.8195	7.5957	7.3308	7.1694	7.1741	7.3304	7.5992	7.9454	8.3440	8.7783
	0.4638	0.4569	0.4336	0.3937	0.3467	0.3022	0.2640	0.2327	0.2072	0.1864	0.1692
6	8.2305	8.1552	7.9418	7.6586	7.4454	7.3860	7.4843	7.7077	8.0206	8.3955	8.8133
	0.4346	0.4295	0.4112	0.3782	0.3373	0.2966	0.2607	0.2307	0.2060	0.1856	0.1687
7	8.6241	8.5594	8.3619	8.0714	7.8140	7.6881	7.7182	7.8829	8.1493	8.4887	8.8801
	0.4057	0.4019	0.3878	0.3610	0.3260	0.2896	0.2563	0.2279	0.2042	0.1844	0.1679
8	9.0671	9.0124	8.8334	8.5462	8.2575	8.0718	8.0323	8.1312	8.3414	8.6352	8.9903
	0.3785	0.3757	0.3649	0.3433	0.3136	0.2814	0.2510	0.2244	0.2018	0.1828	0.1667
9	9.5437	9.4978	9.3381	9.0618	8.7548	8.5209	8.4172	8.4499	8.5995	8.8409	9.1523
	0.3536	0.3515	0.3433	0.3259	0.3009	0.2726	0.2450	0.2202	0.1988	0.1807	0.1652
10	10.0428	10.0045	9.8633	9.6026	9.2881	9.0181	8.8590	8.8297	8.9190	9.1058	9.3692
	0.3312	0.3297	0.3233	0.3094	0.2884	0.2637	0.2387	0.2157	0.1955	0.1782	0.1634

so that $\eta_1'(1) < 1$ is the required condition to assure convergence. However, $\eta_1'(1) = \rho_1\rho_2/(1-\rho_1)(1-\rho_2)$; hence the condition is

$$\rho = \rho_1 + \rho_2 < 1.$$

This will also guarantee exponential convergence for the coefficient of $g_1(1)$: Let $P_k(z) = \prod_{j=0}^{k-1} \delta_1(\eta_1^{(j)}(z))$, then

$$\frac{P_{j+1}(z)}{P_j(z)} = \delta_1(\eta_1^{(j)}(z))$$

$$= \delta_1(1-\xi_j(z)) = 1 - \delta_1'(1)\xi_j(z) + O(\xi_j^2(z))$$

which approaches the limit geometrically fast. The same computation holds for the sums in (5.19).

Acknowledgment. The hospitality we were accorded at the Instit. National de Recherche en Informatique et en Automatique, Le Chesnay, France, is gratefully acknowledged.

REFERENCES

J. S. BARAS, A. J. DORSEY AND A. M. MAKOWSKI, *Two competing queues with linear costs and geometric service requirements: The μc-rule is often optimal*, Adv. Appl. Probab., 17 (1985), pp. 186-209.

O. BOXMA, *Two symmetric queues with alternating service and switching time*, in Performance '84, Proc. of the 10th International Symposium on Computer Performance Evaluation, Paris, December 1984, E. Gelenbe, ed., North-Holland, Amsterdam, New York, 1984, pp. 409-431.

C. BUYUKKOC, P. VARAIYA AND J. WALRAND, *The cμ rule revisited*, Adv. Appl. Probab., 17 (1985), pp. 237-238.

C. COURCOUBETIS AND M. REIMAN, *The optimal control of a queueing system with simultaneous service requirements*, AT&T Bell Laboratories Research Memorandum draft, July 1985.

M. EISENBERG, *Two queues with changeover times*, Oper. Res., 19 (1971), pp. 386-401.

———, *Queues with periodic service and changeover times*, Oper. Res., 20 (1972), pp. 440-451.

M. J. FERGUSON AND Y. J. AMINETZAH, *Exact results for nonsymmetric token ring systems*, IEEE Trans. Comm., COM-33 (1985), pp. 223-231.

J. M. HARRISON, *A priority queue with discounted linear costs*, Oper. Res., 23 (1975), pp. 260-269.

———, *Dynamic scheduling of a multiclass queue: discount optimality*, Oper. Res., 23 (1975a), pp. 270-282.

M. HOFRI, *Two queues and one server with threshold switching*, in Proc. of the International Seminar on Teletraffic Analysis and Computer Performance Evaluation, O. J. Boxma, J. W. Cohen and H. C. Tijms, eds., Elsevier, Amsterdam, New York, 1986, pp. 409-428.

M. HOFRI AND A. G. KONHEIN, *The analysis of a finite quasi-symmetric ALOHA network with reservation*, in Performance of Computer-Communications Systems, Proc. of the International Seminar on Computer Communications, Zurich, March 1984, W. Bux and H. Rudin, eds., North-Holland, Amsterdam, New York, 1984, pp. 505-525.

G. F. KLIMOV, *Time sharing service systems* I, Theory Probab. Appl., 19 (1974), pp. 532-551.

A. G. KONHEIM AND B. MEISTER, *Waiting lines and times in a system with polling*, J. Assoc. Comput. Mach., 21 (1974), pp. 470-490.

S. LIPPMAN, *Semi-Markov decision processes with unbounded rewards*, Management Sci., 19 (1973), pp. 717-731.

I. MEILIJSON AND G. WEISS, *Multiple feedback at a single server station*, Stochastic Process. Appl., 5 (1977), pp. 195-205.

S. ROSS, *Applied Probability Models with Optimization Applications*, Holden-Day, San Francisco, 1970.

S. STIDHAM, *Optimal control of a signalized intersection. Part* III: *Descriptive stochastic models*, Dept. of Operations Research, Cornell Univ., Ithaca, New York; Technical Report #96 (Dept. of Transportation, Federal Highway Administration Report #13), November 1969.

S. STIDHAM AND N. U. PRABHU, *Optimal control of queueing systems*, in Mathematical Methods of Queueing Theory, Springer-Verlag, Berlin, New York, 1973.

K. S. WATSON, *Performance evaluation of cyclic service strategies—A survey*, in Performance '84, Proc. of the 10th International Symposium on Computer Performance Evaluation, Paris, December 1984, North-Holland, Amsterdam, New York, 1984, pp. 521-534.

D. WIDDER, *The Laplace Transform*, Princeton Univ. Press, Princeton, NJ, 1946.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987
© 1987 Society for Industrial and Applied Mathematics

ON EMBEDDING A GRAPH IN THE GRID WITH THE MINIMUM NUMBER OF BENDS*

ROBERTO TAMASSIA†

Abstract. Given a planar graph G together with a planar representation P, a region preserving grid embedding of G is a planar embedding of G in the rectilinear grid that has planar representation isomorphic to P. In this paper, an algorithm is presented that computes a region preserving grid embedding with the minimum number of bends in edges. This algorithm makes use of network flow techniques, and runs in time $O(n^2 \log n)$, where n is the number of vertices of the graph. Constrained versions of the problem are also considered, and most results are extended to k-gonal graphs, i.e., graphs whose edges are sequences of segments with slope multiple of $180/k$ degrees. Applications of the above results can be found in several areas: VLSI circuit layout, architectural design, communication by light or microwave, transportation problems, and automatic layout of graphlike diagrams.

Key words. planar embedding, grid, VLSI layout, bends, network flow

AMS(MOS) subject classifications. 68U05, 68R10, 68Q35, 94C15, 90B10

1. Introduction. Embedding a graph in the rectilinear grid has several applications. Among them we mention: VLSI circuit design [6], [12], architectural floor plan layout [7], [9], and aesthetic layout of diagrams used in Information Systems design [1], [11].

Given a planar graph G together with a planar representation P, a *region preserving grid embedding* of G is an embedding of G in the rectilinear grid such that

(1) Paths of the grid that are mappings of edges are node-disjoint;

(2) The planar representation of the embedding is isomorphic to P.

We consider the problem of minimizing the number of bends in a region-preserving grid embedding. Figure 1 shows an embedding of the complete 4-vertex graph that minimizes the bend count.

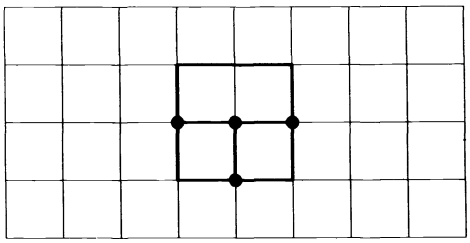


FIG. 1. *A grid embedding with the minimum number of bends.*

This problem has been explicitly considered in a paper of Storer [10], where the existence of a polynomial algorithm for its solution is left as an open problem. We settle this question here by showing an algorithm that finds an embedding with the fewest bends in $O(n^2 \log n)$ time, where n is the number of vertices of the graph. The basic idea of the algorithm is a transformation into a minimum cost flow problem.

The remaining sections of this paper are organized as follows. Section 2 contains basic definitions for a formal statement of the problem, and preliminary results. Section 3 is concerned with the transformation into min-cost-flow, and provides a worst case

* Received by the editors October 3, 1984; accepted for publication (in revised form) June 30, 1986.

† Dipartimento di Informatica e Sistemistica, Universita' di Roma, Via Buonarroti 12, 00185 Rome, Italy.

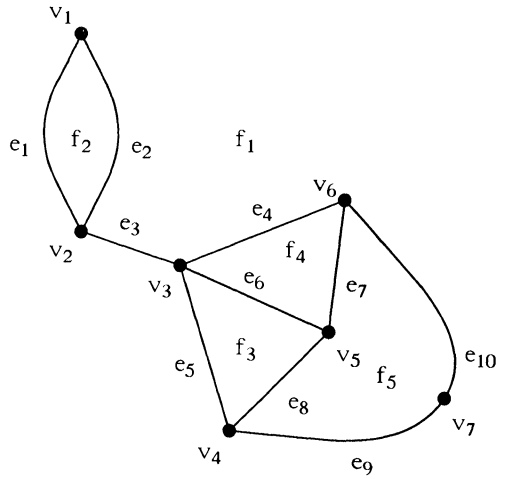
computing time analysis of the embedding algorithm. Section 4 considers constrained versions of the bend minimization problem. Section 5 shows an extension of the above transformation to "k-angular" embeddings, where edges are mapped into sequences of segments with slope multiple of $180/k$ degrees.

2. Definitions and preliminary results. The basic definitions we adopt about graphs are essentially those in [3]. We allow graphs to have multiple edges and self-loops.

A *plane graph* $G=(V, E)$ is defined as follows: (1) V is a set of distinct points of the plane. (2) E is a set of simple curves connecting two points in V. (3) The simple curves in E do not cross each other, except, possibly, at endpoints. Elements in V are called *vertices.* Elements in E are called *edges.* A graph is *planar* if it is isomorphic to some plane graph. A plane graph divides the set of points of the plane that do not belong to edges into topologically connected regions, called *faces.* The unbounded region is referred to as the *external face.* If F is the set of faces of the plane graph $G=(V, E)$, *Euler's formula* states that

$$|V|+|F|=|E|+2.$$

The topological structure of a plane graph can be described by listing edges that appear on the contour of each face, and specifying the external face. We call *planar representation* of a plane graph G a set of circularly ordered edge lists $P(f)$, one for each face f. Edges in $P(f)$ appear in the same sequence as they are encountered when going around the contour of f in the "positive" direction, i.e., having the face at one's right. Note that every edge of G appears exactly twice in the lists. If this happens in the same list $P(f)$, the edge is called *bridge.* An example of planar representation is shown in Fig. 2.



$G=(V, E)$
$V=\{v_1, \cdots, v_7\}$
$E=\{e_1, \cdots, e_{10}\}$
$F=\{f_1, \cdots, f_5\}$
$P(f_1)=(e_1, e_3, e_5, e_9, e_{10}, e_4, e_3, e_2)$
$P(f_2)=(e_2, e_1)$
$P(f_3)=(e_5, e_6, e_8)$
$P(f_4)=(e_6, e_4, e_7)$
$P(f_5)=(e_8, e_7, e_{10}, e_9)$

FIG. 2. *Example of planar representation; edge e_3 is a bridge.*

This definition can be extended to planar graphs as follows: let G be a planar graph; we say that P is a planar representation for G if P is the planar representation of some plane graph isomorphic to G.

The (*rectilinear*) *grid* is the infinite plane graph whose vertices have integer coordinates, and whose edges link pairs of vertices at unit distance. A (*planar*) *grid embedding* [12] of a graph G is a mapping Q of G into the grid such that

(1) Q maps each vertex of G into a distinct point of the grid;

(2) Q maps edge e of G into a path of the grid $Q(e)$ whose endpoints are the mappings of vertices linked by e;

(3) For every pair $\{e', e''\}$ of edges of G, the corresponding paths $Q(e')$ and $Q(e'')$ have no points in common, except, possibly, the endpoints.

The plane graph $Q(G)$ defined by the embedding Q is isomorphic to G. A graph admits a grid embedding if and only if it is planar and has maximum vertex degree less than or equal to 4 [12]. For short, we refer to these graphs as 4-*planar* graphs.

In the following, we consider grid embeddings with a prespecified planar representation. Let $G=(V, E)$ be a planar graph, and P a planar representation for G. The grid embedding Q is called *region preserving* with respect to P if the plane graph $Q(G)$ has planar representation isomorphic to P [10].

We are now able to state formally the problem of our interest.

PROBLEM. *Grid embedding with the minimum number of bends.* Given a 4-planar graph G, and a planar representation P for it, find a grid embedding of G that is region preserving with respect to P and has the minimum number of bends.

Approximation algorithms for the above problem have been given by Storer [10]. Paoluzzi has developed a linear programming formulation for the case of biconnected graphs [9].

The concept of *orthogonal representation*, introduced below, will allow us to deal with the shape of a grid embedding, without taking into account dimensions (a similar concept has been developed by Paoluzzi [9] for biconnected graphs).

An *orthogonal graph* is a plane graph whose edges are alternating sequences of horizontal and vertical segments. Figure 3 shows an orthogonal graph.

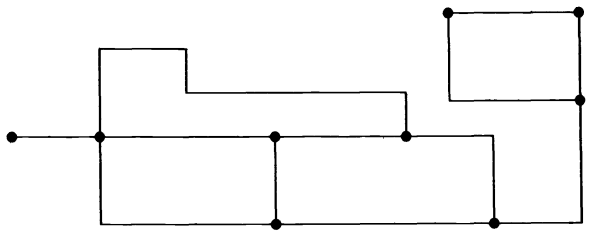


FIG. 3. *Example of orthogonal graph.*

The orthogonal representation describes the shape of an orthogonal graph, without considering segment lengths. This is obtained by enriching the lists of the planar representation with information about angles formed by edges. Let G be an orthogonal graph. The orthogonal representation of G is a set of circularly ordered lists $H(f)$, one for each face of G, whose elements have the format (e, s, a), where

(1) e is an edge of G,

(2) s is a binary string,

(3) a is an integer in the set $\{90, 180, 270, 360\}$.

If r is an element of $H(f)$, its fields are denoted $e[r]$, $s[r]$, and $a[r]$. Each list $H(f)$ can be obtained from the corresponding list $P(f)$ of the planar representation by adding the s- and a-fields to each element, with the following meaning:

(1) String $s[r]$ describes the shape of edge $e[r]$. The kth bit of $s[r]$ represents the kth bend that appears on the right side of $e[r]$, as it is encountered when going around f in the positive direction. The binary symbols **0** and **1** denote an angle of 90 and 270 degrees, respectively.

(2) Number $a[r]$ specifies the angle formed in face f by edges $e[r]$ and $e[r']$, where r' is the element following r in the list $H(f)$.

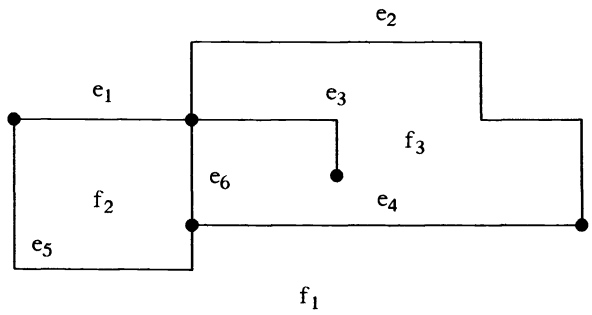
An example of orthogonal representation is given in Fig. 4.

The number of bends $B(H)$ of any orthogonal graph G having H as orthogonal representation is given by

$$B(H)=\frac{1}{2}\sum_{f}\sum_{r\in H(f)}|s[r]| \tag{2.1}$$

where $|s|$ is the length of string s.

For a set of circular lists H to be the orthogonal representation of some orthogonal graph, the following properties are necessary and sufficient, as can be proved by means of elementary geometric considerations.



$H(f_1)=((e_1,\varepsilon,270),(e_5,\mathbf{11},90),(e_4,\varepsilon,270),(e_2,\mathbf{1011},90))$
$H(f_2)=((e_1,\varepsilon,90),(e_6,\varepsilon,180),(e_5,\mathbf{00},90))$
$H(f_3)=((e_2,\mathbf{0010},90),(e_4,\varepsilon,90),(e_6,\varepsilon,90),(e_3,\mathbf{0},360),(e_3,\mathbf{1},90))$

FIG. 4. *Example of orthogonal representation. (The symbol ε denotes the empty string.)*

PROPERTY 1. There is some 4-planar graph whose planar representation is given by the e-fields of the $H(f)$ lists.

PROPERTY 2. Let r' and r'' be a pair of elements in the orthogonal representation associated with the same edge. Then string $s[r'']$ can be obtained from $s[r']$ by reversing $s[r']$ and exchanging zeros with ones.

Property 2 means that each edge must have consistent descriptions in the faces in which it appears.

PROPERTY 3. Define, for each element r,

$$\text{ROTATION}(r)=\text{ZEROS}(s[r])-\text{ONES}(s[r])+(2-a[r]/90)$$

where ZEROS(s) and ONES(s) denote the number of zeros and ones in string s, respectively. The following equality holds:

$$\sum_{r\in H(f)}\text{ROTATION}(r)=\begin{cases}-4 & \text{if } f \text{ is the external face,}\\ +4 & \text{otherwise.}\end{cases}$$

Property 3 means that every face described by H is a rectilinear polygon, i.e. a polygon whose sides are horizontal and vertical segments.

PROPERTY 4. For each vertex v of the plane graph described by H, the sum of angles between pairs of consecutive edges around v, as given by the a-fields of the corresponding elements in the $H(f)$ lists, is equal to 360 degrees.

The orthogonal representation is dimensionless: it defines the reciprocal disposition of a set of horizontal and vertical segments. In order to construct from it a grid embedding, we have to compute the lengths of these segments. Three constraints must be satisfied:

(1) All segment lengths are positive integers;

(2) Every circuit of the graph is mapped into a rectilinear polygon;

(3) No two segments intersect each other, except, possibly, at common endpoints.

An algorithm that accomplishes this task is outlined below.

ALGORITHM FIND_LENGTHS.

Input: Orthogonal representation H of a graph G.

Output: Grid embedding Q of G with orthogonal representation H.

Step 1. Construct a normalized orthogonal representation H' decomposing each face of H into rectangles with the introduction of fictitious edges and vertices.

Step 2. Assign consistently integer lengths to the sides of the rectangles in H', in order to obtain a grid embedding Q' of the graph G' described by H'.

Step 3. Remove from Q' the fictitious vertices and edges introduced by Step 1. This gives the desired grid embedding Q.

Step 1 guarantees that no intersection will occur (constraint 3). It is a shape decomposition, since it operates at the orthogonal representation level. Step 1 can be carried out associating with each face f of H a circular binary string $S(f)$ given by

$$S(f) = s[r_0]w(a[r_0]) \cdots s[r_{n-1}]w(a[r_{n-1}])$$

where

$$H(f) = r_0, r_1, \cdots, r_{n-1},$$

$$w(a) = \begin{cases} \mathbf{0} & \text{if } a = 90, \\ \varepsilon & \text{if } a = 180, \\ \mathbf{1} & \text{if } a = 270, \\ \mathbf{11} & \text{if } a = 360. \end{cases}$$

The circular string $S(f)$ is searched for the pattern **100** and, if found, a rectangle is cut from f performing one of the two basic decomposition operations shown in Fig. 5. $S(f)$ is then updated substituting **0** for **100**, and the procedure is repeated until f becomes a rectangle. A slightly different treatment is given to the external face, which is decomposed into a set of rectangles plus the complement of a rectangle by means of a similar procedure.

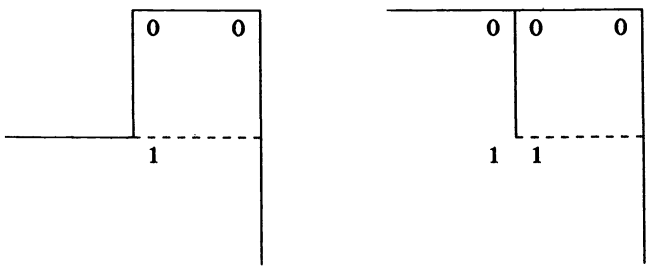


FIG. 5. *Basic decomposition operations.*

An efficient way to perform the above decomposition strategy is to stack the ones encountered while searching for **100**. This makes it possible to decompose each face f into at most $\text{ONES}(f)+1$ rectangles in $O(|H(f)|+|S(f)|)$ time. Thus we have the following lemma.

LEMMA 1. *Step* 1 *of Algorithm* FIND_LENGTHS, *applied to an orthogonal representation H with n vertices and b bends, constructs a normalized orthogonal representation H' with $O(n+b)$ faces in $O(n+b)$ time.*

Step 2 can be efficiently performed using the compaction algorithm described in [4], whose complexity is linear in the number of faces of H'. Step 3 is easily carried out in time linear in the number of faces of H'.

THEOREM 1. *Algorithm* FIND_LENGTHS *embeds in the grid an orthogonal representation with n vertices and b bends in $O(n+b)$ time.*

Proof. The result follows immediately from Lemma 1 and the above discussion. Q.E.D.

In the rest of this section, we recall some definitions about network flow theory.

A *network* $N=(U, A, s, t, b, c)$ consists of

(1) A finite set of *nodes* U;

(2) Two distinguished nodes, s and t, called *source* and *sink*, respectively;

(3) A set A of ordered pairs of vertices, called *arcs*, that includes the special *return arc* (t, s);

(4) A positive *capacity* (*bound*) $b(a)$, for each arc a;

(5) A *cost* $c(a)$, for each arc a.

A *flow* in N is a function $x: A \to \mathbf{R}$ such that

(1) $0 \leqq x(a) \leqq b(a)$ for every arc a, and

(2) $\sum_u x(u, v) - \sum_w x(v, w) = 0$ for every node v

where each sum is over every node $u(w)$ for which the summand is defined.

The *value* of flow x, denoted $|x|$, is given by the flow in the return arc

$$|x| = x(t, s).$$

The *cost* of flow x is the quantity:

$$\text{COST}(x) = \sum_{a \in A} c(a)x(a).$$

The minimum cost flow problem asks for a flow of minimum cost among those with a given fixed value.

PROBLEM. MIN-COST-FLOW. Given a network N, and a positive number z, find a flow x in N of value $|x| = z$ that has minimum cost.

The *augmentation algorithm* for problem MIN-COST-FLOW consists of augmenting the initial zero flow along minimum cost paths [5]. In the case of integer capacities and nonnegative costs, the shortest path computations can be performed using Dijkstra's algorithm implementation with priority queues. Therefore, the augmentation algorithm has complexity $O(z(|U|+|A|)\log|U|)$.

A *flow-augmenting cycle* C with respect to flow x is a cycle in the undirected graph resulting from N by ignoring arc directions and removing the return arc (t, s), with the following properties:

(1) For every arc a traversed by C in the forward direction (called a *forward arc*), we have $x(a) < b(a)$;

(2) For every arc a traversed by C in the reverse direction (called a *backward arc*), we have $x(a) > 0$.

The *cost* of the flow-augmenting cycle C is given by the sum of the costs of the forward arcs minus the sum of the costs of the backward arcs.

3. Transformation of grid embedding into MIN-COST-FLOW. Let G be a 4-planar graph, and P a planar representation for G. In this section, we show that P can be extended to an orthogonal representation H with the minimum number of bends by computing a minimum cost flow in a network obtained from P through a straightforward transformation. A grid embedding for G can be constructed from the orthogonal representation H using the compaction algorithm described in the previous section.

The network $N(P) = (U, A, s, t, b, c)$ associated with the graph $G = (V, E)$ and the planar representation $P = \{P(f): f \in F\}$ is defined as follows:

Let $\hat{V}$ denote the subset of vertices of G having degree less than or equal to 3. Nodes of $N(P)$ are vertices in $\hat{V}$, faces of P, plus the source and the sink:

$$U = \hat{V} \cup F \cup \{s, t\}.$$

Arcs of $N(P)$ are of various types

$$A = A_V \cup A_F \cup A_S \cup A_T \cup \{(t, s)\}$$

where

(1) A_V contains arcs of the type (v, f), where $v \in \hat{V}$ and $f \in F$, such that v is one endpoint of some edge in $P(f)$. Arcs in A_V have infinite capacity and zero cost. We define

$$E(v, f) = \{e \in P(f): \text{both } e \text{ and the next edge in list } P(f) \text{ are incident to vertex } v\}.$$

(2) A_F contains two arcs, (f, g) and (g, f), for each pair $\{f, g\}$ of distinct faces such that the set

$$E(f, g) = \{e \in E: e \text{ appears in both } P(f) \text{ and } P(g)\}$$

is not empty. Moreover, A_F contains a self-loop (f, f) for each face f that has a bridge. In this case, we define

$$E(f, f) = \{e \in E: e \text{ is a bridge of face } f\}.$$

Arcs in A_F have infinite capacity and unit cost.

(3) A_S contains

(a) Arcs of the type (s, f), such that f is an internal face whose list $P(f)$ has length $|P(f)| \leqq 3$;

(b) Arcs of the type (s, v), such that $v \in \hat{V}$.

Arcs in A_S have zero cost and capacity:

$$b(s, f) = 4 - |P(f)|, \qquad b(s, v) = 4 - \deg(v).$$

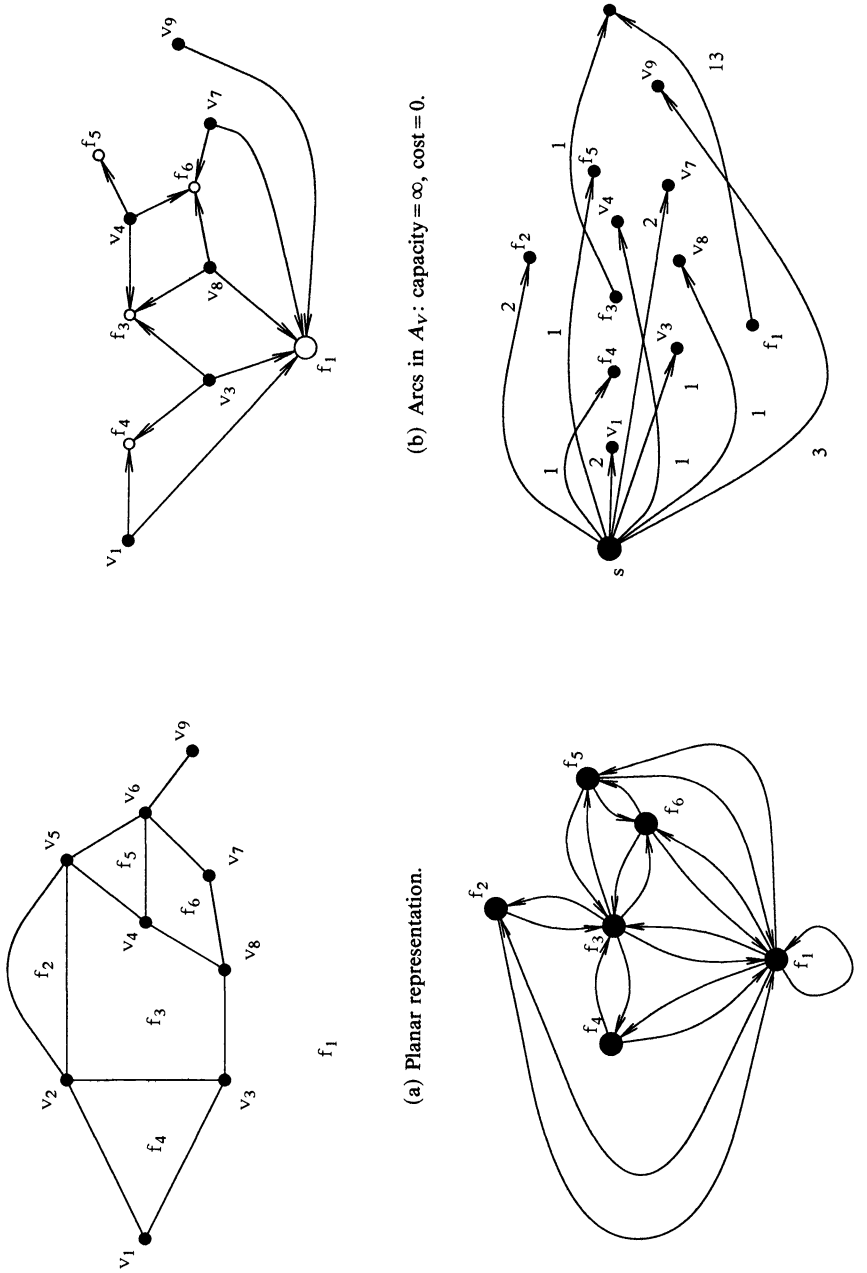
(4) A_T contains arcs of the type (f, t), such that f is either the external face, or an internal face whose list $P(f)$ has length $|P(f)| \geqq 5$. Arcs in A_T have zero cost and capacity given by

$$b(f, t) = \begin{cases} |P(f)| + 4 & \text{if } f \text{ is the external face,} \\ |P(f)| - 4 & \text{otherwise.} \end{cases}$$

See in Fig. 6 an example of this transformation.

The flow value in $N(P)$ is set equal to

$$z(P) = \sum_u b(s, u) = \sum_w b(w, t)$$



(a) Planar representation.

(b) Arcs in A_V: capacity $= \infty$, cost $= 0$.

(c) Arcs in A_F: capacity $= \infty$, cost $= 1$.

(d) Arcs in A_S and A_T: capacity is shown, cost $= 0$.

FIG 6. *Example of network* $N(P)$.

where the above equality follows from Euler's formula

$$\sum_{w\in U} b(w,t) - \sum_{u\in U} b(s,u) = \sum_{f\in F} (|P(f)|-4)+8+\sum_{v\in V} (\deg(v)-4)$$

$$= 2|E|-4|F|+8+2|E|-4|V|$$

$$= 4(|E|-|F|-|V|+2) = 0.$$

Because of the flow value $z(P)$, the flow in arcs of A_S and A_T must be equal to the capacity. Furthermore, since every node in $\hat{V}$ has exactly one ingoing arc with capacity at most 3, the flow in arcs of A_V cannot be greater than 3.

The transformation into a minimum cost flow problem is based on the following intuitive interpretation:

(1) Each unit of flow in network $N(P)$ represents an angle of 90 degrees: for arcs in A_V, flow $x(v,f)$ represents the sum of the measures of angles formed inside face f by the edges incident to v, which is given by $(x(v,f)+1)90$; for arcs in A_F, flow $x(f,g)$ represents the number of bends that appear along the edges separating face f from face g and form an angle of 90 degrees inside face f.

(2) The conservation rule at vertex-nodes means that the sum of angles around each vertex must be equal to 360 degrees (Property 4). The conservation rule at face-nodes means that each face must be a rectilinear polygon (Property 3).

(3) The cost of the flow is equal to the number of bends.

LEMMA 2. *For each orthogonal graph G having planar representation P, there exists an integer flow x in $N(P)$ whose value is $z(P)$, and whose cost is equal to the number of bends in G.*

Proof. Let H be the orthogonal representation of G. The flow x in $N(P)$ is constructed from H as follows.

For each (v,f) in A_V, flow $x(v,f)$ is obtained from the a-fields of the elements of $H(f)$ associated with arc (v,f):

$$x(v,f) = \sum_{r\in R(v,f)} (a[r]/90-1)$$

where

$$R(v,f) = \{r\in H(f)\text{: both } e[r] \text{ and the edge } e[r'] \text{ associated with the element } r' \text{ that follows } r \text{ in the list } H(f) \text{ are incident to vertex } v\}.$$

For each (f,g) in A_F, flow $x(f,g)$ is obtained from the s-fields of the elements of $H(f)$ associated with arc (f,g)

$$x(f,g) = \sum_{r\in R(f,g)} \text{ZEROS}(s[r])$$

where

$$R(f,g) = \{r\in H(f)\text{: } e[r]\in E(f,g)\}.$$

Flow $x(f,g)$ represents the angles of 90 degrees inside face f appearing along edges separating f from g. If $f=g$, then $x(f,f)$ is equal to the global number of bends along the bridges of f.

Flow in arcs of A_S and A_T is equal to the capacity, as imposed by the value of the flow. Figure 7 shows the above construction on the network of Fig. 6.

To prove that the function x so obtained is a legal flow in $N(P)$ we have to show that (1) capacity bounds are not violated; (2) flow is conserved at each node.

The first condition is easily verified from the above construction. The second condition is a consequence of the properties of the orthogonal representation H, as shown below.

For nodes in $\hat{V}$, we have from Property 4

$$\sum_f x(v,f) - x(s,v) = \sum_f \sum_{r \in R(v,f)} (a[r]/90 - 1) - (4 - \deg(v))$$

$$= 360/90 - \deg(v) - 4 + \deg(v) = 0.$$

For nodes in F, we have from Properties 2 and 3

$$\sum_u x(f,u) - \sum_w x(w,f)$$

$$= \sum_g x(f,g) - \sum_g x(g,f) - \sum_v x(v,f) + |P(f)| \pm 4$$

$$= \sum_{r \in H(f)} (\text{ZEROS}(s[r]) - \text{ONES}(s[r]) - a[r]/90 + 1) + |P(f)| \pm 4 = 0.$$

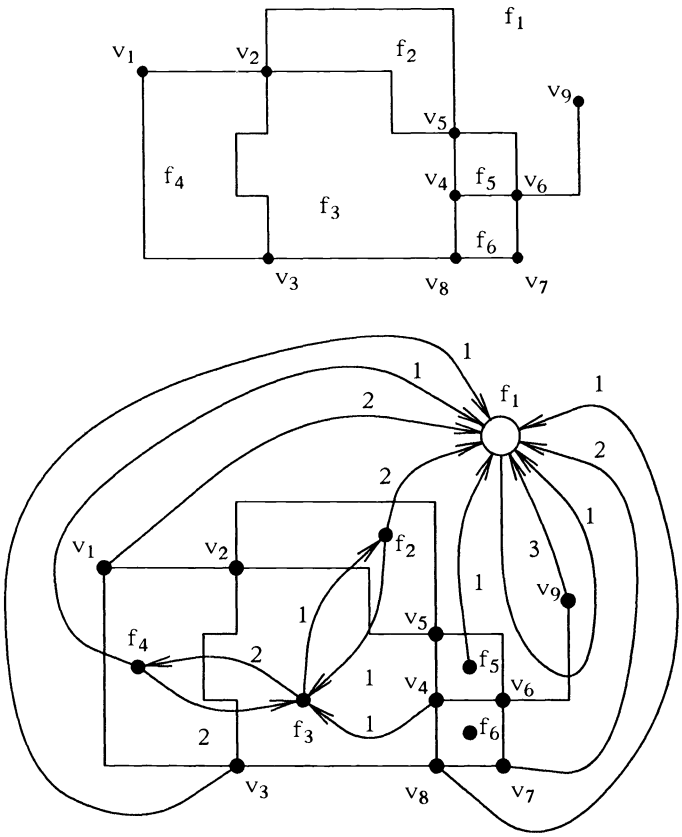


FIG. 7. *Example of construction of flow x in network N(P). (Only arcs in A_V and A_F with nonzero flow are shown.)*

Finally, for nodes s and t, the conservation of flow derives immediately from the definition of the flow value $z(P)$.

To complete the proof observe that, from (2.1) and Property 2, the cost of flow x is given by

$$\text{COST}(x) = \sum_{a \in A} c(a)x(a) = \sum_{(f,g) \in A_F} x(f,g) = \frac{1}{2} \sum_{f} \sum_{r \in H(f)} |s[r]|$$

$$= B(H) = (\text{no. of bends in } G). \qquad \text{Q.E.D.}$$

LEMMA 3. *For each integer flow x in $N(P)$ of value $z(P)$, there exists an orthogonal graph G with planar representation P whose number of bends is equal to the cost of flow x. Furthermore, the orthogonal representation of G can be computed from flow x.*

Proof. We construct the orthogonal representation of G computing the a- and s-fields from the values taken by x in arcs of A_V and A_F, respectively.

For each arc (v,f) in A_V, let $r_1, \cdots, r_n$ be the elements of $R(v,f)$ (see the definition of $R(v,f)$ in the proof of Lemma 2). Set

$$a[r_1] = (x(v,f)+1)90,$$

$$a[r_i] = 90 \quad \text{for } 2 \leqq i \leqq n.$$

For each pair of arcs (f,g) and (g,f) in A_F, let $r'_1, \cdots, r'_n$ be the elements of $R(f,g)$, and $r''_1, \cdots, r''_n$ be the corresponding elements of $R(g,f)$ (see the definition of $R(f,g)$ in the proof of Lemma 2). If $f = g$, then r'_i and r''_i are the pairs of elements in $H(f)$ corresponding to the two sides of the same bridge. Set

$$s[r'_1] = \begin{cases} \mathbf{0}^{x(f,g)} & \text{if } f = g, \\ \mathbf{0}^{x(f,g)}\mathbf{1}^{x(g,f)} & \text{otherwise,} \end{cases} \tag{3.1a}$$

$$s[r''_1] = \begin{cases} \mathbf{1}^{x(g,f)} & \text{if } f = g, \\ \mathbf{0}^{x(g,f)}\mathbf{1}^{x(f,g)} & \text{otherwise,} \end{cases} \tag{3.1b}$$

$$s[r'_i] = s[r''_i] = \varepsilon \quad \text{for } 2 \leqq i \leqq n. \tag{3.1c}$$

Property 1 is automatically satisfied, since we have built H from P. Property 2 easily follows from (3.1). Properties 3 and 4 follow from the conservation of flow in face-nodes and vertex-nodes, respectively. Finally, from (2.1) and (3.1) we have

$$B(H) = \frac{1}{2} \sum_{f} \sum_{r \in H(f)} |s[r]| = \frac{1}{2} \sum_{(f,g) \in A_F} (x(f,g) + x(g,f))$$

$$= \sum_{a \in A} c(a)x(a) = \text{COST}(x). \qquad \text{Q.E.D.}$$

LEMMA 4. *There exists a flow of minimum cost in $N(P)$ that takes only integer values.*

Proof. Since the capacities in $N(P)$ are integers, the lemma follows from well-known results in network flow theory [5]. Q.E.D.

By putting together the previous lemmas, we obtain the following basic result.

THEOREM 2. *Let P be a planar representation. The minimum number of bends for any orthogonal graph with planar representation P is equal to the minimum cost of a feasible flow in $N(P)$ of value $z(P)$. Furthermore, the orthogonal representation H of any optimal orthogonal graph can be computed from some optimal flow in $N(P)$.*

The transformation into min-cost-flow provides a nice characterization of optimal orthogonal graphs.

LEMMA 5. *A flow x for network N is of minimum cost among all flows with the same value if and only if N has no flow-augmenting cycles (with respect to x) of negative cost.*

Proof. See [5].

THEOREM 3. *An orthogonal graph G has the minimum number of bends among all orthogonal graphs with the same planar representation if and only if there is not a directed closed simple curve J such that*

(1) *J intersects each edge of G at most twice;*

(2) *J enters vertices of G only from angles greater than or equal to 180 degrees;*

(3) *More than half of the edges crossed by J have a bend with the angle of 270 degrees on the side from which J enters.*

Furthermore, if such a curve J exists, then an orthogonal graph G' can be found such that

(1) *G' has the same planar representation as G;*

(2) *G' has fewer bends than G;*

(3) *The orthogonal representation of G' differs from the one of G only in edges and vertices traversed by J.*

Proof. We show that a curve J satisfying the above properties exists for G if and only if network $N(P)$ has a negative cost flow-augmenting cycle with respect to the flow x corresponding to the orthogonal representation H of G (see Lemma 2). The first claimed result then follows from Lemma 5.

Suppose that $N(P)$ has a negative cost flow-augmenting cycle $C=(a_1, \cdots, a_m)$. C traverses neither the source s nor the sink t, since arcs incident to these nodes are saturated (see the definition of network $N(P)$). Construct from C a curve J, composed of stretches $J_1, \cdots, J_m$, as follows:

(1) Mark a point p_f in each face f of G;

(2) For each arc a_i of C, draw stretch J_i between the endpoints of a_i, i.e.

(i) if $a_i=(v,f)\in A_V$ is forward, then draw J_i from v to p_f inside face f;

(ii) If $a_i=(v,f)\in A_V$ is backward, then draw J_i from p_f to v inside face f;

(iii) If $a_i=(f,g)$ belongs to A_F, then draw J_i either from p_f to p_g, or from p_g to p_f, according to whether a_i is forward or backward. In the latter case, make J_i traverse an edge between f and g in correspondence to a bend, entering from the angle of 270 degrees.

Condition 1 is satisfied since J traverses edges of G only in correspondence to arcs of C in A_F, and each edge of G is associated with at most two arcs in A_F. Condition 2 of Theorem 3 is satisfied since, if J enters vertex v from face f, then (v,f) is a backward arc in C and has thus nonzero flow, corresponding to an angle of at least 180 degrees (see the proof of Lemma 2). Condition 3 follows from the fact that C has negative cost, and hence goes more times through backward arcs in A_F than it does through forward ones.

To prove the converse, suppose that a curve J satisfying conditions 1, 2, and 3 exists. First, decompose J into stretches $J_1, \cdots, J_m$, where each J_i has one endpoint in a face, and the other endpoint either in a vertex or in a face; in the former case, J_i must not cross any edge, while, in the latter case, it must cross exactly one edge. Then, construct a flow augmenting circuit $C=(a_1, \cdots, a_m)$ as follows:

(1) If J_i goes from face f to vertex $v \in \hat{V}$, then set $a_i = (v, f)$; a_i is a backward arc of C since, from condition 2, arc (v, f) has flow $x(v, f) \geqq 1$;

(2) If J_i goes from vertex $v \in \hat{V}$ to face f, then set $a_i = (v, f)$; a_i is a forward arc;

(3) If J_i goes from face f to face g, then set $a_i = (g, f)$ backward if edge (f, g) has a bend with the angle of 270 degrees inside face f; otherwise, set $a_i = (f, g)$ forward.

Conditions 1 and 2 assure that the circuit so constructed is a flow-augmenting cycle. Condition 3, together with the fact that arcs of type (v, f) have zero cost, assures that C has negative cost.

Finally, by augmenting the flow along cycle C (i.e. augmenting by a unit the flow in forward arcs, and decreasing by a unit the flow in backward arcs), we obtain a new flow x' with the same value as x, but with less cost. Lemma 3 implies then the existence of an orthogonal graph G' that satisfies the properties stated in the second part of the theorem. Q.E.D.

Figure 8 shows examples of application of Theorem 3, that can be used to prove the following intuitive facts.

COROLLARY 1. *Every bridge of an optimal orthogonal graph is a straight line segment.*

COROLLARY 2. *No edge of an optimal orthogonal graph has both a 90-degree and a 270-degree angle on the same side.*

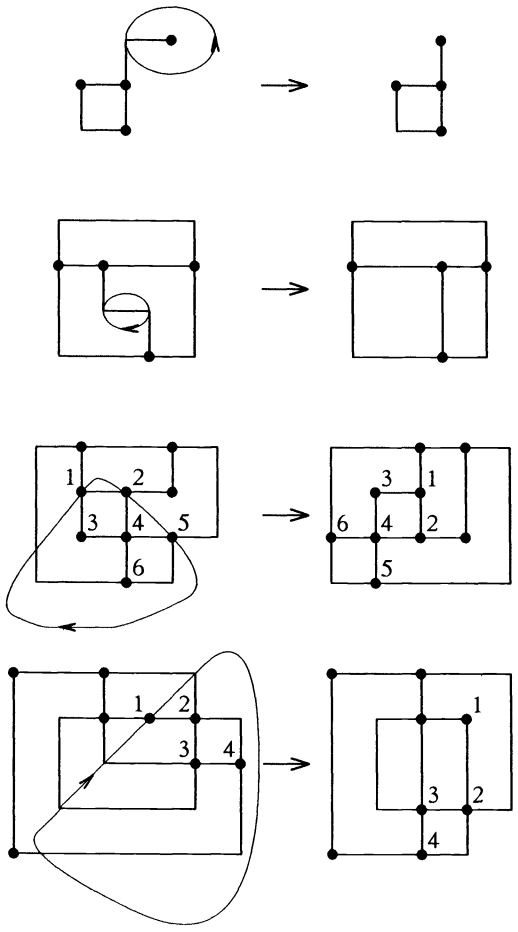


FIG. 8. *Examples of application of Theorem* 3.

We have shown that finding an orthogonal representation with minimum number of bends is equivalent to computing a minimum cost flow in network $N(P)$. The augmentation algorithm can be conveniently used for finding a minimum cost flow in network $N(P)$, since, as shown in the following, both the number of arcs and the value of flow in $N(P)$ grow linearly with the number of vertices of the graph represented by P.

LEMMA 6. *Let P be a planar representation for the 4-planar graph $G=(V, E)$. The following inequalities hold for the network $N(P)=(U, A, s, t, b, c)$ and the flow value $z(P)$:*

(1) $|U| \leqq \frac{3}{2}|V|+4$,

(2) $|A| \leqq \frac{15}{2}|V|+3$,

(3) $z(P) \leqq 3|V|+2$.

Furthermore any one of them is satisfied as an equality for infinitely many planar representations.

Proof. We have from construction that

$$U = \hat{V} \cup F \cup \{s, t\}.$$

Let V_k be the subset of vertices with degree k. We have then

$$|U| = |E| - |V_4| + 4.$$

Since

$$2|E| = \sum_k k|V_k|, \tag{4.1}$$

we obtain

$$|U| = \tfrac{1}{2}(|V_1| + |V_2| + 3|V_3| + 2|V_4|) + 4 \leqq \tfrac{3}{2}|V| + 4$$

where the equality is attained when all vertices of G have degree 3. Figures 9(a) and 9(b) show examples of planar representations where this actually happens.

For the second inequality, we have

$$\begin{aligned} |A| &= |A_F| + |A_V| + |A_S| + |A_T| + 1 \\ &\leqq 2|E| + 2|E| - 4|V_4| + |V| - |V_4| + |F| - |F_4| + 1 \end{aligned}$$

where

$$F_k = \{f \in F : f \text{ is internal and } |P(f)| = k\}.$$

Applying Euler's formula and (4.1) we obtain

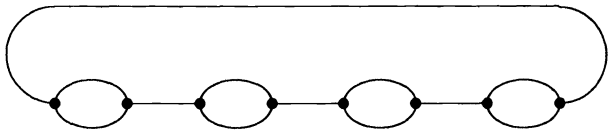
$$\begin{aligned} |A| &\leqq 5|E| - 5|V_4| - |F_4| + 3 \\ &\leqq \tfrac{5}{2}(|V_1| + 2|V_2| + 3|V_3| + 4|V_4|) - 5|V_4| - |F_4| + 3 \\ &\leqq \tfrac{15}{2}|V| + 3. \end{aligned}$$

See in Fig. 9(b) an example of planar representation that satisfies formula 2 with equality.

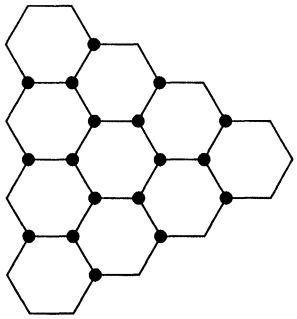
Finally, for the third inequality we have

$$z(P) = |P(q)| + 4 + \sum_{f \neq q} \max\{0, |P(f)| - 4\}$$

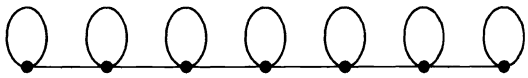
where q is the external face.



(a)



(b)



(c)

FIG. 9. *Planar representations that attain the upper bounds of Lemma 6.*

$$z(P) = |P(q)| + \sum_{k=1}^{2|E|} |F_k|(k-4) - \sum_{k=1}^{3} |F_k|(k-4) + 4$$

$$= 2|E| - 4(|F|-1) + \sum_{k=1}^{3} |F_k|(4-k) + 4.$$

We distinguish now two cases.

CASE 1. All vertices have degree 4.

At least one internal face has length greater than 1, since otherwise the graph would be a tree together with self-loops, and leaves would have odd degree. We have thus

$$z(P) \leqq 4|V| - 4|F| + 3(|F|-2) + 2 + 8 = 3|V| + 2.$$

CASE 2. Some vertex has degree less than four. Since in this case $|E| \leqq 2|V| - 1$, we obtain

$$z(P) \leqq 2(2|V|-1) - 4|F| + 3(|F|-1) + 8 = 3|V| + 2.$$

Planar representations like the one depicted in Fig. 9(c) attain the upper bound on $z(P)$. Q.E.D.

The following algorithm finds an orthogonal representation with the minimum number of bends.

ALGORITHM ORTHOGONALIZE.

Input. 4-planar graph G, planar representation P of G.

Output. Orthogonal representation H for P with the minimum number of bends.

Step 1. Construct network $N(P)$.

Step 2. Find a minimum cost flow x in $N(P)$, of value $z(P)$, by using the augmentation algorithm.

Step 3. Compute H from the optimal flow x, as stated in the proof of Lemma 3.

THEOREM 4. *Given a 4-planar graph G with n vertices, and a planar representation P for G, Algorithm* ORTHOGONALIZE *finds an orthogonal representation for P with the minimum number of bends in $O(n^2 \log n)$ time.*

Proof. Steps 1 and 3 of the algorithm are easily carried out in $O(|U|+|A|)$ time. Since in network $N(P)$ capacities are integer and costs are positive, Step 2 can be performed in $O(z(P)(|U|+|A|)\log|U|)$ time. The claimed result follows then from Lemma 6. Q.E.D.

If the graph G is not biconnected, it is convenient to apply Algorithm ORTHOGONALIZE separately to each biconnected component G_i of G. In order to guarantee a correct reconstruction of the global orthogonal representation, the capacity of each arc (v,f) in $N(P_i)$ associated with a cut-point v of G_i must be suitably decreased. The details are simple and left to the reader.

The complete algorithm for embedding a graph in the grid with the minimum number of bends consists of applying first ORTHOGONALIZE to find an optimal shape for the embedding, and then FIND_LENGTHS to give it dimensions.

ALGORITHM GRID_EMBED.

Input. 4-planar graph G, planar representation P for G.

Output. Grid embedding Q of G that is region preserving with respect to P, and has the minimum number of bends.

Step 1. Apply Algorithm ORTHOGONALIZE to find an optimal orthogonal representation H for P.

Step 2. Apply Algorithm FIND_LENGTHS to give lengths to segments in H, thus obtaining an optimal grid embedding Q of G.

Before stating the main theorem of the paper we need the following result of Storer.

LEMMA 7. *Every 4-planar representation with n vertices admits a grid embedding with no more than $3n+2$ bends.*

Proof. See [10].

THEOREM 5. *Given a 4-planar graph G with n vertices, and a planar representation P for G, Algorithm* GRID_EMBED *finds a region preserving grid embedding of G with the minimum number of bends in $O(n^2 \log n)$ time.*

Proof. From Theorem 4, Step 1 of Algorithm GRID_EMBED has $O(n^2 \log n)$ time complexity. From Theorem 1, Step 2 has $O(n+b)$ complexity, where b is the number of bends in H. Since H is optimal, we have from Lemma 7 that $b = O(n)$. Therefore, the global time complexity of the embedding algorithm is $O(n^2 \log n)$. Q.E.D.

4. A constrained version of the bend minimization problem. The layout algorithm described in § 3 can be modified in order to take into account several types of constraints on the shape of the grid embedding, as well as piecewise linear convex cost functions for bends.

PROBLEM. *Shape constrained grid embedding with minimum bend cost.* Given:

(1) A 4-planar graph $G=(V,E)$;

(2) A planar representation P for G;

(3) A subgraph G' of G;

(4) An orthogonal representation H' for G';
(5) An upper bound $U(e)$ on the number of bends, for each edge $e \in E'' = E - E'$;
(6) A *bend cost* $d_e(k)$ for each edge $e \in E$, such that

$$d_e(0) = 0,$$
$$d_e(k+1) \geqq d_e(k) \quad \text{for each } k \geqq 0,$$
$$d_e(k+1) - d_e(k) \geqq d_e(k) - d_e(k-1) \quad \text{for each } k \geqq 1,$$

find a grid embedding Q of G subject to the following constraints:
(1) Q is region preserving with respect to P;
(2) The orthogonal representation of the embedded subgraph $Q(G')$ is equal to H';
(3) The number of bends $B(e)$ on each embedded edge $Q(e)$ verifies $B(e) \leqq U(e)$;
that minimizes the global bend cost of Q, given by

$$\text{BEND_COST}(Q) = \sum_{e \in E} d_e(B(e)).$$

In order to deal with the new constraints, network $N(P)$ must be modified as follows.

Define $E''(f, g) = E(f, g) \cap E''$ and assign to each arc (f, g) in A_F the capacity

$$b(f, g) = \sum_{e \in E''(f,g)} U(e).$$

This guarantees that constraint (3) is satisfied in any optimal embedding (where $x(f, g)x(g, f) = 0$).

In order to satisfy constraint (2), each arc (f, g) of A_F such that the set $E(f, g)$ is contained in E' must have a flow of fixed value $x'(f, g)$, which can be computed from H' using the transformation described in the proof of Lemma 2. To obtain this, we replace arc (f, g) with two arcs, (s, g) and (f, t), each having capacity equal to $x'(f, g)$. Moreover, for each arc (f, g) of A_F such that the set $E(f, g)$ has a nonempty intersection with E' but is not contained in it, we add the arcs (s, g) and (f, t), but we do not remove the arc (f, g). A similar transformation is performed on the arcs of A_V whose flow is imposed by H'.

As a result of this operation, parallel arcs might have been introduced in the network; in this case, we replace every bundle of parallel arcs $a_1, \cdots, a_m$ with a unique arc a, summing up their capacities, i.e. $b(a) = b(a_1) + \cdots + b(a_m)$. Further simplifications can now be performed, as explained below.

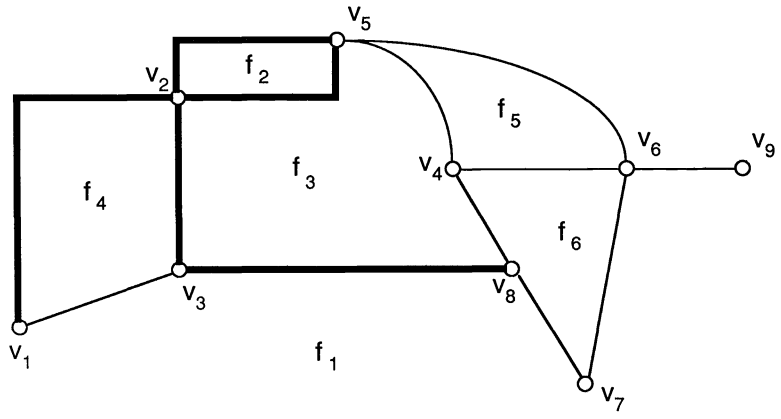
(1) Remove nodes linked by arcs only to s and t;
(2) For each node u that is linked to both s and t
(i) if $b(s, u) = b(u, t)$, then remove both arcs (s, u) and (u, t);
(ii) if $b(s, u) > b(u, t)$, then remove arc (u, t) and reduce by $b(u, t)$ the capacity of arc (s, u);
(iii) if $b(s, u) < b(u, t)$, then remove arc (s, u) and reduce by $b(s, u)$ the capacity of arc (u, t).

The flow value $z^*(P)$ in the resulting modified network $N^*(P)$ is set equal to

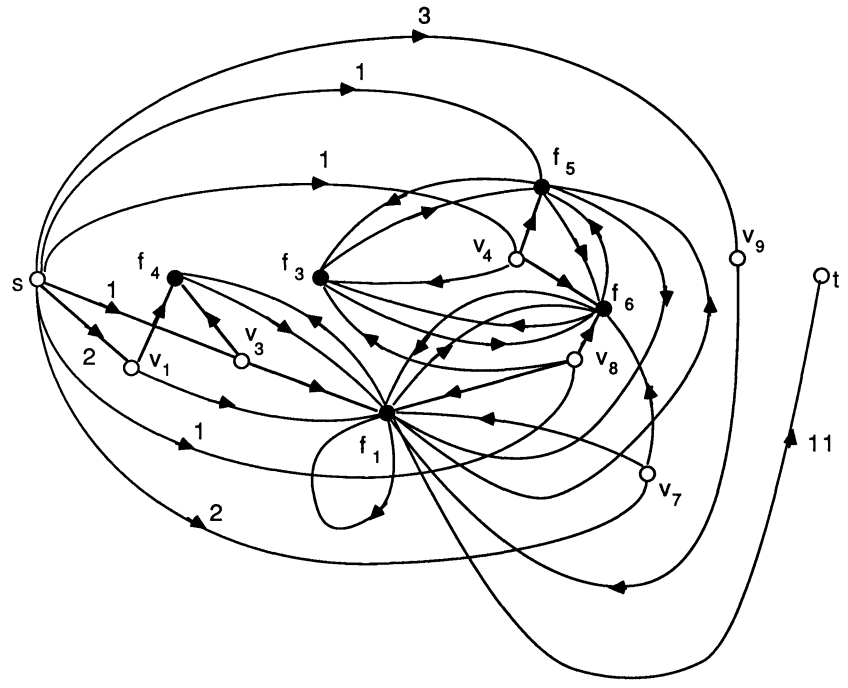
$$z^*(P) = \sum_u b(s, u) = \sum_w b(w, t).$$

Figure 10 shows an example of this construction.

A piecewise linear nondecreasing cost convex cost function $c_{(f,g)}(x)$ is associated with each arc (f, g) between face nodes, while the remaining arcs have zero cost. The cost function $c_{(f,g)}(x)$ is obtained from the bend costs of edges in $E''(f, g)$ as follows: Let $\Delta(f, g) = \delta_1, \cdots, \delta_n, \cdots$ be the list of values $d_e(k) - d_e(k-1)$ in nondecreasing



(a) Planar representation. Subgraph G' and its orthogonal representation are shown in bold lines.



(b) Network N^*: capacities are shown only for arcs incident to s or t.

FIG. 10. *Example of construction of network* $N^*(P)$.

order, where $e \in E''(f, g)$ and $1 \leqq k \leqq U(e)$; values are repeated in the list if they are generated by distinct edges e, or by distinct arguments k. For each arc (f, g) in A_F, the function $c_{(f,g)}(x)$ is given by

$$c_{(f,g)}(x) = \begin{cases} 0 & \text{if } x = 0, \\ \sum_{i=1}^{k} \delta_i & \text{if } x = k,\ k \in N, \\ c_{(f,g)}(k) + (x-k)\,\delta_{k+1} & \text{if } k < x < k+1,\ k \in N. \end{cases}$$

The cost of a feasible flow x in network $N^*(P)$ is given by

$$\text{COST}(x) = \sum_a c_a(x(a)).$$

From the above discussion and the results in § 3, we obtain the following theorem.

THEOREM 6. *The minimum bend cost for any grid embedding Q that satisfies constraints* (1), (2), *and* (3) *is equal to the minimum cost of a feasible flow in $N^*(P)$ of value $z^*(P)$. Furthermore, the orthogonal representation H of any optimal grid embedding can be computed from some optimal flow in $N^*(P)$.*

Notice that the modified network $N^*(P)$ might not have any flow of value $z^*(P)$; this happens if and only if the constrained problem does not have any solution.

The out-of-kilter algorithm can handle piecewise linear convex cost functions without increase in the order of complexity [2], [5], [8]. This allows us to solve the constrained embedding problem by modifying Algorithm ORTHOGONALIZE of § 3 as follows.

ALGORITHM ORTHOGONALIZE2.

Input. See description of the problem *Shape constrained grid embedding with minimum bend cost.*

Output. Orthogonal representation H of a grid embedding Q that satisfies the constraints (1), (2), and (3), and has minimum bend cost.

Step 1. Construct network $N^*(P)$.

Step 2. If $N^*(P)$ does not have a feasible flow of value $z^*(P)$, report that the problem has no solution and stop. Otherwise, find a minimum cost flow $x^*(P)$ for $N^*(P)$ of value $z^*(P)$ by means of the out-of-kilter algorithm.

Step 3. Compute H from the optimal flow x^* and the orthogonal representation H'. Bends are assigned to edges of $E''(f, g)$ according to the sequence $\Delta(f, g)$, i.e. the number of bends in edge $e \in E''(f, g)$ is set equal to the number of occurrences of the differences δ_i computed from d_e in the sequence $\delta_1, \cdots, \delta_{x(f,g)+x(g,f)}$.

5. Extension to k-gonal graphs. In this section, we show an extension of the previous results to a class of graphs that generalizes orthogonal graphs.

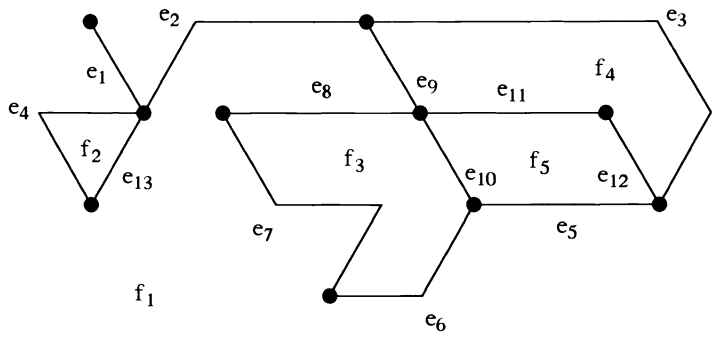
We call a *k-gonal graph*, where $k \geqq 2$, a plane graph whose edges are sequences of segments with slope multiple of $180/k$ degrees, with respect to a reference axis. For $k = 2$, we obtain orthogonal graphs. The *k-gonal representation* decribes the shape of a k-gonal graph G. It is a set of circularly ordered lists $K(f)$, one for each face f of the k-gonal graph, whose elements have the format (e, s, a), where

(1) e is an edge of G;

(2) s is a string from an alphabet of $2(k-1)$ symbols, which is useful to identify with the numbers $\pm 1, \cdots, \pm(k-1)$;

(3) a is an integer in the set $\{m \cdot 180/k\colon m = 1, \cdots, 2k\}$.

The meaning of the fields in an element r of $K(f)$ is the same as in the orthogonal representation, with the only difference that a symbol p in the string $s[r]$ stands for an angle of $180(1-p/k)$ degrees. See the example in Fig. 11.



$K(f_1) = ((e_1, \varepsilon, 60), (e_4, +2, 300), (e_{13}, \varepsilon, 180), (e_2, +1, 120),$
$(e_9, \varepsilon, 60), (e_8, \varepsilon, 300), (e_7, -1+2, 300), (e_6, -1, 120),$
$(e_5, \varepsilon, 240), (e_3, -1-1, 180), (e_2, -1, 60), (e_1, \varepsilon, 360))$
$K(f_2) = ((e_4, +2, 60), (e_{13}, \varepsilon, 60))$
$K(f_3) = ((e_6, +1, 60), (e_7, -2+1, 60), (e_8, \varepsilon, 120), (e_{10}, \varepsilon, 120))$
$K(f_4) = ((e_9, \varepsilon, 60), (e_3, +1+1, 60), (e_{12}, \varepsilon, 240), (e_{11}, \varepsilon, 120))$
$K(f_5) = ((e_5, \varepsilon, 120), (e_{10}, \varepsilon, 60), (e_{11}, \varepsilon, 120), (e_{12}, \varepsilon, 60))$

FIG. 11. *Examples of 3-gonal representation.*

To each bend in a k-gonal graph a cost is given that increases with the deviation from the straight line, i.e., if a bend forms an angle α, its cost is $|\alpha - 180| \cdot k/180$. The *bend cost* of a k-gonal graph G is the sum of the costs of its bends and can be computed from its k-gonal representation K:

$$\text{BEND_COST}(G) = \frac{1}{2} \sum_{f} \sum_{r \in K(f)} \text{COST}(s[r]),$$

where, given $s[r] = p_1 \cdots p_n$, we define

$$\text{COST}(s[r]) = \sum_{i=1}^{n} |p_i|.$$

A necessary and sufficient condition for a set of circularly ordered lists $K(f)$ to be the k-gonal representation of some k-gonal graph is the satisfaction of the following four properties.

PROPERTY 1′. There is some $2k$-planar graph whose planar representation is given by the e-fields of the $K(f)$ lists.

PROPERTY 2′. Let r' and r'' be a pair of elements in K associated with the same edge. Then string $s[r'']$ is obtained from $s[r']$ by reversing $s[r']$ and changing sign to all its symbols.

PROPERTY 3′. Define, for each element r

$$\text{ROTATION}(r) = \sum_{i=1}^{n} p_i + k(1 - a[r]/180)$$

where we denote $s[r] = p_1 \cdots p_n$. Then the following equality holds:

$$\sum_{r \in K(f)} \text{ROTATION}(r) = \begin{cases} -2k & \text{if } f \text{ is the exernal face,} \\ +2k & \text{otherwise.} \end{cases}$$

PROPERTY 4′. For each vertex v of the plane graph described by K, the sum of angles between pairs of consecutive edges around v, as given by the corresponding elements in the $K(f)$ lists, is equal to 360 degrees.

Given a $2k$-planar representation P, the problem of finding a k-gonal representation K of minimum cost can be solved by computing a minimum cost flow in the network $N_k(P)$, which is defined in the same way as $N(P)$ (see § 3), but with the following changes:

(1) $\hat{V}$ denotes the set of vertices with degree less than or equal to $2k-1$.

(2) A_S contains an arc (s, v) for each vertex-node $v \in \hat{V}$. The capacity of these arcs is $b(s, v) = 2k - \deg(v)$.

(3) A_S contains an arc (s, f) for each face f whose list $P(f)$ has length $|P(f)| < 2k/(k-1)$. The capacity of these arcs is $b(s, f) = 2k - |P(f)|(k-1)$.

(4) A_T contains an arc (f, t) for each face f such that

(i) f has length $|P(f)| > 2k/(k-1)$, or

(ii) f is the external face.

In case (i) we have $b(f, t) = |P(f)|(k-1) - 2k$, while in case (ii) we have $b(f, t) = |P(f)|(k-1) + 2k$.

(5) The flow value in $N_k(P)$ is set equal to

$$z_k(P) = \sum_u b(s, u) = \sum_w b(w, t).$$

See the example in Fig. 12.

The following results are extensions of Lemmas 2-4 and 6, and of Theorems 2-4.

LEMMA 8. *For each k-gonal graph G having planar representation P, there exists an integer flow x in $N_k(P)$ whose value is $z_k(P)$, and whose cost is equal to the bend cost of G.*

LEMMA 9. *For each integer flow x in $N_k(P)$ of value $z_k(P)$, there exists a k-gonal graph G with planar representation P whose bend cost is equal to the cost of flow x. Furthermore, the k-gonal representation of G can be computed from flow x.*

LEMMA 10. *There exists a flow of minimum cost in $N_k(P)$ that takes only integer values.*

LEMMA 11. *Let P be a planar representation for the $2k$-planar graph $G = (V, E)$. We have then, for the network $N_k(P) = (U, A, s, t, b, c)$ and the flow value $z_k(P)$,*

(1) $|U| \leqq (2k-1)|V|/2 + 4$,

(2) $|A| = O(k|V|)$,

(3) $z_k(P) = O(k|V|)$.

Furthermore, these bounds are attained in infinitely many planar representations.

THEOREM 7. *Let P be a planar representation. The minimum bend cost for a k-gonal graph with planar representation P is equal to the minimum cost of a feasible flow in $N_k(P)$ of value $z_k(P)$. Furthermore, the k-gonal representation K of an any optimal k-gonal graph can be computed from some optimal flow in $N_k(P)$.*

THEOREM 8. *A k-gonal graph G has the minimum bend cost among all k-gonal graphs with the same planar representation if and only if there is not a directed closed simple curve J such that*

(1) *J intersects each edge of G at most twice;*

(2) *J enters vertices of G only from angles greater than or equal to $360/k$ degrees;*

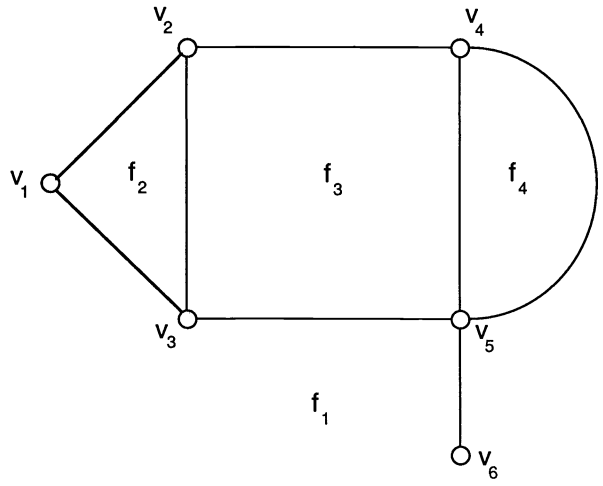
(3) *More than half of the edges crossed by J have a bend that forms an angle greater than 180 degrees on the side on which J enters.*

Furthermore, if such a curve J exists, then a k-gonal graph G′ can be found such that

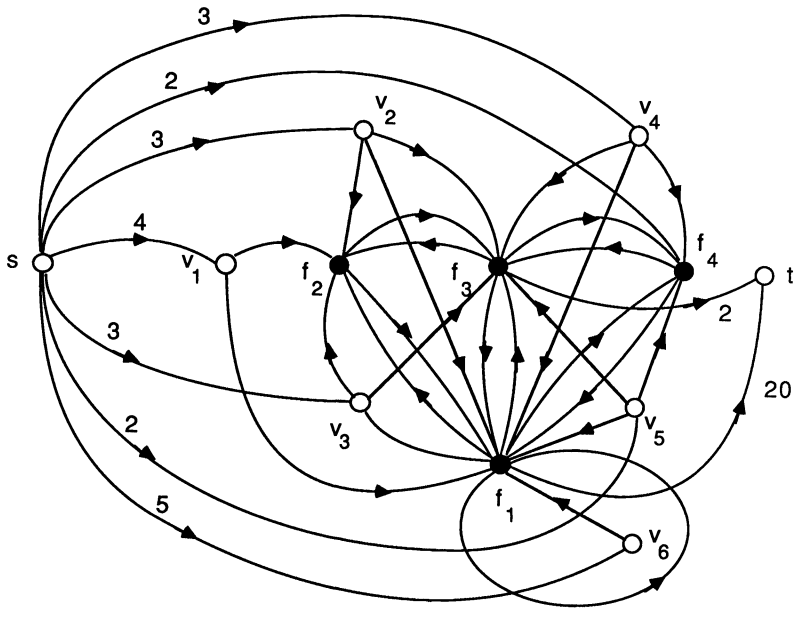
(1) *G′ has the same planar representation as G;*

(2) *G′ has bend cost less than the one of G;*

(3) *The k-gonal representation of G′ differs from the one of G only in edges and vertices traversed by J.*



(a) Planar representation P.



(b) Network $N_3(P)$: capacities are shown only for arcs in A_S and A_T.

FIG. 12. *Example of network* $N_3(P)$.

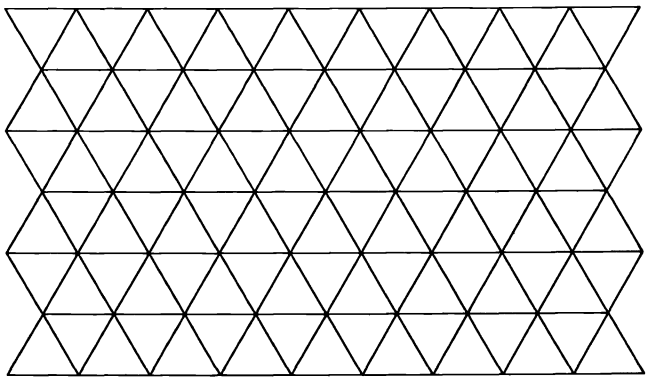


FIG. 13. *Triangled grid for 3-gonal graphs.*

THEOREM 9. *Given a $2k$-planar graph G with n vertices, and a planar representation P for G, an orthogonal representation for P with minimum bend cost can be computed in $O(k^2n^2 \log (kn))$ time.*

There is not a natural extension of the concept of grid to the general k-gonal case. For $k=3$, the triangled grid shown in Fig. 13 seems to work. However, for $k>3$, there is not an infinite plane graph with regular structure whose vertices all have degree $2k$ and whose edges are segments. This fact prevents us from giving an embedding algorithm for k-gonal representations similar to the one described in § 2.

Constrained versions of the bend cost minimization problem can also be considered, with results parallel to the ones presented in § 4.

Acknowledgment. I am very grateful to Carlo Batini for his encouragement and support during this research work.

REFERENCES

[1] C. BATINI, M. TALAMO AND R. TAMASSIA, *Computer aided layout of entity-relationship diagrams*, J. Systems Software, 4 (1984), pp. 163-173.

[2] D. R. FULKERSON, *An out-of-kilter method for minimal cost flow problems*, SIAM J. Appl. Math., 9 (1961), pp. 18-27.

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

[4] M. Y. HSUEH, *Symbolic layout and compaction of integrated circuits*, Ph.D. thesis, Univ. of California at Berkeley, 1979, pp. 26-28, 46-52.

[5] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976, Chapter 4.

[6] C. E. LEISERSON, *Area efficient* VLSI *computation*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1981; MIT Press, Cambridge, MA, 1983.

[7] R. LIGGETT AND W. MITCHELL, *Optimal space planning in practice*, Computer Aided Design, 13 (1981), pp. 277-288.

[8] G. J. MINTY, *Monotone networks*, Proc. Roy. Soc. London Ser. A, 257 (1960), pp. 194-212.

[9] A. PAOLUZZI, *Il Progetto di Componenti Edilizi Industrializzati ed il Controllo di Qualità attraverso l'Approccio Sistemistico*, Technical Report 14, Università di Roma, Istituto di Architetura Edilizia e Tecnica Urbanistica, 1978.

[10] J. A. STORER, *The node cost measure for embedding graphs in the planar grid*, Proc. 12th ACM Symposium on the Theory of Computing, 1980, pp. 201-210.

[11] R. TAMASSIA, C. BATINI AND M. TALAMO, *An algorithm for automatic layout of entity relationship diagrams*, in Entity-Relationship Approach to Software Engineering, Proc. 3rd Internat. Conf. on Entity-Relationship Approach, C. G. Davis, S. Jajodia, P. A. Ng and R. T. Yeh, eds., North-Holland, Amsterdam, New York, 1983, pp. 421-439.

[12] L. G. VALIANT, *Universality considerations in* VLSI *circuits*, IEEE Trans. Comput., C-30 (1981), pp. 135-140.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987

© 1987 Society for Industrial and Applied Mathematics

FAST DISTRIBUTED AGREEMENT*

SAM TOUEG†, KENNETH J. PERRY† AND T. K. SRIKANTH†

Abstract. We describe a Byzantine Agreement algorithm, with early stopping, for systems with arbitrary process failures. The algorithm presented is simpler and more efficient than those previously known. It was derived using a broadcast primitive that provides properties of message authentication and thus restricts the disruptive behavior of faulty processes. This primitive is a general tool for deriving fault-tolerant algorithms in the presence of arbitrary failures.

Key words. Byzantine Agreement, fault-tolerance, distributed computing

AMS(MOS) subject classifications. 68M10, 68M15

1. Introduction.

Fault-tolerance is one of the main advantages claimed by distributed systems. A common technique is to replicate processors, and reach agreement on inputs and results. For this reason, much attention has been paid recently to algorithms for reaching processor agreement in failure-prone distributed systems.

We consider a collection of n distributed, potentially faulty processes that communicate through a completely connected and reliable network. In this network, the receiver of a message can always identify the immediate sender of this message. The processes execute a synchronous algorithm in which a distinguished process, the *General*, broadcasts a value m, and all the processes attempt to reach agreement on the value broadcast. *Correct* processes faithfully follow the algorithm, while *faulty* processes may deviate from it. The algorithm solves the *Byzantine Agreement* (or *Byzantine Generals*) problem if it satisfies the properties:

validity: If the *General* is correct then all correct processes agree on m.

agreement: All correct processes agree on a common value.

This problem is recognized as a fundamental problem of fault-tolerant computing, and has been extensively studied under various types of failures [Peas80], [Lamp82a], [Lamp82b], [Dole82c], [Dole83], [Hadz83], [Perr86].

The most restrictive type of failure considered is *crash* faults, where processes fail by halting. Even under this restricted model of failure, it was shown that at least $t+1$ phases[1] of message exchange are needed to reach agreement by any algorithm that overcomes up to t faulty processes [Lamp82a].

This lower bound of $t+1$ on the number of phases required to reach Byzantine Agreement has been used to argue that solutions to this problem are too expensive to be practical. However, by relaxing certain requirements of the problem, agreement can be reached in fewer than t phases, if f, the *actual* number of failures that occur during execution of the algorithm, is less than t, the *maximum* number of failures that the algorithm can overcome.

* Received by the editors December 17, 1984; accepted for publication (in revised form) May 16, 1986. This work was supported in part by the National Science Foundation under grant MCS-8303135.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

[1] Informally, a phase consists of a synchronized message exchange where processes first broadcast messages (according to their state), then wait to receive messages sent by other processes in the same phase, and change their state accordingly.

Dolev et al. [Dole82b] identify two types of Byzantine Agreement algorithms, depending on whether the action resulting from the agreement is to be synchronized or not. These are referred to as *immediate* Byzantine Agreement and *eventual* Byzantine Agreement.

Byzantine Agreement is *immediate* if correct processes can also agree on the phase number at which they reach agreement. Thus, once immediate agreement is reached, processes not only agree on the action to be performed, but are also sure that all correct processes perform the action in the same phase. If this condition is relaxed, agreement is *eventual*: we are only guaranteed that if a correct process decides on some value, then every correct process eventually decides on the same value. This type of agreement is sufficient for many applications where simultaneity is not necessary, for example, when committing a transaction in database systems.

Eventual Byzantine Agreement can be reached in time proportional to f rather than t. Algorithms with this property are said to exhibit *early stopping* [Dole82a]. With these algorithms, one pays for the fault-tolerance actually needed, rather than always paying the worst-case time complexity.

Agreement algorithms with early stopping have previously been developed for systems with crash faults [Lamp82b], omission faults (where processes fail by halting or by omitting to send some of the required messages) [Hadz83], and *sr*-omission faults (where processes may also fail to receive messages) [Perr86]. Agreement algorithms with early stopping were also derived for systems with *arbitrary* failures [Dole82b], [Reis82]. In this model of failure, processes deviate arbitrarily from the algorithm, even maliciously colluding with one another in order to foil agreement (these are also known as *Byzantine* faults).

However, the algorithms proposed in [Dole82b], [Reis82] are generally regarded as complicated and difficult to understand. Furthermore, the algorithm in [Reis82] cannot tolerate more than $n/20$ simultaneous failures. The first algorithm in [Dole82b] reaches eventual Byzantine Agreement in $\min(2f+5, 2t+3)$ phases and requires $O(nt^2f)$ messages. Thus, when t failures occur, this algorithm is less efficient, in time and message complexity, than the one without early stopping in [Srik84]. In [Dole82b] it is also shown that at least $\min(f+2, t+1)$ phases are required to reach eventual Byzantine Agreement in the worst case. The second algorithm in [Dole82b] meets this lower bound, but requires $n > 2t^2+3t+5$ processes.

In this paper, we describe a simple early-stopping algorithm that overcomes up to $n/3$ faulty processes. It terminates in $\min(2f+4, 2t+1)$ phases and requires $O(nt^2)$ messages. Hence, it is simpler, terminates earlier and has a lower communication complexity than the first algorithm in [Dole82b].

The algorithm in this paper was inspired by an early-stopping algorithm that assumes message authentication [Perr84]. To remove this assumption, we used a broadcast primitive that provides properties of authentication. This primitive is interesting and important in its own right: it effectively restricts the disruptive behavior of faulty processes, and hence simplifies the design of fault-tolerant algorithms for systems with arbitrary failures. Several other applications of such primitives that resulted in efficient fault-tolerant algorithms are described in [Srik84], [Srik85].

The remainder of this paper is organized as follows. In § 2, we describe the properties of the broadcast primitive that is needed by our algorithm. Section 3 presents an early-stopping Byzantine Agreement algorithm for binary-valued messages (the multivalued case is considered in § 6). In § 4, we give an implementation of the broadcast primitive. The complexity of the algorithm is investigated in § 5. Optimizations are given in § 7. A discussion of the results concludes the paper.

2. Properties of the broadcast primitive. Srikanth and Toueg [Srik84] introduced a broadcast primitive that provides properties of authenticated broadcasts without using digital signatures. With this primitive, a process p wishing to broadcast m in round k (where a round is a *logical* phase) executes *broadcast* (p, m, k) and correct processes *accept* (p, m, k) in round k or later. Accepting a message in round k corresponds to receiving this message directly from the sender. Accepting a message in a subsequent round corresponds to receiving a message that was relayed by other processes. The broadcast primitive provides the following properties:

(1) *Correctness:* If correct process p executes broadcast (p, m, k) in round k, then every correct process accepts (p, m, k) in the same round.

(2) *Unforgeability:* If correct process p does not execute broadcast (p, m, k), then no correct process ever accepts (p, m, k).

(3) *Relay:* If a correct process accepts (p, m, k) in round r, then every other correct process accepts (p, m, k) in round $r+1$ or earlier.

These are properties of authenticated broadcasts. Property 1 shows that the primitive indeed implements a broadcast. A message broadcast by a correct process at the beginning of round k is received by all correct processes at the end of round k.

Property 2 asserts that a faulty process cannot forge the messages of a correct process. Accepting (p, m, k) corresponds to receiving a message m with p's signature in an authenticated system: in both cases, it can be deduced that if p is correct then p is indeed the originator of m.

Property 3 captures the relaying of signed messages: if a process receives a message m with p's signature in round r and relays it to all processes in round $r+1$, then all processes receive this signed message by round $r+1$.

The broadcast primitive introduced in [Srik84] achieves the above three properties of authentication without using digital signatures. It requires $n > 3t$ processes, and two phases of communication exchange for each logical round of the algorithm in which it is applied. By replacing signed communication with this primitive, we were able to automatically translate several *authenticated* algorithms (i.e., algorithms that assumed message authentication using unforgeable digital signatures) into *nonauthenticated* ones (i.e., algorithms that do not need this assumption) [Srik84], [Srik85].

Similarly, we could directly translate the authenticated Byzantine Agreement algorithm with early-stopping described in [Perr84] into a nonauthenticated one. However, since the authenticated algorithm in [Perr84] terminates in $\min(2f+4, 2t+2)$ phases, the nonauthenticated translation would terminate in $\min(4f+8, 4t+4)$ phases, and this is too inefficient.

The authenticated algorithm in [Perr84] uses a mechanism that enables a process to detect broadcasts by other processes, even if it does not directly receive the broadcast message. To achieve this "detection of broadcasts" property, the authenticated algorithm required two phases of communication per round. In this paper, we extend the primitive of [Srik84] to achieve this property without any additional cost in the number of phases. Using this modified primitive allows us to derive a nonauthenticated early-stopping algorithm that does not require any more phases than the authenticated one in [Perr84].

The "detection of broadcasts" property allows each correct process to maintain a set, *broadcasters*, that contains the names of processes that have broadcast messages using the primitive. The set *broadcasters* is initially empty. The primitive updates *broadcasters* as follows:

(4) *Detection of broadcasts:* If a correct process accepts (p, m, k) in round k or later, then every correct process has $p \in$ *broadcasters* at the end of round $k+1$.

Furthermore, if correct process p does not execute *broadcast* then a correct process can never have $p \in$ *broadcasters*.

Informally, if p broadcasts a message in round k, Property 4 provides that the *initiation* of this broadcast is detected by all correct processes by round $k+1$, even if the first correct process to accept p's message does so in a later round.

The extended primitive is described in detail in § 4. In the next section, we present a simple, nonauthenticated Byzantine Agreement algorithm with early-stopping that is based on this extended primitive.

3. A binary Byzantine Agreement algorithm with early-stopping.

3.1. Informal description. In Fig. 1, we present a nonauthenticated Byzantine Agreement algorithm with early-stopping, for systems of $n > 3t$ processes.[2] We first assume that $m \in \{0, 1\}$. This *binary* algorithm is later extended to the case where m is drawn from an arbitrary universe **M**.

```
p: broadcasters := ∅;
   if p = General then value = m else value = 0 fi    /* note that m ∈ {0, 1}*
   for r = 1 to t + 1 do
      if value ≠ 0 then broadcast (p, 1, r); stop fi
      consider broadcasters as updated in round r:
      if |broadcasters| < r − 1 then stop fi
      if in rounds r' ≦ r accepted (p_i, 1, i) for all i, 1 ≦ i ≦ r,
         where p_1 is the General, and the p_i's are distinct,
         then value := 1 fi
   od
```

FIG. 1. *The early-stopping binary Byzantine Agreement algorithm for process p.*

In our algorithm, 1 is the only value broadcast by correct processes. The value 0 is decided upon only by default. To decide 1 in round r, a correct process p must have accepted r messages, each message originated by a distinct process in a distinct round. It is also required that the originator of the first of these messages be the *General*. If p decides 1 in round r, it broadcasts 1 in round $r+1$ and stops. Properties 1 and 3 of the broadcast primitive ensure that p's message, and all r messages that p accepted by round r, will also be accepted by all correct processes by round $r+1$, so they all decide 1 and stop.

To decide 0 and stop in round r, a process must be sure that no correct process will later decide 1. This is the case if it has proof that in some previous round (say round $i < r$) no process broadcasts a message. (Note that to decide 1 after round i, a process must accept a message broadcast in round i.) To have such a proof, each process keeps a list of broadcasters using Property 4. If a process p discovers that there are fewer than $r-1$ broadcasters by round r, this is proof that there were no broadcasters in some previous round i. By Property 4, no correct process *ever* accepts a message originating in round i. Therefore, no process can later decide 1. Hence, p can safely decide 0 and stop.

In Theorem 3.2 we show that this algorithm terminates by round min $(f+2, t+1)$, where f is the number of processes that are actually faulty during execution.

[2] It is known that the Byzantine Agreement problem can be solved without authentication only if $n > 3t$ [Peas80].

3.2. Proof of correctness. We begin with some definitions. A process stops *at round* k if $r = k$ when it stops. A process *decides* m *at round* k if it stops at round k with *value* $= m$.

LEMMA 3.1. *All correct processes decide a value after at most* $t+1$ *rounds.*

Proof. The algorithm consists of a loop that terminates after at most $t+1$ rounds. □

We now show the algorithm's *validity.*

LEMMA 3.2. *If the General is correct and broadcasts* m*, then all correct processes decide* m.

Proof. Since the *General* is correct, it sets *value* $= m$. Two cases are possible.

(1) If $m = 1$, then the *General* broadcasts (*General*, 1, 1) in round $r = 1$, and stops. In round 1, all other correct processes have $|broadcasters| \geq r-1$ and, by Property 1 of our broadcast primitive, they accept (*General*, 1, 1). Therefore they all set *value* to 1 in round 1, and decide 1 in round 2.

(2) If $m = 0$, then the *General* does not broadcast any message in round 1. Therefore, correct processes never accept a (*General*, 1, 1) message. Hence, they never update their *value* variable, initially set to 0, and they all decide 0 when they stop. □

To prove the algorithm's *agreement* property we need the following two lemmas.

LEMMA 3.3. *If a correct process decides* 1 *at round* r*, then every correct process that executes round* r (*i.e., did not stop in an earlier round*) *decides* 1 *at round* r *or* $r+1$.

Proof. Let p be a correct process that decides 1 at round r, where $1 \leq r \leq t+1$. Consider when p sets *value* to 1. If it is at the beginning of round 1, then p must be the *General* and the proof of Lemma 3.2 (case 1) can be applied. There are two other possible cases:

(1) Process p sets *value* to 1 at the end of round $r-1$. Therefore, by round $r-1$, p must have accepted $(p_i, 1, i)$ messages for all i, $1 \leq i \leq r-1$, where p_1 is the *General*, and the p_i's are distinct. Note that $p \neq p_i$ for all i, $1 \leq i \leq r-1$. In round r, p broadcasts $(p, 1, r)$. From Property 4 of our broadcast primitive, since p accepted $(p_i, 1, i)$ for all i, $1 \leq i \leq r-1$, then every correct process that accepts round r messages must have $p_i \in broadcasters$ for all i, $1 \leq i \leq r-1$, after accepting round r messages. So at this point every correct process has $|broadcasters| \geq r-1$, and therefore cannot stop and decide 0 in round r.

By Properties 1 and 3 of our broadcast primitive, all correct processes that do not stop in round r must accept $(p, 1, r)$ and $(p_i, 1, i)$ for all i, $1 \leq i \leq r-1$, by the end of round r. Hence, they all set *value* to 1 in round r, and decide 1 in round r or $r+1$.

(2) Process p sets *value* to 1 at the end of round r. Since p also stops in round r it must be that $r = t+1$. So p must have accepted some messages $(p_i, 1, i)$ for $1 \leq i \leq t+1$, with distinct p_i's. One of these $t+1$ processes, say p_j, must be correct. By Property 2 of our broadcast primitive, p_j broadcast $(p_j, 1, j)$ in round j, decided 1 in round j, and set *value* to 1 in round $j-1 < j$. The proof now follows by applying case 1 to p_j. □

From Lemma 3.3 it is clear that if a correct process decides 1 at round r, then no correct process decides 0 at round $r' \geq r$. We now show the analogous result if a correct process decides 0.

LEMMA 3.4. *If a correct process decides* 0 *at round* r *then no correct process decides* 1 *at round* $r' \geq r$.

Proof. Let p be a correct process that decides 0 at round r. By Lemma 3.3, no correct process decides 1 at round $r' = r$. Assume, for contradiction, a correct process q decides 1 at round $r' > r$. We must have $r < t+1$, and therefore p decided 0 by stopping in round r with $|broadcasters| < r-1$. Since q decides 1 at round $r' > r$ it must have accepted $(p_i, 1, i)$ messages for all i, $1 \leq i \leq r'-1$. Hence, by Property 4 of our

broadcast primitive, p must have $p_i \in broadcasters$ for all i, $1 \leqq i \leqq r-1$, just after accepting round r messages. So p has $|broadcasters| \geqq r-1$ at round r, a contradiction. □

We now prove the algorithm's *agreement* property.

LEMMA 3.5. *All correct processes decide the same value.*

Proof. Let r be the first round in which a correct process decides, and p a correct process that decides in round r. No correct process decides in a round $r' < r$. If p decides 1 then, by Lemma 3.3, no correct process decides 0 in round $r' \geqq r$. If p decides 0 then, by Lemma 3.4, no correct process decides 1 in round $r' \geqq r$. Note that all correct processes stop, and therefore decide. □

THEOREM 3.1. *The algorithm in Fig. 1 solves the Byzantine Agreement problem.*

Proof. Immediate from Lemmas 3.1, 3.2 and 3.5. □

We now show the algorithm is early-stopping.

THEOREM 3.2. *If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop by round $\rho = \min(f+2, t+1)$.*[3]

Proof. By Lemma 3.1, all correct processes stop and therefore decide by round $t+1$. Let r be the first round a correct process decides, and p a correct process that decides in round r. There are two possible cases.

(1) Suppose p decides 1. If $r=1$ then p must be the *General*, and the proof of Lemma 3.2 shows that all other correct processes decide 1 at round 2. We now assume $r>1$. Since p decides 1 at round $r>1$, then p must have accepted $(p_i, 1, i)$ messages for $1 \leqq i \leqq r-1$. Since r is the first round a correct process broadcasts a message, then all p_i's, $1 \leqq i \leqq r-1$, must be faulty, and therefore $r-1 \leqq f$. From Lemmas 3.3 and 3.4, every correct process decides 1 by round $r+1 \leqq f+2$.

(2) If p decides 0, then, by Theorem 3.1, all correct processes also decide 0. Therefore, no correct process ever broadcasts a message. By Property 4 of our broadcast primitive, the variable *broadcasters* of a correct process can contain only names of faulty processes. Hence $|broadcasters| < f+1$, and any correct process that executes round $f+2$ must stop at this round. □

Suppose the *General* is correct. From the proof above, if $m=1$ then all correct processes decide 1 in round 2. However, if $m=0$ and there are f faulty processes during the execution of the algorithm, it could take $f+2$ rounds for the correct processes to decide 0. This undesirable characteristic is easily eliminated by allowing only the *General* to be inserted in *broadcasters* in round 2. With this modification, if $m=0$, then the correct processes have $|broadcasters| = 0 < r-1$ in round 2, and stop.

In the next section we describe the broadcast primitive invoked by our Byzantine Agreement algorithm. We postpone the algorithm's complexity analysis to the subsequent section.

4. The broadcast primitive and its communication complexity. The broadcast primitive in [Srik84] provides the three properties of authentication needed by our early-stopping algorithm. In this section, we extend a version of this primitive to provide the fourth property needed by our algorithm: *detection of broadcasts.* This extension is achieved by inserting one statement that updates the variable *broadcasters.*

The resulting primitive is presented in Fig. 2. It can be used in systems where $n > 3t$, and simulates the authenticated broadcast of message m by process p in round k of a synchronized algorithm. As presented here, it can be applied to algorithms in which correct processes broadcast at most one message throughout the algorithm (as in the algorithm of Fig. 1). The primitive requires two phases of communication

[3] Note that our implementation of the broadcast primitive requires two phases of communication per round.

Rules for broadcasting and accepting (p, m, k):

Round k:
  Phase $2k-1$: process p sends $(init, p, m, k)$ to all processes.
    Each process executes the following for each $m \in \mathbf{M}$:
  Phase $2k$:
    if received $(init, p, m, k)$ from p in phase $2k-1$
      and received only one $(init, p, _, _)$ message in all previous phases
    then send $(echo, p, m, k)$ to all;
    if received $(echo, p, m, k)$ from $\geq n-t$ distinct processes in phase $2k$
    then *accept* (p, m, k);

Round $k+1$:
  Phase $2k+1$:
    if received $(echo, p, m, k)$ from $\geq n-2t$ distinct processes q in phase $2k$
      and received only one $(echo, p, _, _)$ message from each such q
    then send $(init', p, m, k)$ to all;
    if received $(init', p, m, k)$ from $\geq n-2t$ distinct processes in phase $2k+1$
    then *broadcasters* := *broadcasters* $\cup\{p\}$;
  Phase $2k+2$:
    if received $(init', p, m, k)$ from $\geq n-t$ distinct processes in phase $2k+1$
    then send $(echo', p, m, k)$ to all;
    if received $(echo', p, m, k)$ from $\geq n-t$ distinct processes in phase $2k+2$
    then *accept* (p, m, k);

Round $r \geq k+2$;
  Phase $2r-1, 2r$:
    if received $(echo', p, m, k)$ from $\geq n-2t$ distinct processes in previous phases
      and not sent $(echo', p, m, k)$
    then send $(echo', p, m, k)$ to all;
    if received $(echo', p, m, k)$ from $\geq n-t$ distinct processes in previous phases
    then *accept* (p, m, k);

FIG. 2. *The broadcast primitive.*

exchange for each round of the algorithm in which it is applied: round k corresponds to phases $2k-1$ and $2k$ of the primitive.

There are four types of messages. A process wishing to broadcast sends an *init* message to all processes in the first phase of the broadcast. Processes send *echo* messages in the second phase, *init'* in the third phase, and *echo'* in later phases of the broadcast, according to the algorithm described in Fig. 2. When a process receives sufficient echoes of a message it *accepts* the message. In the description of the primitive, message fields marked by an "_" (underscore) can contain arbitrary values. For example, $(_, p, _, _)$ refers to any message sent by process p, and $(echo, p, _, _)$ refers to any message of type *echo* sent by p.

LEMMA 4.1. *If a correct process ever sends* $(echo', p, m, k)$, *then at least one correct process must have sent* $(echo', p, m, k)$ *in phase* $2k+2$.

Proof. Let l be the earliest phase in which any correct process q sends $(echo', p, m, k)$. If $l \geq 2k+3$, process q must have received $(echo', p, m, k)$ messages from at least $n-2t$ distinct processes, i.e., it must have received $(echo', p, m, k)$ from at least one correct process in phase $l-1$ or earlier. Hence, some correct process sends $(echo', p, m, k)$ before phase l, a contradiction. Therefore, $l = 2k+2$. □

LEMMA 4.2. *If a correct process ever sends* $(echo', p, m, k)$, *then p must have sent* $(init, p, m, k)$ *to at least one correct process in phase* $2k-1$.

Proof. By Lemma 4.1, if a correct process ever sends $(echo', p, m, k)$, then some correct process q must have sent $(echo', p, m, k)$ in phase $2k+2$. Therefore, process q must have received $(init', p, m, k)$ from at least $n-t$ processes in phase $2k+1$. At least

$n-2t$ of these processes are correct, and each of them must have received at least $n-2t$ (*echo*, p, m, k) messages in phase $2k$. Hence, at least one correct process must have sent (*echo*, p, m, k) in phase $2k$ and it must have received (*init*, p, m, k) from p in phase $2k-1$.

THEOREM 4.1. *The broadcast primitive in Fig. 2 has the four properties stated in* § 2.

Proof.

Correctness: Since p is correct, every correct process receives (*init*, p, m, k) in phase $2k-1$ and sends (*echo*, p, m, k) in phase $2k$. Hence, every process receives (*echo*, p, m, k) from at least $n-t$ correct processes in phase $2k$ and every correct process accepts (p, m, k) at the end of phase $2k$, i.e. at the end of round k.

Unforgeability: If p is correct and does not execute broadcast (p, m, k), it does not send any message (*init*, p, m, k) in phase $2k-1$, and no correct process sends (*echo*, p, m, k) in phase $2k$. Hence, no correct process can accept (p, m, k) in phase $2k$. If some correct process accepts (p, m, k) at the end of phase $2k+2$ or later, it must have received (*echo'*, p, m, k) messages from at least $n-t$ distinct processes, i.e., it must have received (*echo'*, p, m, k) from at least $n-2t$ correct processes. By Lemma 4.2, p must have sent (*init*, p, m, k) to at least one correct process in phase $2k-1$, a contradiction. Thus, no correct process ever accepts (p, m, k).

Relay: If $r=k$, then q must have received (*echo*, p, m, k) from at least $n-t$ distinct processes in phase $2k$. Hence, in the same phase, every correct process receives (*echo*, p, m, k) from at least $n-2t$ distinct processes and sends (*init'*, p, m, k) in phase $2k+1$. Therefore, every correct process sends (*echo'*, p, m, k) during phase $2k+2$ and every correct process accepts (p, m, k) at the end of phase $2k+2$, i.e., round $r+1$. Now, suppose $r \geqq k+1$. Let q be a process that accepts (p, m, k) in phase i where $i=2r-1$ or $2r$. Process q must have received (*echo'*, p, m, k) from at least $n-t$ distinct processes by phase i. Hence, every correct process receives (*echo'*, p, m, k) from at least $n-2t$ distinct processes by phase i, and therefore sends (*echo'*, p, m, k) at or before phase $i+1$. Thus, every correct process receives (*echo'*, p, m, k) from at least $n-t$ distinct processes by phase $i+1$, and therefore accepts (p, m, k) by the end of phase $i+1$, i.e., in round $r+1$ or earlier.

Detection of broadcasts: We first show the second part of this property. If a correct process adds p to *broadcasters*, then it received (*init'*, p, __, __) messages from at least $n-2t$ distinct processes in phase $2k+1$. Therefore, at least one correct process sent an (*init'*, p, __, __) message in phase $2k+1$, and it must have received (*echo*, p, __, __) messages from at least $n-2t$ distinct processes in phase $2k$. Hence, at least one correct process must have received an (*init*, p, __, __) message directly from p in phase $2k-1$. Therefore p executed the broadcast primitive.

We now prove the first part of Property 4. If a correct process accepts (p, m, k) in phase $2k$ (round k), then it must have received (*echo*, p, m, k) messages from at least $n-t$ distinct processes in phase $2k$. Therefore, every correct process received (*echo*, p, m, k) messages from at least $n-2t$ distinct correct processes in phase $2k$, and these must be the first (*echo*, p, __, __) message that each one sent. Hence, every correct process sends (*init'*, p, m, k) messages in phase $2k+1$. So, every correct process receives (*init'*, p, m, k) messages from at least $n-t$ distinct processes in phase $2k+1$, and adds p to *broadcasters* in this phase, i.e., in round $k+1$.

If a correct process accepts (p, m, k) in phase $k' \geqq 2k+2$, then it must have received (*echo'*, p, m, k) messages from at least $n-t$ distinct processes by phase k'. Let l be the earliest phase in which any correct process q sends an (*echo'*, p, m, k) message. By Lemmas 4.1 and 4.2, $l=2k+2$, and q must have received (*init'*, p, m, k) messages from at least $n-t$ distinct processes in phase $2k+1$. Therefore all correct processes received

(*init'*, *p*, *m*, *k*) messages from at least $n-2t$ distinct correct processes in phase $2k+1$, and they all add *p* to *broadcasters* in this phase. □

Note that the broadcast primitive need not terminate if the broadcaster is faulty, i.e., there is no a priori bound on the number of phases needed to establish the four properties. However, when a process stops in round *r* of an agreement algorithm, then it also stops its execution of the underlying broadcast primitive at the end of round *r*, i.e., at the end of phase $2r$. We now informally show that processes stopping their participation in the underlying broadcast primitive in the same round they decide does not affect the Byzantine Agreement algorithm presented in the previous section.

Let *r* be the first round in which a correct process decides, and let *p* be a process that decides in round *r*. If *p* stops and decides 1 in round *r* then it continues its execution of the broadcast primitive till the end of round *r*. From the proof of Lemma 3.3, every correct process will set *value* = 1 by the end of round *r*, and therefore decide 1 when it stops, regardless of *p*'s participation in execution of the broadcast primitive after round *r*. If *p* decides 0, stopping its participation in the broadcast primitive can only prevent a message from being accepted by some process, or prevent the insertion of a process's name into the *broadcasters* variable of some other process (so the size of |*broadcasters*| can only *decrease* compared to its size if *p* never stops executing the primitive). This cannot cause any correct process to decide 1 or to continue executing the algorithm for more rounds than if *p* never stopped its execution of the broadcast primitive.

5. The complexity of the binary Byzantine Agreement algorithm. In computing the message complexity of the primitive, we consider only messages sent by correct processes. It is not possible to restrict the number of messages sent by faulty processes.

LEMMA 5.1. *In an algorithm in which the primitive of Fig.* 2 *is applied, each correct process sends a total of* $O(n)$(—, *p*, —, —) *messages for each process p.*

Proof. Consider a given process *p*. Each correct process p_i accepts at most one (*init*, *p*, —, —) message from *p* and thus sends at most one (*echo*, *p*, —, —) message to each process. p_i considers at most one (*echo*, *p*, —, —) message from each other process p_j throughout the algorithm. Since p_i sends (*init'*, *p*, *m*, *k*) only on receiving at least $n-2t$ (*echo*, *p*, *m*, *k*) messages, and since $n>3t$, each correct process sends at most $\lfloor n/(n-2t)\rfloor \leqq 2$(*init'*, *p*, —, —) messages.

We now show that at most 3 distinct (*echo'*, *p*, —, —) messages are sent (to all) by the set of correct processes (and hence by each individual correct process). Let *C* be a set of $n-t$ correct processes. By Lemma 4.1, if a correct process sends (*echo'*, *p*, *m*, *k*) in any phase, then some correct process *q* must have sent (*echo'*, *p*, *m*, *k*) in phase $2k+2$. For correct process *q* to send (*echo'*, *p*, *m*, *k*) in phase $2k+2$, it must receive (*init'*, *p*, *m*, *k*) messages from at least $n-t$ processes, i.e., from at least $n-2t$ correct processes in *C*. Therefore, the sending of an (*echo'*, *p*, *m*, *k*) by a correct process requires the sending of (*init'*, *p*, *m*, *k*) by $n-2t$ correct processes in *C* to all processes, i.e., a total of $(n-2t)n$ such messages sent by processes in *C*. Since each correct process sends at most 2 (*init'*, *p*, —, —) messages to all processes, then processes in *C* send a total of at most $2(n-t)n$ (*init'*, *p*, —, —). Hence, the total number of distinct (*echo'*, *p*, —, —) sent by the set of correct processes is bounded by $\lfloor 2(n-t)n/(n-2t)n\rfloor \leqq 3$. Thus, each correct process sends at most 3 (*echo'*, *p*, —, —) messages. Therefore, each correct process sends at most 6 (—, *p*, —, —) messages to all processes, i.e., a total of $O(n)$(—, *p*, —, —) messages. □

We now analyse the message complexity of the Byzantine Agreement algorithm in Fig. 1. By Lemma 5.1, each correct process sends $O(n)$(—, *p*, —, —) messages

throughout the algorithm for each process p. There are n processes, and therefore each correct process sends $O(n^2)$ messages throughout the algorithm. Hence, all correct processes send a total of $O(n^3)$ messages.

However, we can apply some standard techniques [Dole83], [Srik84] to reduce the number of messages to $O(nt^2)$ as follows. A set of $3t+1$ *active* processes execute the algorithm as described in Fig. 1. Broadcasts by the active processes are to *all* processes. The rest of the processes are *passive*; they only receive messages sent by the active processes. By Theorem 3.2, the active processes decide the same value $m \in \{0, 1\}$ by round $\rho = \min(f+2, t+1)$. It is easy to see that the passive processes can also be made to decide m by round ρ, from the messages they received from active processes.

THEOREM 5.1. *The binary Byzantine Agreement algorithm described above requires $O(nt^2)$ messages and a total of $O(nt^2 \log n)$ message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within* $\min(2f+4, 2t+2)$ *phases.*

Proof. Only active processes p broadcast, and therefore each correct active process sends $(_, p, _, _)$ messages only for active p's. Hence, by Lemma 5.1, each correct active process sends at most $O(nt)$ messages during the execution of the algorithm. So all correct active processes send a total of $O(nt^2)$ messages. We assume it takes $\log n$ bits to encode the name of a process. By Theorem 3.2, the algorithm terminates within $\min(f+2, t+1)$ rounds. Since the broadcast primitive requires two phases of communication for each round, the algorithm terminates within $\min(2f+4, 2t+2)$ phases. □

6. A multivalued Byzantine Agreement algorithm with early-stopping. In the Appendix, we extend our algorithm to the case where the *General*'s value m is drawn from an arbitrary universe $\mathbf{M}$. In that section we prove the following theorem:

THEOREM 6.1. *The multivalued Byzantine Agreement algorithm requires $O(nt^2)$ messages and a total of $O(nt \cdot (\log n + \log |\mathbf{M}|) + nt^2 \cdot \log n)$ message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within* $\min(2f+5, 2t+3)$ *phases.*

Proof. In the Appendix. □

7. Optimizations. We can easily modify the two Byzantine Agreement algorithms to terminate one phase earlier without increasing the communication complexity. Note that a process broadcasting a message in round $t+1$, the last round, can send the message *directly* to all the processes (without using the broadcast primitive) because messages sent in the last round are not relayed further and therefore do not need to be authenticated. So the last round of the algorithms takes only one communication phase instead of two, and termination occurs at phase $\min(2f+4, 2t+1)$ and $\min(2f+5, 2t+2)$ for the binary and multivalued algorithms, respectively. For this to work however, a slightly modified version of the broadcast primitive must be used in round t, to ensure that a message accepted by a correct process in round t is accepted by all correct processes just one phase after round t [Srik84].

The concept of a process *committing* a value v during the execution of a Byzantine Agreement algorithm was introduced in [Dole82b]. Informally, a process commits a value v if it knows that all correct processes will decide v by the time the algorithm terminates. This can happen before the process is able to stop its participation in the algorithm. However, as soon as a process commits a value, it can use it for the purpose for which the algorithm was started. With our algorithms, it is not difficult to see that processes may commit one phase earlier than they stop, without increasing the

communication complexity. Processes can commit at phase $\min(2f+3, 2t+1)$ and $\min(2f+4, 2t+2)$ for the binary and multivalued algorithms, respectively.

Finally, the communication complexity can be further reduced to $O(nt \cdot \log t + t^3 \cdot \log t)$ bits and $O(nt \cdot (\log t + \log |\mathbf{M}|) + t^3 \cdot \log t)$ bits for all the binary and multivalued algorithms above, at the cost of one additional phase. This is achieved by executing the algorithm among active processes only At termination, the value decided is directly broadcast to the passive processes and they decide on the majority of the values received.

8. Discussion and conclusions. Nonauthenticated algorithms for reaching agreement in systems with arbitrary failures have been notoriously complicated, and difficult to derive and prove correct. Such algorithms are even more sophisticated and difficult to understand when they exhibit early stopping.

In this paper, we took advantage of the properties provided by our broadcast primitive to derive a conceptually simple early-stopping algorithm. The algorithm derived is also more efficient than those previously known. It is interesting to note that the communication complexity of the best known Byzantine Agreement algorithm *without* early stopping is no lower than ours. This is somewhat surprising, since one would expect to pay some price in communication complexity for the property of stopping early.

The algorithm was derived and proved correct assuming four properties of broadcasts. The algorithm (and its proof) is independent of any particular implementation of the broadcast primitive that achieves these four properties. There may be more efficient or simpler implementations of such a primitive than the one we proposed.

The results presented in this paper illustrate once more the power and versatility of our primitive. It provides properties of message authentication and other properties that effectively restrict the disruptive behavior of faulty processes. The problem of designing algorithms for systems with arbitrary failures is thereby simplified. We have applied the same technique to problems other than agreement, e.g., to the problem of optimal clock synchronization in the presence of arbitrary faults [Srik85], [Beck86].

Appendix. The multivalued algorithm. We consider the case where the *General*'s value m is drawn from an arbitrary universe $\mathbf{M}$. In this case, the *General* executes a modified broadcast primitive in the first round. This primitive is presented in Fig. 3, and has the following properties:

THEOREM A.1. *The broadcast primitive in Fig. 3 has the four properties described in § 2, and the following uniqueness property*:

(5) *If a correct process accepts* $(p, m, 1)$, *then no correct process ever accepts* $(p, m', 1)$ *with* $m' \neq m$.

Proof. The proof of the first four properties is similar to the proof of the four properties in Theorem 3.2, and is left to the reader. We now sketch the proof of Property 5.

We first note that if a correct process sends $(echo', p, m, 1)$ in phase 2, then no correct process sends $(echo', p, m', 1)$ in phase 2 for any $m' \neq m$. Otherwise, at least $n-t$ processes sent $(echo, p, m, 1)$ and $n-t$ processes sent $(echo, p, m', 1)$ in phase 1. Since $n > 3t$, this implies that at least one correct process sent both $(echo, p, m, 1)$ and $(echo, p, m', 1)$ in phase 1, and this is not possible.

We also note that if a correct process accepts $(p, m, 1)$, then at least one correct process sends $(echo', p, m, 1)$, and therefore at least one correct process sends $(echo', p, m, 1)$ in phase 2. Property 5 is an immediate consequence of the two remarks above. □

```
Rules for broadcasting and accepting (p, m, 1) for p = General:
Round 1:
  Phase 0: process p sends (init, p, m, 1) to all processes.
    Each process executes the following for each m ∈ M:
  Phase 1:
    if received (init, p, m, 1) from p in phase 0
      and received only one (init, p, __, __) message in this phase
    then send (echo, p, m, 1) to all;
    if received (echo, p, m, 1) from ≧ n − 2t distinct processes in phase 1
    then broadcasters := broadcasters ∪ {p};
  Phase 2:
    if received (echo, p, m, 1) from ≧ n − t distinct processes q in phase 1
      and received only one (echo, p, __, __) message from each such q
    then send (echo', p, m, 1) to all;
    if received (echo', p, m, 1) from ≧ n − t distinct processes in phase 2
    then accept (p, m, 1);
Round r ≧ 2:
  Phase 2r − 1, 2r:
    if received (echo', p, m, 1) from ≧ n − 2t distinct processes in previous phases
      and not sent (echo', p, m, 1)
    then send (echo', p, m, 1) to all;
    if received (echo', p, m, 1) from ≧ n − t distinct processes in previous phases
    then accept (p, m, 1);
```

FIG. 3. *The monobroadcast primitive invoked by the General in the first round.*

This primitive is denoted *monobroadcast* because of the *uniqueness* property. Note that the first round takes 3 phases, but the later rounds take two phases each. Consider the communication complexity of *monobroadcast*:

LEMMA A.1. *In an algorithm in which the General invokes monobroadcast in the first round, each correct process sends a total of* $O(n)$ (__, *General*, __, 1) *messages.*

Proof. The proof is similar to the proof of Lemma 5.1, and will be omitted here. □

The multivalued Byzantine Agreement algorithm is derived from the binary one by having the *General* execute *monobroadcast* to broadcast $m \neq 0$ in round 1. All broadcasts initiated in later rounds use *broadcast.* A process sets its value to $m \neq 0$ in round r if it accepted (*General*, m, 1) and (p_i, 1, i) for all i, $2 \leqq i \leqq r$, where the p_i's are distinct.

THEOREM A.2. *The algorithm described above is a multivalued Byzantine Agreement algorithm. If there are f faulty processes during the execution of the algorithm, then all correct processes decide by round* $\rho = \min(f+2, t+1)$.

Proof. The *uniqueness* property of the *General*'s broadcast primitive effectively reduces the multivalued case to the binary case, and the proof of Theorem A.2 is nearly identical to the proofs of Theorems 3.1 and 3.2. It will not be repeated here. □

To reduce the message complexity, the algorithm could be executed on a selected set of $3t+1$ active processes, as explained in § 5. The complexity of the resulting multivalued Byzantine Agreement algorithm is as follows.

THEOREM A.3. *The multivalued Byzantine Agreement algorithm requires* $O(nt^2)$ *messages and* $O(nt \cdot (\log n + \log |\mathbf{M}|) + nt^2 \cdot \log n)$ *message bits. If there are f faulty processes during the execution of the algorithm, then all correct processes decide and stop within* $\min(2f+5, 2t+3)$ *phases.*

Proof. By Lemma A.1, each correct active process sends $O(n)$(__, *General*, __, 1) messages. Hence, all active processes send a total of $O(nt)$(__, *General*, __, 1) messages. Assuming it takes $\log n$ bits to encode a process's name, and $\log |\mathbf{M}|$ to encode a

message $m \in \mathbf{M}$, then each (__, *General*, m, 1) message is $\log n + \log |\mathbf{M}|$ bits. By Lemma 5.1, the total number of other messages sent by the active processes is $O(nt^2)$, and each such message is $O(\log n)$ bits. For the number of phases required for termination, note that the first round takes three phases, and all subsequent ones take two phases, and apply Theorem A.2. □

REFERENCES

[Beck86] M. BECK, T. K. SRIKANTH AND S. TOUEG, *Implementation issues in clock synchronization*, Lecture Notes of the Asilomar Workshop on Fault-Tolerant Distributed Computing, Asilomar, CA, March 16, 1986, Springer-Verlag, Berlin-New York-Heidelberg, to appear.

[Dole81] D. DOLEV, *Unanimity in an unknown and unreliable environment.* Proc. 22nd Annual Symposium on Foundations of Computer Science, Nashville, TN, Oct. 1981, pp. 159-168.

[Dole82a] D. DOLEV AND H. R. STRONG, *Requirements for agreement in a distributed system*, Tech. Rep. RJ 3418, IBM Research Laboratory, San Jose, CA, March 1982.

[Dole82b] D. DOLEV, R. REISCHUCK AND H. R. STRONG, '*Eventual*' *is earlier than* '*immediate*', Proc. 23rd Symposium on Foundations of Computer Science, Chicago, IL, November 1982, pp. 196-203.

[Dole82c] D. DOLEV, M. J. FISCHER, R. FOWLER, N. A. LYNCH AND R. STRONG, *An efficient algorithm for Byzantine Agreement without authentication*, Inform. Control, 52 (1982), pp. 257-274.

[Dole83] D. DOLEV AND H. R. STRONG, *Authenticated algorithms for Byzantine Agreement*, this Journal, 12 (1983), pp. 656-666.

[Hadz83] V. HADZILACOS, *Byzantine Agreement under restricted types of failures (not telling the truth is different from telling lies)*, Tech. Rep. 19-83, Aiken Computation Laboratory, Harvard Univ., Cambridge, MA, June 1983.

[Lamp82a] L. LAMPORT AND M. FISCHER, *Byzantine Generals and transaction commit protocols*, Opus 62, SRI International, April 1982.

[Lamp82b] L. LAMPORT, R. SHOSTAK AND M. PEASE, *The Byzantine Generals problem*, ACM Trans. Prog. Languages Systems, 4 (1982), pp. 382-401.

[Lync82] N. A. LYNCH, M. J. FISCHER AND R. J. FOWLER, *A simple and efficient Byzantine Generals algorithm*, Proc. Second IEEE Symposium on Reliability in Distributed Software and Data Base Systems, Pittsburgh, PA, 1982, pp. 46-52.

[Peas80] M. PEASE, R. SHOSTAK AND L. LAMPORT, *Reaching agreement in the presence of faults*, J. ACM, 27 (1980), pp. 228-234.

[Perr84] K. J. PERRY AND S. TOUEG, *An authenticated Byzantine Generals algorithm with early-stopping*, Tech. Rep. 84-620, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, June 1984.

[Perr86] ———, *Distributed agreement in the presence of processor and communication faults*, IEEE Trans. Software Engrg., SE-12 (1986), pp. 477-482.

[Reis82] R. REISCHUK, *A new solution for the Byzantine Generals problem*, Tech. Rep. RJ 3673, IBM Research Laboratory, San Jose, CA, Aug. 1982.

[Srik84] T. K. SRIKANTH AND S. TOUEG, *Simulating authenticated broadcasts to derive simple fault-tolerant algorithms*, Tech. Rep. 84-623, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, July 1984. *Distributed Computing*, Springer-Verlag, Berlin-Heidelberg-New York, to appear.

[Srik85] ———, *Optimal clock synchronization*, J. Assoc. Comput. Mach., to appear.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987

© 1987 Society for Industrial and Applied Mathematics

TIGHT COMPARISON BOUNDS ON THE COMPLEXITY OF PARALLEL SORTING*

YOSSI AZAR† AND UZI VISHKIN‡

Abstract. The problem of sorting n elements using p processors in a parallel comparison model is considered. Lower and upper bounds which imply that for $p \geqq n$, the time complexity of this problem is $\Theta(\log n/\log(1+p/n))$ are presented. This complements [AKS-83] in settling the problem since the AKS sorting network established that for $p \leqq n$ the time complexity is $\Theta(n \log n/p)$. To prove the lower bounds we show that to achieve $k \leqq \log n$ parallel time, we need $\Omega(n^{1+1/k})$ processors.

Key words. parallel computation, parallel sorting, sorting networks, sorting lower bounds, bounded parallelism, AKS network

1. Introduction. Apparently, there is no problem in Computer Science which received more attention than sorting. Knuth [Kn-73], for instance, found that existing computers devote approximately a quarter of their time to sorting. The advent of parallel computers stimulated intensive research of the sorting with respect to various models of parallel computation. Extensive lists of references which recorded this activity are given in [Ak-85], [BHe-86] and [Th-83].

Most of the fastest serial and parallel sorting algorithms are based on binary comparisons. In these algorithms the number of comparisons is typically the primary measure of time complexity. Any lower bound on the number of comparisons required for a problem clearly implies a time lower bound for such algorithms. In the present paper, we restrict our attention to a parallel comparison model where only comparisons are counted. In measuring time complexity within this model, we do not count steps in which communication among the processors, movement of data and memory addressing are performed. We also avoid counting steps in which consequences are deduced from comparisons that were performed. Note that our lower bounds apply to all algorithms, based on comparison, in any parallel access machine (PRAM) including PRAMs which allow simultaneous access to the same common memory location for read and write purposes. See [BHo-82] for a discussion on hierarchy of models that implies this.

In a serial decision tree model, we wish to minimize the number of comparisons. The goal of an algorithm in a parallel comparison model is to minimize the number of comparison rounds as well as the total number of comparisons performed.

Let k stand for the number of comparison rounds (time) of an algorithm in the parallel comparison model. Given an algorithm, let $u(k, n)$ denote the *total* (over all rounds of the algorithm) number of comparisons required by the algorithm to sort any n elements in k rounds. $u(k, n)$ is the upper bound on the number of comparisons in the worst case. Let $c(k, n)$ denote the *minimum total* number of comparisons required to sort any n elements in k rounds (over all possible algorithms).

* Received by the editors February 19, 1986; accepted for publication (in revised form) July 14, 1986.

† Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, 69 978 Tel Aviv, Israel. This research is part of the M.Sc. thesis of this author.

‡ Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. Present address, Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, 69 978 Tel Aviv, Israel. The research of this author was supported by National Science Foundation grants DCR-8318874 and DCR-841-3359, Office of Naval Research grant N00014-85-K-0046 and Department of Energy grant DE-ACO2-76ER03077.

The known $\Omega(n \log n)$ comparisons lower bound for sorting in a serial decision tree model implies that, for any k, $c(k, n) = \Omega(n \log n)$. This lower bound can be matched by upper bounds as follows: For $k = c \log n$, the sorting network of [AKS-83] obtains $u(k, n) = O(n \log n)$, where $c > 0$ is a constant which is implied by the network. For $k > c \log n$, the result $u(k, n) = O(n \log n)$ also holds. To see this, simply simulate the AKS network by slowing it down to work in k rounds.

For $k = 1$, $c(1, n) = \frac{1}{2}(n^2 - n)$. This is since any sorting algorithm which works in one round must perform all comparisons. Otherwise, suppose that a dispensed comparison is between two successive elements in the sorted order; the algorithm will clearly fail to distinguish their order. On the other hand, observe that performing all comparisons simultaneously yields a one round algorithm in the parallel comparison model that matches this lower bound exactly, i.e., $u(1, n) = \frac{1}{2}(n^2 - n)$.

So, it remains to consider the situation for $1 < k \leqq c \log n$.

We state our **main result** which is proved in § 3:

$$c(k, n) > k\left(\frac{n^{1+1/k}}{e} - n\right) \quad \text{for any } k,\ n \geqq 1, \quad \text{where } e \text{ is the base of the natural logarithm.}$$

Corollaries of the main result. Suppose we have p processors with the interpretation that each processor can perform at most one comparison at each round. Observe that $kp \geqq c(k, n)$ or $p \geqq c(k, n)/k$. Therefore,

COROLLARY 1. *Any k-round* $(k \geqq 1)$ *parallel algorithm for sorting n elements needs* $p > (n^{1+1/k}/e) - n$ *processors. This yields* $p = \Omega(n^{1+1/k})$ *for* $k \leqq c \log n$ *where c is any constant such that* $0 < c < 1$.[1]

COROLLARY 2. *The number of rounds required to sort n elements using* $p \geqq n$ *processors is* $k = \Omega(\log n / \log (1 + p/n))$.

Proof. $p > (n^{1+1/k}/e) - n$ implies $1 + p/n > n^{1/k}/e$ and therefore $k > \log n/(1 + \log (1 + p/n))$. Hence, for $p \geqq n$, $k = \Omega(\log n / \log (1 + p/n))$.

COROLLARY 3. *If* $p = n \log^{\beta} n$ *for* $\beta > 0$ *then the number of rounds required to sort n elements is* $k = \Omega(\log n / \beta \log \log n)$. *This is an immediate corollary of Corollary* 2.

A parallel algorithm is said to achieve *optimal speed up* if its running time is proportional to $\text{Seq}(n)/p$, where $\text{Seq}(n)$ is a lower bound on the serial running time, n is the size of the problem being considered and p is the number of processors used.

COROLLARY 4. *If the number of processors is larger than n by an order of magnitude, then it is impossible to design an optimal speed up comparison sorting algorithm. More formally, suppose that the number of processors p is not* $O(n)$ *(i.e.,* $n = o(p)$*) then there is no (comparison) sorting algorithm which runs in time* $O(n \log n/p)$.

Section 4 presents upper bounds which match these new lower bounds. Specifically, we describe a parallel comparison algorithm that sorts n elements in $O(\log n / \log (1 + p/n))$ rounds using $p \geqq n$ processors.

To understand better the significance of the lower and upper bounds of the present paper we will use one more equivalent formulation of the results.

COROLLARY 5. *Suppose we are given* $p \geqq n$ *processors to sort n elements. The total number of comparisons performed by the fastest parallel sorting algorithm is*

$$\Theta\left(\frac{p/n}{\log (1 + p/n)} n \log n\right).$$

The factor $n \log n$ *represents the serial lower and upper bounds for sorting using comparisons. The other factor represents the deviation from optimal speed up.*

[1] Throughout the paper, log refers to the natural logarithm.

In studying the limit of parallel algorithms it is interesting to identify asymptotically the minimal time k that can be achieved by an optimal speed up algorithm. We call this minimal time the parallelism *break point* of the problem being considered. Valiant [Va-75] proved that $\Theta(\log\log n)$ is the break point for finding the maximum among n elements. Borodin and Hopcroft [BHo-82] gave a lower bound and Kruskal [Kr-83] an upper bound to prove that $\Theta(\log\log n)$ is the break point for merging two sorted lists, where n is the length of each list. The above two lower bounds were also obtained in a parallel comparison model (which is therefore often referred to as Valiant's model). The present paper enables us to add sorting to the list of problems for which the break point was identified. Specifically, Corollary 4 complements the sorting network of [AKS-83] in proving that $\Theta(\log n)$ is the break point for sorting n elements. It is interesting to compare the "pattern" in which the break point occurs in these three problems. The elegant lower bound proofs of Valiant and Borodin and Hopcroft show that $\Omega(\log\log n)$ rounds are required if n processors are used for the problems of finding the maximum and merging, respectively. The algorithms of Valiant and Kruskal run in $O(\log\log n)$ rounds using $n/\log\log n$ processors for each of these problems, respectively. This distinctly isolates the break points for these two problems since the asymptotic time bound cannot improve by increasing the number of processors from $n/\log\log n$ to n. On the other hand, such degenerate isolation does not occur in the sorting problem. Specifically, Corollary 5 implies that increasing the number of processors asymptotically always yields asymptotic decrease in the number of comparison rounds.

We note that we have proved a first nontrivial lower bound in a parallel comparison model for $\log n$ time. The problem solved in this paper was open for some time. Interestingly, our proof is relatively simple and based only on elementary methods from discrete and continuous mathematics.

Let us review works on sorting n elements in a parallel comparison model. Haggkvist and Hell [HH-81] proved that if k, the number of rounds, is *constant*, then $\Omega(n^{1+1/k})$ processors are required to sort n elements. Using random graphs and a certain probability space, Bollobas and Thomason [BT-83] proved that almost every algorithm that uses $p = O(n^{3/2}\log n)$ processors sorts n elements in two rounds. Bollobas [Bo-86] iterated the random graph techniques to prove that n elements can be sorted in a *constant* number of rounds k using $O(n^{1+1/k}\log n)$ comparisons. This almost matches the Haggkvist–Hell lower bound.

Remark. Conversely, these results imply that for $p = O(n^{1+\varepsilon})$ processors, it is impossible to sort in less than $k = 1/\varepsilon$ rounds, but we can sort in $k = 1/\varepsilon + 1$ rounds. So these upper and lower bounds are at most one round apart when k is constant.

However, a closer look at this lower bound of Haggkvist and Hell reveals the following. They actually proved that if k, the number of rounds, is a variable, then $p > (n^{1+1/k}/2^{k+1}k) - (n/2k)$ processors are required to sort n elements. We compare this result with Corollary 1 which was given above. Observe that their proof implies that $p = \Omega(n^{1+1/k})$ *only* when k is constant. Even for constant k which is not very large this asymptotic lower bound contains a very small constant factor (for instance, if $k = 100$ then $n^{1+1/k}$ is multiplied by less than 2^{-100}). Moreover, their result becomes trivial for $k \geqq \sqrt{\log n}$. This is since for this range their result implies an asymptotic bound which is $O(n)$ for the number of processors p as can be readily verified. On the other hand, Corollary 1 states that $p > (n^{1+1/k}/e) - n$ for every k. As was indicated above, this implies that $p = \Omega(n^{1+1/k})$, for any $k < c\log n$, where $0 < c < 1$ is a constant.

We note two additional papers whose titles are related to the title of the present paper. Leighton [Le-84] proposed an adaptation of AKS network to bounded degree

n-node networks. Meyer auf der Heide and Wigderson [MW-85] gave a $\sqrt{\log n}$ lower bound for parallel sorting by n processors in some variant of PRAM. Their model is not comparable to the parallel comparison model considered here. The trivial $\log n$ lower bound for parallel sorting by n processors in the parallel comparison model does not allow noncomparison algorithms like bucket sort. On the other hand, ranking an element among n other elements can be done in one round of comparisons using n processors in the parallel comparison model, while their PRAM seems to require nonconstant time using n processors.

2. The parallel comparison model. Let V be a set of n elements taken from a totally ordered domain. The *parallel comparison model of computation* allows algorithms that work as follows. The algorithm consists of time steps called *rounds.* In each round binary comparisons are performed simultaneously. The input for each comparison are two elements of V. The output of each comparison is one of the following two: $<$ or $>$. Note that we do not allow equality between two elements of V. This can be done without loss of generality, since we define the order between two equal input elements to be the order of their indices. Each item may take part in several comparisons during the same round.

Remark. Our discussion uses the following correspondence between each round and a graph. The elements are the vertices. Each comparison to be performed is an undirected edge which connects its input elements. Each computation results in orienting this edge from the largest element to the smallest.

Suppose we performed r rounds where $r>0$ is some integer. Consider any function of V that can be computed using the comparisons performed in these r rounds without any further comparisons of elements in V. Our model defines such a function to be *computable following round r.* Note that this definition suppresses all computational steps that do not involve comparisons of elements in V. Which comparisons to perform at round $r+1$ and the input for each such comparison should be functions which are computable following round r. We are interested in sorting the elements in V from the smallest to the largest in k rounds, where the integer k can be either constant or a function of n.

Recall that $c(k, n)$ denotes the minimum *total* number of comparisons required to sort any n elements in k rounds (over all possible algorithms).

3. The lower bound.

THE MAIN THEOREM. *$c(k, n) > k((n^{1+1/k}/e)-n)$ for any k, $n \geqq 1$, where e is the base of the natural logarithm.*

Proof of the Main Theorem. By induction on k and n.

The base of the induction. For $k=1$ and every $n \geqq 1$, clearly $c(1, n) = (n^2-n)/2 > (n^2/e)-n$. For $n=1, 2$ and every $k \geqq 1$, $c(k, 1) \geqq 0 > k(1/e-1)$, $c(k, 2) > 0 > k(4/e-2) \geqq k((2^{1+1/k}/e)-2)$.

The inductive assumption. Given k, n, if $k' \leqq k$ and $n' < n$, or $k' < k$ and $n' \leqq n$, then $c(k', n') > k'((n'^{1+1/k'}/e)-n')$.

Take any k-round algorithm for sorting a set V of n elements. The first round of the algorithm consists of some set E of comparisons. Recall that we look at them as edges in the graph $G=(V, E)$. An *independent set* in G is a subset of vertices from V such that no two vertices are adjacent by an edge in E. An independent set is *maximal* if it is not a proper subset of another independent set. Consider the graph of the first round of comparisons. Let S be a maximal independent set in this graph and denote $x=|S|$. Each of the $n-x$ elements of $\bar{S}$ must share an edge with an element of S,

otherwise S is not maximal. For our lower bound proof, we restrict our attention to linear orders on V, in which element of S is greater than each element of $\bar{S}$. For any of these orders it is impossible to obtain any information regarding the relation between two elements of S or two elements of $\bar{S}$ using comparisons between an element of S and an element of $\bar{S}$. Therefore, aside from these $n-x$ comparisons, there must be at least $c(k-1, x)$ comparisons to sort S and at least $c(k, n-x)$ comparisons to sort $\bar{S}$. This implies the following recursive inequality:

$$c(k, n) \geqq c(k, n-x)+n-x+c(k-1, x).$$

By the inductive assumption

$$> k\left(\frac{(n-x)^{1+1/k}}{e}-(n-x)\right)+(n-x)+(k-1)\left(\frac{x^{1+1/(k-1)}}{e}-x\right).$$

By opening parentheses and permuting terms we get

$$=\frac{k}{e}(n-x)^{1+1/k}+\frac{k-1}{e}x^{1+1/(k-1)}+n-kn$$

$$=\frac{k}{e}n^{1+1/k}\left[\left(1-\frac{x}{n}\right)^{1+1/k}+\left(1-\frac{1}{k}\right)\frac{x^{1+1/(k-1)}}{n^{1+1/k}}+\frac{1}{k}\frac{e}{n^{1/k}}\right]-kn.$$

Recall the Geometric Arithmetic Mean Inequality: $\alpha a+\beta b \geq a^{\alpha}b^{\beta}$, where $\alpha+\beta=1$, $\alpha, \beta, a, b \geqq 0$. By taking $\alpha=1-1/k$, $\beta=1/k$, $a=x^{1+1/(k-1)}/n^{1+1/k}$, $b=e/n^{1/k}$ we get

$$\geqq\frac{k}{e}n^{1+1/k}\left[\left(1-\frac{x}{n}\right)^{1+1/k}+\frac{x}{n^{1-1/k^2}}\frac{e^{1/k}}{n^{1/k^2}}\right]-kn.$$

Recall that the increasing sequence $(1+1/k)^k$ converges to e and therefore, $e^{1/k}> 1+1/k$. This implies

$$\geqq\frac{k}{e}n^{1+1/k}\left[\left(1-\frac{x}{n}\right)^{1+1/k}+\frac{x}{n}\left(1+\frac{1}{k}\right)\right]-kn.$$

Recall Bernoulli's Inequality: $(1-\alpha)^t \geqq 1-\alpha t$ for $t \geqq 1$, $\alpha \leqq 1$. This implies

$$\geqq\frac{k}{e}n^{1+1/k}\left[1-\frac{x}{n}\left(1+\frac{1}{k}\right)+\frac{x}{n}\left(1+\frac{1}{k}\right)\right]-kn$$

$$=\frac{k}{e}n^{1+1/k}-kn=k\left(\frac{n^{1+1/k}}{e}-n\right).$$

This completes the proof of the Main Theorem.

4. Upper bound.

THEOREM. (Due to Noga Alon.) *Given n elements from a totally ordered domain, there is an algorithm in a parallel comparison model for sorting these elements in $O(\log n/\log(1+p/n))$ rounds using $p \geqq n$ processors.*

Proof. First recall the AKS comparison network. It sorts n elements in $O(\log n)$ rounds using $p = n/2$ processors (i.e., comparisons in each round). We give an algorithm in a parallel comparison model. Each round of the new algorithm is called *superround.* The algorithm is derived from AKS network by simply shrinking $\delta = 0.5 \log (1+p/n)$ rounds of this network into one superround.

The construction of the algorithm is based on the following idea. We aim that the following assertion will hold.

ASSERTION. *After superround r, the following things are available*: (1) *The pair of input elements for each comparison performed in the first* δr *rounds of* AKS *network.* (2) *The result of each such comparison.*

This assertion implies that after $O(\log n/\log (1+p/n))$ superrounds the results of all comparisons of AKS network are available and the sorting is completed (since it is computable).

We show how to satisfy the assertion for any superround r. For $r=0$ the assertion trivially holds. We show how to satisfy the assertion for superround r assuming that it is satisfied for any superround $<r$.

The fact that we relate to a comparison network implies that each element, which is compared in round $\delta(r-1)+i$, where $1 \leqq i \leqq \delta$, is one of at most 2^{i-1} elements which are outputs of comparisons of the first $\delta(r-1)$ rounds (or input elements). By the inductive assumption, each of these outputs is available following superround $r-1$. Therefore, each comparison in round $\delta(r-1)+i$, is actually one of $(2^{i-1})^2$ possible pairs. All we do is perform all these possible comparisons simultaneously (for $1 \leqq i \leqq \delta$). These comparisons clearly include the actual comparisons performed by AKS network in these rounds. It remains to show that this construction also yields the pairs of input elements to each comparison which was actually performed in each of these rounds. For this we show by simple induction that the actual pair of each comparison as well as its result are available for all rounds $\leqq \delta(r-1)+i$, $0 \leqq i \leqq \delta$. For $i=0$, this follows from the inductive assumption of the assertion for $r-1$. Suppose that for all rounds $<\delta(r-1)+i$, the actual pairs compared as well as their result are available. Each element participating in round $\delta(r-1)+i$ is an outcome of the actual comparisons of preceding rounds and their results. They are known by the inductive assumption. Therefore, the input pair for each such comparison is known. We already argued that the result of each such comparison was found by our algorithm. This completes the proof of the induction for i. Taking $i=\delta$, we complete the inductive proof of the assertion.

The number of comparisons that the algorithm has to perform in each superround is:

$$\frac{n}{2} \sum_{i=1}^{\delta} (2^{i-1})^2 < \frac{n}{2} \frac{4^{\delta}}{3} < \frac{n}{2} 2^{2\delta}.$$

But $\delta = 0.5 \log (1+p/n)$, and therefore, this number of comparisons is not more than

$$\frac{n}{2} 2^{\log (1+p/n)} \leqq \frac{n}{2}\left(1+\frac{p}{n}\right) = \frac{n+p}{2} \leqq p.$$

So there are enough processors to perform all these comparisons.

Acknowledgments. We are grateful to Noga Alon for letting us include his proof of the upper bound in this paper. Helpful comments by Noga Alon, Alan Borodin, Moshe Dubiner and Mike Paterson are gratefully acknowledged.

REFERENCES

[Ak-85] S. AKL, *Parallel Sorting Algorithms*, Academic Press, New York, 1985.

[AKS-83] M. AJTAI, J. KOMLOS AND E. SZEMEREDI, *An $O(n \log n)$ sorting network*, Proc. 15th ACM Symposium on Theory of Computation, 1983, pp. 1-9. *Sorting in $c \log n$ parallel steps*, Combinatorica, 3 (1983), pp. 1-19.

[Al-85] N. ALON, *Expanders, sorting in rounds and superconcentrators of limited depth*, Proc. 17th ACM Symposium on Theory of Computing, 1985, pp. 98-102.

[BR-82] B. BOLLOBAS AND M. ROSENFELD, *Sorting in one round*, Israel J. Math., 38 (1981), pp. 154-160.

[BT-83] B. BOLLOBAS AND A. THOMASON, *Parallel sorting*, Discrete Appl. Math., 6 (1983), pp. 1-11.

[BHe-86] B. BOLLOBAS AND P. HELL, *Sorting and graphs*, preprint.

[Bo-86] B. BOLLOBAS, *Sorting algorithms*, chapter of future monograph.

[BHo-82] A. BORODIN AND J. E. HOPCROFT, *Routing, merging and sorting on parallel models of computation*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 338-344.

[HH-80] R. HAGGKVIST AND P. HELL, *Graphs and parallel comparison algorithms*, Congr. Numer., 29 (1980), pp. 497-509.

[HH-81] ———, *Parallel sorting with constant time for comparisons*, this Journal, 10 (1981), pp. 465-472.

[HH-82] ———, *Sorting and merging in rounds*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 465-473.

[Kn-73] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[Kr-83] C. P. KRUSKAL, *Searching, merging and sorting in parallel computation*, IEEE Trans. Comput., C-32 (1983), pp. 942-946.

[Le-84] F. T. LEIGHTON, *Tight bounds on the complexity of parallel sorting*, Proc. 16th ACM Symposium on Theory of Computing, 1984, pp. 71-80.

[MW-85] F. MEYER AUF DER HEIDE AND A. WIGDERSON, *The complexity of parallel sorting*, Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 532-540.

[Re-81] R. REISCHUK, *A fast probabilistic parallel sorting algorithm*, Proc. 22nd IEEE Symposium on Foundations of Computer Science, 1981, pp. 212-219.

[RV-83] J. REIF AND L. G. VALIANT, *A logarithmic time sort for linear size network*, Proc. 15th ACM Symposium on Theory of Computation, 1983, pp. 10-16.

[SV-81] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging and sorting in a parallel model of computation*, J. Algorithms, 2 (1981), pp. 88-102.

[Th-83] C. THOMPSON, *The VLSI complexity of sorting*, IEEE Trans. Comput., C-32 (1983).

[Va-75] L. G. VALIANT, *Parallelism in comparison problems*, this Journal, 4 (1975), pp. 348-355.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987

© 1987 Society for Industrial and Applied Mathematics
004

EQUATIONAL BASES FOR IF-THEN-ELSE*

ALAN H. MEKLER† AND EVELYN M. NELSON‡

Abstract. Four formalizations of the instruction if_then_else are considered. Let K be an equational class of algebras. Assume the language has constants $\perp$ in case II, T, F, $\perp$ in case III and T in case IV. Any equational class with $\perp$ is assumed to be strict, i.e. satisfies $f(_, \perp, _) = \perp$ for all functions f. Form the classes I K^d, II $K^{d\perp}$, III K^t and IV $K^{t'}$ by adding an operation $[\ ,\ ,\ ,\ ]$ in cases I and II defined by

$$\text{I}\quad [x, y, z, w] = \begin{cases} z & \text{if } x = y, \\ w & \text{else,} \end{cases} \qquad \text{II}\quad [x, y, z, w] = \begin{cases} z & \text{if } x = y \neq \perp, \\ w & \text{if } \perp \neq x \neq y \neq \perp, \\ \perp & \text{else,} \end{cases}$$

and by adding an operation $[\ ,\ ,\ ]$ in cases III and IV defined by

$$\text{III}\quad [x, y, z] = \begin{cases} y & \text{if } x = T, \\ z & \text{if } x = F \\ \perp & \text{else,} \end{cases} \quad \text{and} \quad \text{IV}\quad [x, y, z] = \begin{cases} y & \text{if } x = T, \\ z & \text{else.} \end{cases}$$

(In case III only the one point algebra and those algebras in which T, F, $\perp$ are distinct are used to form K^t.) In all four cases a finite basis for the equational theory of K^x ($x = d, d\perp, t, t'$) relative to the equational theory of K are given. For K^d such a basis was previously known. If the word problem for K is decidable then so is the equational theory of K^x. For K^d, as was previously known, the converse holds, but for K^t the implication does not always reverse.

Key words. if then else, equational logic, equational basis, variety, decidability, discriminator algebra, test algebra

AMS(MOS) subject classifications. 68C99, 08B05

Introduction. The instruction "if_then_else_" is a common feature of high level programming languages; in order to understand the syntax of programs involving such an instruction, it is necessary to know when two expressions, built from "if_then_else_" and other operations, are equal under all appropriate interpretations. Thus, a description and, where possible, an algorithm, are required for the equational logic of algebras equipped with an operation which models "if_then_else_."

This paper is devoted to a study of that equational logic. It was inspired by, and is a generalization of, the paper by Bloom and Tindell [3]. Following them, we will discuss four different interpretations of "if_then_else_." In each case, we begin with an equational class K of some type of algebras, and give an equational basis for the class of all algebras obtained from members of K by adding an appropriate operation which models "if_then_else_." We prove that, in each case, the equational theory of this class is decidable whenever the word problem for K is decidable. Moreover, we present examples where this equational theory is undecidable, even though the equational theory of K is decidable.

Our approach is different from that of Bloom and Tindell, although the axioms we present are essentially the same as theirs. In addition to the syntactic methods in [3], we use a semantic analysis of the intended models. We give one uniform approach

* Received by the editors August 12, 1985; accepted for publication (in revised form) July 20, 1986. This work was partly supported by the Natural Sciences and Engineering Research Council of Canada, and conducted during the second author's visiting appointment at Simon Fraser University.

† Department of Mathematics and Statistics, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6.

‡ Department of Mathematics and Statistics, McMaster University, Hamilton, Ontario, Canada L8S 4K1.

which handles all four variants of "if_then_else_" with only minor modifications. For one variant, the quaternary discriminator (see below) an equational basis for the class described above, starting with *any* equational class K, was given both in McKenzie [9] and Bloom and Tindell [3]. For the other three variants of "if_then_else," Bloom and Tindell give an equational basis only for the case that the class K contains no operations, except appropriate constants. Moreover, we do not see any way to extend the methods of Bloom and Tindell to yield our results.

The first interpretation of "if_then_else_" which we consider is a ternary operation $[\ ,\ ,\ ]$ together with constants $\perp$, T, F which are to model "undefined," "true," and "false," respectively. An equational class K of a type Σ which contains the constants $\perp$, T, F is called strict if it satisfies, for all $\sigma \in \Sigma$, $\sigma(x_1, \cdots, x_n) = \perp$ whenever one of the $x_i = \perp$. Given a strict equational class K, we form the class K^t of all algebras of type $\Sigma \cup \{[\ ,\ ,\ ]\}$ whose underlying Σ-algebras belong to K and in which the operation $[\ ,\ ,\ ]$ satisfies

$$[a, b, c] = \begin{cases} b & \text{if } a = T, \\ c & \text{if } a = F, \\ \perp & \text{else.} \end{cases}$$

We present a finite set AX of equations in $[\ ,\ ,\ ]$, $\perp$, T, F which, together with the equations of K, provide an equational basis for K^t; for the case that Σ consists of just the constants $\perp$, T, F, this axiomatization was given in Bloom and Tindell [3]. Our results also provide an algorithm for the equational theory of K^t modulo an algorithm for the word problem for K. The latter is not a necessary condition. We provide an equational class K of semigroups with constants T, F, and $\perp$ with an undecidable word problem such that the equational theory of K^t is decidable. (Of course there are also trivial examples, since for any K satisfying $T \approx F$, K^t contains only one element algebras.) We also provide an example where the equational theory of K^t is undecidable, although the equational theory of K is decidable.

We remark that Benoit [1] claims to give an equational basis for K^t; although his basis is correct (in fact, it is the same as ours), his purported proof is not correct, and we provide a counterexample at the end of § 2. The original version of the preprint of Guessarian and Meseguer [7] also claimed this result, but the proof there was also incorrect, for essentially the same reason. In the final version [this Journal, 16 (1987), pp. 332–357] they deal instead with many-sorted algebras with certain restrictions on the allowed types of operations, and give an equational basis for the analogue of what we call K^t.

The second variant of "if_then_else_" which we consider is the quaternary discriminator; given any equational class K of any type Σ, we consider the class K^d of all $\Sigma \cup \{[\ ,\ ,\ ,\ ]\}$-algebras whose underlying Σ-algebras belong to K and on which $[\ ,\ ,\ ,\ ]$ acts like the discriminator, i.e., $[a, b, c, d] =$ *if* $a = b$ *then* c *else* d. Equational classes generated by discriminator algebras (or, equivalently, by algebras with the ternary discriminator) have been studied extensively; the reader is referred to [4] and [11]. Equational bases for K^d were given both in McKenzie [9] and Bloom and Tindell [3]. It seems to have been known for some time (see the introduction to McKenzie [8] and Lemma 9.2 of Burris and Werner [5]) that the decidability of the equational theory of K^d is equivalent to the decidability of the universal theory of K; since K is an equational class, the latter is equivalent to the word problem for K. We present here a new approach to the equational logic of K^d which is analogous to that for K^t, produces easy proofs of the decidability results, and extends, with minor modifications, to the case of a quaternary discriminator modified by the constant $\perp$.

The last case which we consider in detail, then, is that of modified discriminator algebras. Given a strict equational class K of type Σ (which includes the constant $\perp$), $K^{d\perp}$ is the class of all $\Sigma \cup \{[\ ,\ ,\ ,\ ]\}$-algebras whose underlying Σ-algebras belong to K and such that

$$[a, b, c, d] = \begin{cases} \perp & \text{if } a = \perp \text{ or } b = \perp, \\ c & \text{if } a = b \neq \perp, \\ d & \text{else.} \end{cases}$$

Bloom and Tindell [3] present an equational basis for $K^{d\perp}$ in the case that Σ contains no operations besides $\perp$; we extend their results by providing equational bases for $K^{d\perp}$ for all strict equational classes K. An immediate consequence of our methods is that the equational theory of $K^{d\perp}$ is decidable whenever the word problem for K is decidable, and the converse is true provided every equation of the form $s \approx \perp$ satisfied by K is a consequence of the strictness of the operations.

The paper ends with some comments on the modification of the first variant of "if_then_else_" obtained by dropping the constants $\perp$ and F.

1. Preliminaries on equational deduction. We recall that, given some type of (universal) algebras, an *equation* is a pair (u, v) of elements of the absolutely free algebra (word algebra or term algebra) of that type over some countable set X of generators (variables). We usually write $u \approx v$ for (u, v). An algebra A satisfies the equation $u \approx v$ iff $h(u) = h(v)$ for every homomorphism h from the free algebra to A.

For a set E of equations, and an equation $u \approx v$, $E \vdash u \approx v$ iff the equation $u \approx v$ is a logical consequence of E via the usual rules of equational logic, as described, for example, in Bloom and Tindell [3, p. 680]:

(R1) For each term s infer $s \approx s$;
(R2) From $s \approx t$ infer $t \approx s$;
(R3) From $s \approx t$ and $t \approx u$ infer $s \approx u$;
(R4) From $s_i \approx t_i$ for $i \leqq n$ infer $\sigma(s_1, \cdots, s_n) \approx \sigma(t_1, \cdots, t_n)$ for each n-ary operation σ;
(R5) From $s \approx t$ infer $e(s) \approx e(t)$ for each endomorphism e of the absolutely free algebra.

It is a well-known result, due to Birkhoff (see [4, 14.19]) that $E \vdash u \approx v$ iff every algebra which satisfies all the equations in E also satisfies $u \approx v$.

We further define $E \vdash^c u \approx v$ (Bloom and Tindell use $\vdash^0$) iff the equation $u \approx v$ is a logical consequence of the equations in E using only rules (R1) to (R4).

Note that $E \vdash^c u \approx v$ iff (u, v) belongs to the congruence $\theta(E)$, generated by E, on the absolutely free algebra. Moreover, since $\vdash^c$ essentially corresponds to equational deduction in the type where all the variables $x \in X$ are treated as constants, Birkhoff's result also implies that $E \vdash^c u \approx v$ iff for each homomorphism h from the absolutely free algebra to an algebra A, if $h(s) = h(t)$ for all $(s, t) \in E$ then $h(u) = h(v)$.

Notation. For terms p and s and a variable x, $p(x \to s)$ is the term obtained by substituting s for x wherever x appears in p.

Remark. Define $E \vdash^1 u \approx v$ to hold iff there exists a term p, a variable $x \in X$, and $(s, t) \in E$ such that $\{u, v\} = \{p(x \to s), p(x \to t)\}$. That is, $E \vdash^1 u \approx v$ iff v can be obtained from u by replacing some (but not necessarily all) subterms equal to s by t, or vice versa, for $(s, t) \in E$. Then it is well known, and easy to prove, that the relation $\vdash^c$ is the transitive hull of $\vdash^1$, i.e., $E \vdash^c u \approx v$ iff there exist terms $u_0 = u, u_1, u_2, \cdots, u_n = v$ $(n \geqq 0)$ such that $E \vdash^1 u_i \approx u_{i+1}$ for each $i < n$.

2. Test algebras. The first version of "if_then_else_" which we consider is a ternary operation [, ,]. An algebra A with this operation and constants $\perp$, T, F is called a *test algebra* if, for all $a, b, c \in A$

$$[a, b, c] = \begin{cases} b & \text{if } a = T, \\ c & \text{if } a = F, \\ \perp & \text{else.} \end{cases}$$

Note that if A is a test algebra then either $\perp$, T, F are all distinct in A, or A is a one-element algebra.

In this section, Σ will be a type of algebras which includes the constants $\perp$, T, F but not [, ,] and Σ^t will be the type obtained by adding the ternary operation [, ,] to Σ. A Σ-algebra A is called *strict* iff for all operations $\sigma \in \Sigma$ of arity $n \geqq 1$, $\sigma(x_1, \cdots, x_n) = \perp$ whenever one of the $x_i = \perp$. This condition is evidently equational.

Our aim here is to give, for any equational class K of strict Σ-algebras, an equational basis for the class K^t defined as in the introduction. Note that if K satisfies one of the equations $\perp \approx T$, $\perp \approx F$, $T \approx F$ then K^t contains only the one element algebra.

Let AX be the set consisting of the identities (1) to (10) below; these identities were introduced by Bloom and Tindell [3], as a basis for all identities satisfied by test algebras.

(1) $[T, x, y] \approx x$,
(2) $[x, x, y] \approx [x, T, y]$,
(3) $[F, x, y] \approx y$,
(4) $[x, y, x] \approx [x, y, F]$,
(5) $[\perp, x, y] \approx \perp$,
(6) $[x, \perp, \perp] \approx \perp$,
(7) $[x, [y, u, v], [y, z, w]] \approx [y, [x, u, z], [x, v, w]]$,
(8) $[x, [x, y, u], z] \approx [x, y, z]$,
(9) $[x, z, [x, y, u]] \approx [x, z, u]$,
(10) $[[x, y, z], u, v] \approx [x, [y, u, v], [z, u, v]]$.

Remark. An immediate consequence of identities (8) and (9) is

(11) $[x, [x, y, z], [x, u, v]] = [x, y, v]$.

We define AX_Σ to be the set AX above, together with, for each $\sigma \in \Sigma$ of arity $n \geqq 1$, and each $i \leqq n$, the equation

(*) $$\sigma(x_1, \cdots, x_{i-1}, [y, u, v], \cdots, x_n) = [y, \sigma(x_1, \cdots, x_{i-1}, u, \cdots, x_n), \sigma(x_1, \cdots, x_{i-1}, v, \cdots, x_n)].$$

We are going to prove that AX_Σ together with the equations satisfied by K is an equational basis for K^t.

Note that every Σ^t-algebra satisfying AX_Σ is strict; this is an immediate consequence of (*) and (5).

For a set X (of "variables"), let $F_\Sigma X$ and $F_{\Sigma^t} X$ be the absolutely free algebras (i.e., word algebras, or term algebras) of type Σ and Σ^t respectively, over the set X.

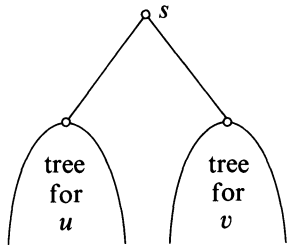
We define the notion of *simple term* in $F_{\Sigma^t} X$, the absolutely free Σ^t algebra over X, as follows:

The set S of simple terms is the smallest subset of $F_{\Sigma^t} X$ such that
(i) $F_\Sigma X \subseteq S$,
(ii) If $s \in F_\Sigma X$ and $u, v \in S$ then $[s, u, v] \in S$.

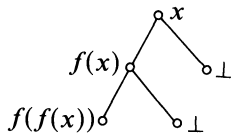
The simple terms are analogous to the "simple premise trees" of Bloom and Tindell. They are precisely those terms with the property that if $[r, u, v]$ is a subterm then r does not involve the operation $[\ ,\ ,\ ]$, and if $\sigma(u_1, \cdots, u_n)$ is a subterm then none of the u_i involve $[\ ,\ ,\ ]$. It is an immediate consequence of (10) and (*) that for every term $t \in F_{\Sigma'}X$ there is an effectively given simple term s such that $AX_\Sigma \vdash s \approx t$. Thus we may, and will, confine ourselves to equational deduction for simple terms.

The simple terms can again be represented by binary trees, now labelled with elements of $F_\Sigma X$. The tree corresponding to $s \in F_\Sigma X$ is

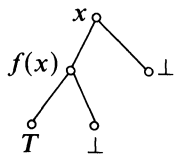
and the tree corresponding to the simple term $[s, u, v]$ is



Thus, for example, the binary tree corresponding to the simple term $u = [x, [f(x), f(f(x)), \bot], \bot]$ is



Now, under any interpretation of u in any test algebra, $u = f(f(x))$ if $x = T$ and $f(x) = T$, and $u = \bot$ otherwise. However, in any algebra, if $x = T$ and $f(x) = T$ then $f(f(x)) = f(T) = f(x) = T$, and hence under any interpretation in any test algebra, u has the same value as the following binary tree:



Thus, we must consider the logic of the equations which must be satisfied as we "evaluate" a tree down one of its paths. The appropriate logical deduction here is that denoted by $\vdash^c$, and we must prove that our axioms allow us to replace a label v, at some node of a tree, by w, whenever the equations which "occur" along the path to that node imply (in the sense of $\vdash^c$) that $v \approx w$, since that tree and its thereby obtained modification will be equal under any interpretation in any test algebra. In fact, this result, which appears as Proposition 2.3 below, is our principal tool. First, we give a precise definition of what is meant by "the equations that occur on the path to a node," and present some preliminary results.

For a binary tree t and a vertex α of t, $E(t, \alpha)$, *the set of equations on the path through t from the root to α*, is defined inductively on the depth of t as follows:

(i) If α is the root of t, then $E(t, \alpha) = \varnothing$;

(ii) If $t=[x, u, v]$ then

$$E(t, \alpha)=\begin{cases}\{x \approx T\} \cup E(u, \alpha) & \text{if } \alpha \text{ is a vertex in } u, \\ \{x \approx F\} \cup E(v, \alpha) & \text{if } \alpha \text{ is a vertex in } v.\end{cases}$$

The first half of the following lemma, and its proof, appear in Bloom and Tindell [3, 39.1 and 39.2], and the proof of the second half is analogous to the proof of the first.

LEMMA 2.1.

$$AX \vdash [x, [y, u, v], z] \approx [x, [y, [x, u, w], v], z],$$
$$[x, [y, u, v], z] \approx [x, [y, u, [x, v, w]], z],$$
$$[x, z, [y, u, v]] \approx [x, z, [y, [x, w, u], v]]$$

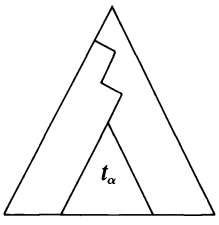
and

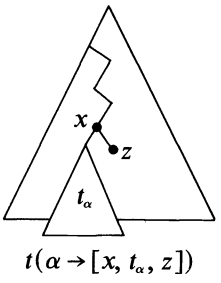
$$[x, z, [y, u, v]] \approx [x, z, [y, u, [x, w, v]]].$$

Notation. For a vertex α of a tree t, t_α, *the branch of* t *at* α is the tree consisting of α and all its successors in t, labelled as in t. If α is not a leaf, then $t_{\alpha T}$, the T-branch of t at α, and $t_{\alpha F}$, the F-branch of t at α, are the unique trees such that $t_\alpha=[x, t_{\alpha T}, t_{\alpha F}]$ (where x is the label of α). Further, for trees s and t and a vertex α in t, $t(\alpha \to s)$ is the tree obtained from t by replacing the branch at α with the tree s.

PROPOSITION 2.1. *If* $x \approx T$ *is in* $E(t, \alpha)$ *then* $AX \vdash t \approx t(\alpha \to [x, t_\alpha, z])$ *for any* $z \in X \cup \{\perp, T, F\}$. *Analogously, if* $x \approx F$ *is in* $E(t, \alpha)$ *then* $AX \vdash t \approx t(\alpha \to [x, z, t_\alpha])$.

Proof.





The proof is by induction on the length of the path from the root of t to α.

If the path has length 1, then the result is an immediate consequence of (8) which implies

$$[x, s, u] \approx [x, [x, s, z], u] \quad \text{for all terms } s \text{ and } u.$$

If the path has length $n+1$ and if β is the predecessor of α, then there are two cases: either β has label x and α is the root of $t_{\beta T}$, so that we can use the same argument as above, or $x \approx T$ is one of the equations in $E(t, \beta)$ and we may use the induction hypothesis to obtain that

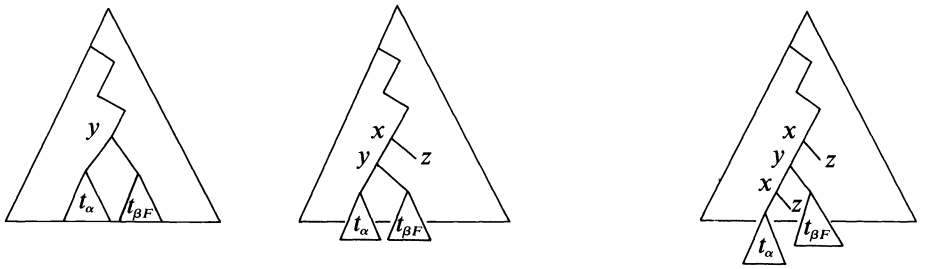
$$AX \vdash t \approx t(\beta \to [x, t_\beta, z]).$$

Suppose the label of β is y. There are again two cases: α is the root of $t_{\beta T}$ or $t_{\beta F}$. In the first case, $t_\beta=[y, t_\alpha, t_{\beta F}]$ and

$$AX \vdash [x, t_\beta, z]=[x, [y, t_\alpha, t_{\beta F}], z]$$
$$\approx [x, [y, [x, t_\alpha, z], t_{\beta F}], z] \quad \text{by Lemma 2.1;}$$

hence (see Fig. 1 below),

$$\begin{aligned} t &\approx t(\beta \to [x, t_\beta, z]) \\ &\approx t(\beta \to [x, [y, [x, t_\alpha, z], t_{\beta F}], z]) \\ &\approx t(\beta \to [y, [x, t_\alpha, z], t_{\beta F}]) \quad \text{by the induction hypothesis} \\ &= t(\alpha \to [x, t_\alpha, z]) \quad \text{as required.} \end{aligned}$$



t $\quad$ $t(\beta \to [x, t_\beta, z])$ $\quad$ $t(\beta \to [x, [y, [x, t_\alpha, z], t_{\beta F}], z])$

FIG. 1

In the second case $t_\beta = [y, t_{\beta T}, t_\alpha]$ and

$$\begin{aligned} AX \vdash [x, t_\beta, z] &= [x, [y, t_{\beta T}, t_\alpha], z] \\ &\approx [x, [y, t_{\beta T}, [x, t_\alpha, z]], z] \quad \text{by Lemma 2.1,} \end{aligned}$$

and hence

$$\begin{aligned} t &\approx t(\beta \to [x, t_\beta, z]) \\ &\approx t(\beta \to [x, [y, t_{\beta T}, [x, t_\alpha, z]], z]) \\ &\approx t(\beta \to [y, t_{\beta T}, [x, t_\alpha, z]]) \quad \text{by the induction hypothesis} \\ &= t(\alpha \to [x, t_\alpha, z]) \quad \text{as required.} \end{aligned}$$

The second part of the proposition is proved similarly, using the part of Lemma 2.1 for F instead of T.

PROPOSITION 2.2. *For all $p \in F_\Sigma X$ and $x \in X$, if x appears in p then*

$$AX_\Sigma \vdash p(x \to [y, u, v]) \approx [y, p(x \to u), p(x \to v)].$$

Proof. The proof is by induction on the complexity of p. If $p \in X \cup \{\bot, T, F\}$ then $p = x$ and the claim is trivial.

Suppose $p_1, \cdots, p_n$ satisfy the claim, and that $p = \sigma(p_1, \cdots, p_n)$ for $\sigma \in \Sigma$. If x appears in p, then x appears in at least one of the p_i. It is enough to prove the claim in case x appears in exactly one of the p_i, say in p_1. Then we have

$$\begin{aligned} AX_\Sigma \vdash p(x \to [y, u, v]) &= \sigma(p_1(x \to [y, u, v]), p_2, \cdots, p_n) \\ &\approx \sigma([y, p_1(x \to u), p_1(x \to v)], p_2, \cdots, p_n) \\ &\qquad \text{by the induction hypothesis} \\ &\approx [y, \sigma(p_1(x \to u), p_2, \cdots, p_n), \sigma(p_1(x \to v), p_2, \cdots, p_n)] \\ &\qquad \text{by (*)} \\ &\approx [y, p(x \to u), p(x \to v)] \quad \text{as required.} \end{aligned}$$

COROLLARY 1. *For all* $p \in F_\Sigma X$ *and* $x \in X$,

$$AX_\Sigma \vdash [y, p(x \to y), z] \approx [y, p(x \to T), z]$$

and

$$AX_\Sigma \vdash [y, z, p(x \to y)] \approx [y, z, p(x \to F)].$$

Proof. If x does not appear in p the claim is trivial, so assume x does appear in p. Then

$$\begin{aligned} AX_\Sigma \vdash [y, p(x \to y), z] &\approx [y, [y, p(x \to y), p(x \to y)], z] \\ &\approx [y, p(x \to [y, y, y]), z] \quad \text{by Proposition 2.2} \\ &\approx [y, p(x \to [y, T, y]), z] \quad \text{by (2)} \\ &\approx [y, [y, p(x \to T), p(x \to y)], z] \\ &\approx [y, p(x \to T), z] \quad \text{as required.} \end{aligned}$$

The other identity is proved similarly, using (4) instead of (2).

COROLLARY 2. *For all* $p \in F_\Sigma X$ *and* $x \in X$

$$AX_\Sigma \vdash [y, [p(x \to y), u, v], z] \approx [y, [p(x \to T), u, v], z]$$

and

$$[y, z, [p(x \to y), u, v]] \approx [y, z, [p(x \to F), u, v]].$$

Proof.

$$\begin{aligned} AX_\Sigma \vdash [y, [p(x \to y), u, v], z] &\approx [y, [y, [p(x \to y), u, v], [p(x \to y), u, v], z]] \quad \text{by (8)} \\ &\approx [y, [[y, p(x \to y), p(x \to y)], u, v], z] \quad \text{by (10)} \\ &\approx [y, [[\, y, p(x \to T), p(x \to y)], u, v], z] \quad \text{by Corollary 1} \\ &\approx [y, [y, [p(x \to T), u, v], [p(x \to y), u, v], z]] \quad \text{by (10)} \\ &\approx [y, [p(x \to T), u, v], z] \quad \text{by (8).} \end{aligned}$$

The other equation is proved similarly.

DEFINITION. For any equational class K of strict Σ-algebras, let $E(K)$ be the set of all equations satisfied by K. For a tree t as above and a vertex α, we define $E_K(t, \alpha) = E(t, \alpha) \cup E(K)$.

PROPOSITION 2.3. *For a tree* t *and a vertex* α *with label* u, *if* $E_K(t, \alpha) \vdash^c u \approx v$ *then* $AX_\Sigma \cup E(K) \vdash t \approx t'$ *where* t' *is obtained from* t *by changing the label at* α *to* v.

Proof. It is enough, by the Remark in § 1, to prove the claim in the case that $E_K(t, \alpha) \vdash^1 u \approx v$. Suppose that $(s, s') \in E_K(t, \alpha)$ and $p \in F_\Sigma X$ such that $u = p(x \to s)$, $v = p(x \to s')$. If $(s, s') \in E(K)$ then we trivially have $AX_\Sigma \cup E(K) \vdash t \approx t'$, so we may assume $(s, s') \in E(t, \alpha)$. Thus, s' is either T or F; we suppose $s' = T$ (the case $s' = F$ is analogous).

If α is a leaf in t, then

$$\begin{aligned} AX_\Sigma \vdash t &\approx t(\alpha \to [s, t_\alpha, z]) \quad \text{by Proposition 2.1} \\ &= t(\alpha \to [s, p(x \to s), z]) \end{aligned}$$

$$\approx t(\alpha \to [s, p(x \to T), z]) \quad \text{by Corollary 1 to Proposition 2.2}$$

$$= t(\alpha \to [s, v, z])$$

$$\approx t(\alpha \to v) \quad \text{by Proposition 2.1}$$

$$= t'.$$

If α is not a leaf in t, then

$$AX_\Sigma \vdash t \approx t(\alpha \to [s, t_\alpha, z]) \quad \text{by Proposition 2.1}$$

$$= t(\alpha \to [s, [p(x \to s), t_{\alpha T}, t_{\alpha F}], z])$$

$$\approx t(\alpha \to [s, [p(x \to T)), t_{\alpha T}, t_{\alpha F}], z]) \quad \text{by Corollary 2 to Proposition 2.2}$$

$$= t(\alpha \to [s, [v, t_{\alpha T}, t_{\alpha F}], z])$$

$$\approx t(\alpha \to [v, t_{\alpha T}, t_{\alpha F}]) \quad \text{by Proposition 2.1.}$$

DEFINITION. A vertex α in a tree t is called *K-inaccessible* if $E_K(t, \alpha) \vdash^c$ either $T \approx F$ or $T \approx \perp$ or $F \approx \perp$. If K is the class of all Σ-algebras then α is called *inaccessible.* If α is not K-inaccessible, then it is called *K-accessible.*

THEOREM 2.1. *For trees t, t' the following are equivalent*:

(1) *The equation $t \approx t'$ is satisfied in K^t.*

(2) *For every K-accessible leaf α in t with label v such that $E_K(t, \alpha) \not\vdash^c v \approx \perp$ there is a (necessarily unique) accessible leaf α' in t' with label v' such that $E_K(t, \alpha) \vdash^c E(t', \alpha') \cup \{v \approx v'\}$ and vice versa.*

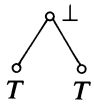
(3) $E(K) \cup AX_\Sigma \vdash t \approx t'$.

Proof.

(1) $\Rightarrow$ (2). Suppose $t \approx t'$ is satisfied in every K^t. Let α be a K-accessible leaf in t with label v such that $E_K(t, \alpha) \not\vdash^c v \approx \perp$, and let θ be the congruence on $F_\Sigma X$ generated by $E_K(t, \alpha)$. Since α is K-accessible, none of (F, T), $(T, \perp)$, $(F, \perp)$ belongs to θ, and hence $\perp, T, F$ are all distinct in $F_\Sigma X/\theta$. Moreover, since $E(K) \subseteq \theta$ it follows that $F_\Sigma X/\theta \in K$. Let $g: F_\Sigma X \to F_\Sigma X/\theta$ be the quotient homomorphism.

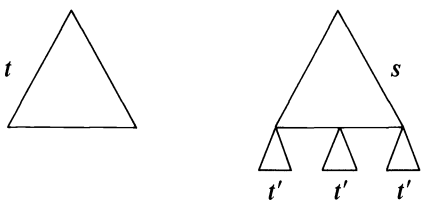
Define A to be the Σ-test algebra whose underlying Σ-algebra is $F_\Sigma X/\theta$, and let $h: F_{\Sigma'} X \to A$ be the unique homomorphism with $h(x) = g(x)$ for all $x \in X$, then $h(s) = g(s)$ for all $s \in F_\Sigma X$. Since $h(t) = h(t') = g(v)$, and $g(v) \neq \perp$, it follows that there is an accessible leaf α' in t' with label v' such that $g(v') = g(v)$ and the equations in $E(t', \alpha')$ are satisfied in A, and hence $E_K(t, \alpha) \vdash^c E(t', \alpha')$.

(2) $\Rightarrow$ (3). We may assume without loss of generality that t has no accessible leaf α with label v such that $E_K(t, \alpha) \vdash^c v \approx \perp$, since by Proposition 2.3 and (6) we may replace such a leaf with



Of course, we may also assume the same for t'.

Let s be the tree obtained from t by replacing each leaf with a copy of t'.



Then $AX_\Sigma \cup E(K) \vdash t \approx s$:

Consider a leaf α in t.

Case 1. α is inaccessible.

If $E_K(t, \alpha) \vdash^c T \approx F$ then by Proposition 2.3,

$$E(K) \cup AX_\Sigma \vdash t \approx t(\alpha \to [T, t_\alpha, t']) \approx t(\alpha \to [F, t_\alpha, t'])$$
$$\approx t(\alpha \to t') \quad \text{by equation (3).}$$

If $E_K(t, \alpha) \vdash T \approx \bot$ then we have

$$\begin{aligned} E(K) \cup AX_\Sigma \vdash t &\approx t(\alpha \to [T, t_\alpha, t']) && \text{by (1)} \\ &\approx t(\alpha \to [\bot, t_\alpha, t']) && \text{by Proposition 2.3} \\ &\approx t(\alpha \to [\bot, t', t']) && \text{by (5)} \\ &\approx t(\alpha \to [T, t', t']) && \text{by Proposition 2.3} \\ &\approx t(\alpha \to t') && \text{by (1).} \end{aligned}$$

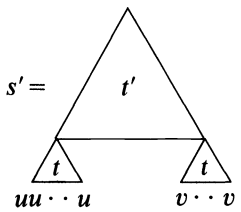
The case $E(K) \cup AX_\Sigma \vdash F \approx \bot$ is handled similarly.

Case 2. α is accessible with label v, and $E_K(t, \alpha) \nvdash^c v \approx \bot$. Then by hypothesis there is a leaf α' in t' with label v' such that $E_K(t, \alpha) \vdash E(t', \alpha') \cup \{v \approx v'\}$. Suppose the first equation in $E(t', \alpha')$ (i.e. the one appearing at the root of t') is $x \approx T$. Then the root of t' is labelled x, and

$$\begin{aligned} E(K) \cup AX_\Sigma \vdash t &\approx t(\alpha \to [x, t_\alpha, t_F]) && \text{by Propositions 2.1 and 2.3} \\ &= t(\alpha \to [x, v, t_F]) \\ &= t(\alpha \to [x, v', t_F]) && \text{by Proposition 2.3.} \end{aligned}$$

Thus we have "pushed" the label v' down one level, and at the same time added the correct branch on the other side of the root. Continue in this way until we reach α', to obtain $t \approx t(\alpha \to t')$.

Now, apply the "interchange" (7) to s to obtain $s \approx s'$ where s' is the tree obtained from t' by replacing each leaf labelled u with a copy of t with each leaf relabelled u.



An argument similar to the one above showing that $t \approx s$ establishes that $s' \approx t'$ and hence $AX_\Sigma \cup E(K) \vdash t \approx t'$.

(3) $\Rightarrow$ (1) is trivial, since every Σ-test algebra evidently satisfies all the equations in AX_Σ.

COROLLARY 1. *$E(K) \cup AX_\Sigma$ is an equational basis for K^t.*

COROLLARY 2. *If the word problem for K is decidable then the equational theory of K^t is decidable.*

Proof. The word problem for K is decidable precisely (see Evans [6]) if there is an algorithm to determine, given a finite set E of Σ-equations and an equation $s \approx t$, whether $E \cup E(K) \vdash^c s \approx t$, and hence the above theorem yields an algorithm to determine, given two trees t, t', whether or not $t \approx t'$ holds in every algebra in K^t. Since

there is an effective procedure which, given *any* term $t \in F_{\Sigma'}X$, yields a simple term s such that $E(K) \cup AX_{\Sigma} \vdash s \approx t$, this then yields an algorithm to determine, for any terms $t, t' \in F_{\Sigma'}X$, whether or not every algebra in K^t satisfies $t \approx t'$.

Remark. We do not really need the full decidability of the word problem for K, since the equations in our sets E are always of a special type, namely, the right-hand side of each equation is either T or F. We will see that this is a substantial weakening. Let K be the equational class with one binary operation $*$ and constants T, F, and $\perp$ defined by the following equations:

$$(x*y)*z \approx x*(y*z),$$
$$x*T \approx T*x \approx x*F \approx F*x \approx \perp*x \approx x*\perp \approx \perp.$$

Hence K is an equational class of semigroups with a zero. If S is any semigroup then define $S^{\perp}$ to be the algebra on $S \cup \{T, F, \perp\}$ where

$$x*y = \begin{cases} xy & \text{if } x, y \in S, \\ \perp & \text{otherwise.} \end{cases}$$

It is routine to check that $S^{\perp}$ is a K-algebra. Further, if P is a presentation of S in the equational class of semigroups, then P is a presentation of $S^{\perp}$ in K. Since the equational class of semigroups has an undecidable word problem (in fact there is a finitely presented semigroup with an undecidable word problem, see [10]), K has an undecidable word problem.

Call a set E of equations of the type of the class K, *special* iff the right side of each equation is either T or F. We say that E is *consistent* iff none of $E \cup E(K) \vdash^c T \approx F$, $E \cup E(K) \vdash^c T \approx \perp$, $E \cup E(K) \vdash^c F \approx \perp$ hold. To show that K^t has a decidable equational theory, it suffices to prove the following proposition.

PROPOSITION 2.4. *There is an effective procedure which, given a finite special set E of equations, determines whether or not E is consistent, and in the former case gives a decision procedure for the question as to whether or not $E \cup E(K) \vdash^c s \approx t$.*

Proof. If one of $(\perp, T)$, $(\perp, F)$, (T, F), (F, T) belongs to E, then we know that E is inconsistent. If (v, G) is in E, and v has T, F, or $\perp$ as a subterm, then again E is inconsistent. Finally if (u_1, G_1) and (u_2, G_2) belong to E and u_1 is a proper subterm of u_2, then $E \cup E(K) \vdash^c u_2 \approx \perp \approx G_2$. Hence E is inconsistent. Also E is inconsistent if for some u, both (u, T) and (u, F) belong to E.

Suppose, now, that none of the above possibilities occur. For simplicity assume that (F, F) and (T, T) belong to E. We associate to any term v, a term $\sigma(v)$ as follows. If v is T, F or $\perp$ then $\sigma(v) = v$. If (v, G) is in E, then $\sigma(v) = G$. If there exists (u, G) in E such that u is a proper subterm of v or $\perp$ is a subterm of v, then let $\sigma(v) = \perp$. Otherwise let $\sigma(v) = v$. Define a relation $\equiv$ by $v \equiv u$ iff $\sigma(v) = \sigma(u)$. Obviously $\equiv$ is an equivalence relation which is contained in the congruence generated by E. It remains to show that $\equiv$ is a congruence relation. Then since E is contained in $\equiv$, we will know that $\equiv$ is the congruence generated by E. Suppose $u_1 \equiv u_2$ and $v_1 \equiv v_2$. If $u_1 = u_2$ and $v_1 = v_2$, then $u_1 * v_1 = u_2 * v_2$ and so $u_1 * v_1 \equiv u_2 * v_2$. Suppose now that $u_1 \neq u_2$ or $v_1 \neq v_2$. Then examining all the cases reveals that $\sigma(u_1 * v_1) = \perp = \sigma(u_2 * v_2)$.

COROLLARY 2.5. *For K as defined above, the equational theory of K^t is decidable.*

Proof. This is a direct consequence of Proposition 2.4 and Theorem 2.1.

Remark 1. The equational theory of K^t may be undecidable, even if the equational theory of K is decidable. For example, let Σ consist of a binary operation (denoted by juxtaposition), a unary operation $^{-1}$, and constants $a, b, T, F, \perp$. Let K be the

equational class defined by the following axioms:

$$x(yz) = (xy)z,$$
$$(xy)^{-1} = y^{-1}x^{-1},$$
$$x\perp = \perp x = \perp,$$
$$aa^{-1} = a^{-1}a = bb^{-1} = b^{-1}b = TT = T,$$
$$aT = Ta = a,$$
$$a^{-1}T = Ta^{-1} = a^{-1},$$
$$bT = Tb = b,$$
$$b^{-1}T = Tb^{-1} = b^{-1},$$
$$\perp^{-1} = \perp.$$

For any K-algebra A and $x \in A$, and any integer $n \geqq 0$, define x^n inductively by: $x^0 = T$, $x^{n+1} = x(x^n)$, and define $x^{-n} = (x^n)^{-1}$. Then, in any free K-algebra, the multiplicative closure of $\{a^m \mid m \in Z\} \cup \{b^m \mid m \in Z\}$ is (isomorphic to) the free group on two generators a, b, with unit T. Using the solvability of the equational theory of groups, it can be shown that each Σ-term has a normal form modulo the equations of K, and hence that the equational theory of K is decidable. Nevertheless, the equational theory of K^t is not decidable, since that would give an algorithm to determine, given any $t_1, \cdots, t_{n+1} \in \{a^m b^n \mid m, n \in Z\}$, whether $\{t_i \approx T \mid 1 \leqq i \leqq n\} \cup E(K) \vdash^c t_{n+1} \approx T$, thereby solving the word problem for two-generated groups, which is known to be undecidable (see Boone [2]).

Remark 2. Benoit [1] claims to prove that $AX_\Sigma \cup E(K)$ is an equational basis for K^t, but as mentioned in the introduction, his purported proof is incorrect. Benoit associates to each Σ^t-term t a Σ-term $s_E(t)$, called the "skeleton" of t; his method to prove the result is by proving (1) $\Leftrightarrow$ (2), (2) $\Leftrightarrow$ (3), and (3) $\Leftrightarrow$ (4) for the following four conditions:

(1) $AX_\Sigma \cup E(K) \vdash t \approx t'$.

(2) $AX \vdash s_E(t) \approx s_E(t')$.

(3) The equation $s_E(t) \approx s_E(t')$ holds in every test algebra.

(4) The equation $t \approx t'$ holds in every algebra in K^t.

Now, (2) $\Leftrightarrow$ (3) follows, as Benoit points out, from the results of Bloom and Tindell. However, consider the case that Σ has a unary operation λ and K is the class of all strict Σ-algebras. Let $t = [\lambda(T), [\lambda(\lambda(T)), u, v], w]$ and $t' = [\lambda(T), [\lambda(T), u, v], w]$. Note that $t \approx t'$ in every algebra in K^t. However, by definition, $s_E(t) = [k_1, [k_2, u, v], w]$ and $s_E(t') = [k_1, [k_1, u, v], w]$, and hence the equation $s_E(t) \approx s_E(t')$ does not hold in every test algebra. Thus (3) and (4) are not equivalent. Also, since our Theorem 2.1 proves that (1) and (4) *are* equivalent, it follows that, for the same example, conditions (1) and (2) are not equivalent either.

3. Discriminator algebras. In this section we consider algebras with a quaternary operation $[x, y, u, v]$ instead of the ternary "test" of the previous sections; such an algebra is called a *discriminator algebra* iff it satisfies

$$[x, y, u, v] = \begin{cases} u & \text{if } x = y, \\ v & \text{if } x \neq y. \end{cases}$$

Our aim is to give, for any equational class K of any type, an equational basis for the class K^d obtained by adding to each algebra A in K a new quaternary operation which makes A a discriminator algebra.

As mentioned in the introduction, equational bases for K^d were given both in McKenzie [8] and Bloom and Tindell [3]. We present an alternative approach, modelled after the one in the preceding section, which will not only give easy proofs of decidability results, but will lend itself to a modification in the next section which handles the case of a discriminator in the presence of $\perp$.

Let Σ be any type of algebras, and let Σ^d be Σ with a new quaternary operation $[\ ,\ ,\ ,\ ]$ added. We define AX_D to be the following set of equations (note that these are some, but not all, of the Bloom-Tindell equations):

(D1) $[x, x, u, v] \approx u$,
(D2) $[x, y, u, u] \approx u$,
(D3) $[x, y, u, v] \approx [y, x, u, v]$,
(D4) $[x, y, x, u] \approx [x, y, y, u]$,
(D5) $[x, y, [u, v, a, b], [u, v, z, w]] \approx [u, v, [x, y, a, z], [x, y, b, w]]$,
(D6) $[x, y, [x, y, u, v], w] \approx [x, y, u, w]$,
(D7) $[x, y, u, [x, y, v, w]] = [x, y, u, w]$,
(D8) $[[x, y, u, v], z, a, b] \approx [x, y, [u, z, a, b], [v, z, a, b]]$.

Further, let $AX_{D\Sigma}$ be AX_D with the following identities added:

For each $\sigma \in \Sigma$ of arity n and each $i \leqq n$,

$$(D^*) \qquad \sigma(x_1, \cdots, x_{i-1}, [u, v, a, b], \cdots, x_n) \approx [u, v, \sigma(x_1, \cdots, a, \cdots, x_n), \sigma(x_1, \cdots, b, \cdots, x_n)].$$

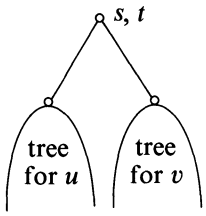
Note that (D5)-(D8) are the exact analogues of (7)-(10) of the previous sections, and that (D*) is the analogue of (*).

We define the notion of *simple term* in $F_{\Sigma^d}X$ just as in § 2: the set S of simple terms is the smallest subset of $F_{\Sigma^d}X$ such that

(i) $F_\Sigma X \subseteq S$,
(ii) If $s, t \in F_\Sigma X$ and $u, v \in S$, then $[s, t, u, v] \in S$.

As before, there is an effective procedure, using equations (D8) and (D*) which, given any term $t \in F_{\Sigma^d}X$, produces a simple term s such that $AX_{D\Sigma} \vdash s \approx t$.

The simple terms can again be represented by binary trees, where the leaves are labelled with elements of $F_\Sigma X$ and the nonleaf vertices are labelled with pairs of elements of $F_\Sigma X$, so that the tree corresponding to the simple term $[s, t, u, v]$ is



The "premise" is now the question "is $s = t$ or $s \neq t$?" rather than "is $s = T$ or $s = F$?" Therefore $E(t, \alpha)$, the "equations" on the path in the tree to the vertex α will now be a set of equations and negations of equations. Let $E^+(t, \alpha)$ be the set of equations occurring in $E(t, \alpha)$ and let $E^-(t, \alpha)$ be the set of equations whose negations occur in $E(t, \alpha)$. We note that $E^+(t, \alpha)$ will play a role analogous to that played by the equations $s = T$ in the previous of $E(t, \alpha)$, and $E^-(t, \alpha)$ will correspond to the equations $s = F$.

Note that the following results, namely Lemma 3.1, and Propositions 3.1, 3.2 and its corollaries, and 3.3 do not use (D1) and (D2); this fact will be needed in the next section.

LEMMA 3.1.

$$AX_D \vdash [x_1, x_2, [y_1, y_2, u, v], z] \approx [x_1, x_2, [y_1, y_2, [x_1, x_2, u, w], v], z],$$
$$[x_1, x_2, [y_1, y_2, u, v], z] \approx [x_1, x_2, [y_1, y_2, u, [x_1, x_2, v, w]], z],$$
$$[x_1, x_2, z, [y_1, y_2, u, v]] \approx [x_1, x_2, z, [y_1, y_2, [x_1, x_2, w, u], v]]$$

and

$$[x_1, x_2, z, [y_1, y_2, u, v]] \approx [x_1, x_2, z, [y_1, y_2, u, [x_1, x_2, w, v]]].$$

Proof. The proof uses only (D5)-(D8). We present the proof of the last equation; the others are similar, and are, of course, exactly analogous to the proof of Lemma 2.1 as in Bloom and Tindell [3].

We have

$$AX_D \vdash [x_1, x_2, z, [y_1, y_2, u, v]]$$
$$\approx [x_1, x_2, z, [x_1, x_2, [y_1, y_2, u, w], [y_1, y_2, u, v]]] \quad \text{by (D7)}$$
$$\approx [x_1, x_2, z, [y_1, y_2, [x_1, x_2, u, u], [x_1, x_2, w, v]]] \quad \text{by (D5)}$$
$$\approx [x_1, x_2, z, [y_1, y_2, [x_1, x_2, u, u], [x_1, x_2, w, [x_1, x_2, w, v]]]] \quad \text{by (D7)}$$
$$\approx [x_1, x_2, z, [x_1, x_2, [y_1, y_2, u, w], [y_1, y_2, u, [x_1, x_2, w, v]]]] \quad \text{by (D5)}$$
$$\approx [x_1, x_2, z, [y_1, y_2, u, [x_1, x_2, w, v]]] \quad \text{by (D7).}$$

PROPOSITION 3.1. *For any tree t and vertex α of t, if $x \approx y \in E^+(t, \alpha)$ then $AX_D \vdash t \approx t(\alpha \to [x, y, t_\alpha, z])$, and if $x \approx y \in E^-(t, \alpha)$ then $AX_D \vdash t \approx t(\alpha \to [x, y, z, t_\alpha])$.*

Proof. The proof is exactly analogous to the proof of Proposition 2.1, and uses Lemma 3.1 instead of Lemma 2.1.

The proofs of the following proposition and its corollaries are exactly analogous to the proof of Proposition 2.2 and its corollaries.

PROPOSITION 3.2. *If p is any term in $F_\Sigma X$ which contains the variable $x \in X$ then $AX_{D\Sigma} \vdash p(x \to [y, z, u, v]) \approx [y, z, p(x \to u), p(x \to v)]$.*

COROLLARY 1. *Under the same hypothesis on p and x,*

$$AX_{D\Sigma} \vdash [y, z, p(x \to y), w] \approx [y, z, p(x \to z), w].$$

COROLLARY 2. *Under the same hypothesis on p and x,*

$$AX_{D\Sigma} \vdash [y, z, [p(x \to y), u, v, w], b] \approx [y, z, [p(x \to z), u, v, w], b].$$

DEFINITION. For any equational class K of Σ-algebras, let $E(K)$ be all equations satisfied by K. For a tree t representing a simple term in $F_{\Sigma^d}X$ and a vertex α of t, we define $E_K^+(t, \alpha) = E^+(t, \alpha) \cup E(K)$. Further, K^d is the class of all Σ-discriminator algebras whose underlying Σ-algebra is in K.

PROPOSITION 3.3. *For a tree t and a vertex α such that one part of the label at α is u, if $E_K^+(t, \alpha) \vdash^c u \approx v$ then $AX_{D\Sigma} \cup E(K) \vdash t \approx t'$, where t' is obtained from t by changing the "u" at α to "v".*

Proof. The proof is the same as the proof of Proposition 2.3, using the above corollaries instead of the corollaries to Proposition 2.1.

Now, we define a set E consisting of equations and negations of equations to be *inconsistent* iff $E^+ \vdash^c s \approx t$ for some $(s, t) \in E^-$; otherwise E is called *consistent.* A vertex α in a tree t is called *K-accessible* (*K-inaccessible* respectively) iff $E(t, \alpha) \cup E(K)$ is consistent (inconsistent, respectively).

THEOREM 3.1. *For trees t, t' the following are equivalent:*

(1) $t \approx t'$ *holds in* K^d.

(2) *For all leaves* α, α' *in* t, t' *respectively, with labels* u, u', *if* $E_K(t, \alpha) \cup E_K(t', \alpha')$ *is consistent then* $E_K^+(t, \alpha) \cup E_K^+(t', \alpha') \vdash^c u \approx u'$.

(3) $E(K) \cup AX_{D\Sigma} \vdash t \approx t'$.

Proof.

(1) $\Rightarrow$ (2). Suppose $E_K(t, \alpha) \cup E_K(t', \alpha')$ is consistent. Let θ be the congruence on $F_\Sigma X$ generated by $E_K^+(t, \alpha) \cup E_K^+(t', \alpha')$, and $g: F_\Sigma X \to F_\Sigma X/\theta$ be the quotient map. Let A be the Σ-discriminator algebra whose underlying Σ-algebra is $F_\Sigma X/\theta$, and define $h: F_{\Sigma^d} X \to A$ to be the unique homomorphism with $h(x) = g(x)$ for all $x \in X$, then $h(s) = g(s)$ for all $s \in F_\Sigma X$. It follows from the consistency of $E_K(t, \alpha) \cup E_K(t', \alpha')$ that $h(v) \neq h(w)$ for any $(v, w) \in E_K^-(t, \alpha) \cup E_K^-(t', \alpha')$, and hence $h(t) = u$, $h(t') = u'$, which implies that $E_K^+(t, \alpha) \cup E_K^+(t', \alpha') \vdash^c u \approx u'$.

(2) $\Rightarrow$ (3). Let s be the tree obtained from t by replacing each leaf with a copy of t'; we will first prove that $E(K) \cup AX_{D\Sigma} \vdash t \approx s$.

Note first that if β is any inaccessible leaf of a tree r, $E(K) \cup AX_{D\Sigma} \vdash r = r(\beta \to x)$ for any term x: since β is inaccessible there exists $(v, w) \in E^-(r, \beta)$ such that $E_K^+(r, \beta) \vdash^c v \approx w$. Now, by (D1), we have

$$\begin{aligned} r &\approx r(\beta \to [v, v, u, x]) && \text{where } u \text{ is the label at } \beta \\ &\approx r(\beta \to [v, w, u, x]) && \text{by Proposition 3.3} \\ &\approx r(\beta \to x) && \text{by Proposition 3.1 (second part).} \end{aligned}$$

Now, suppose α is any leaf of t, and let t'/u be the tree obtained from t' by relabelling all leaves with u, the label of α in t. It follows from (D2) that $u \approx t'/u$, and hence $AX_{\Sigma D} \cup E(K) \vdash t \approx s'$, where s' is obtained from t by replacing each leaf labelled u with t'/u. Now, use hypothesis (2) and Proposition 3.3 to change the label at an accessible leaf in s' to its label in s, and use the above remark about inaccessible leaves to change the label at each inaccessible leaf in s' to its label in s, to obtain $AX_{\Sigma D} \cup E(K) \vdash t \approx s$.

Now apply the interchange equation (D5) to obtain that $s \approx r$, where r is the tree obtained from t' by replacing each leaf labelled v with a copy of t which has all leaves relabelled v; the above argument then shows that $AX_{D\Sigma} \cup E(K) \vdash r \approx t'$, implying the required result.

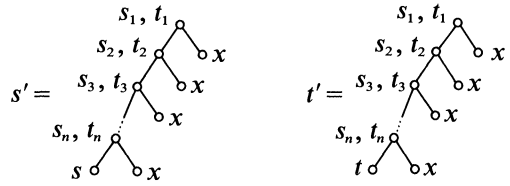
(3) $\Rightarrow$ (1) is trivial.

COROLLARY 1. $E(K) \cup AX_{D\Sigma}$ *is an equational basis for* K^d.

COROLLARY 2. *The word problem for K is decidable iff the equational theory of K^d is decidable.*

Proof. As in the preceding section, if the word problem for finitely presented algebras in K is decidable, then the above result gives an algorithm to determine whether an arbitrary equation $t \approx t'$ holds in K^d.

Conversely, if there is an algorithm to decide whether $t \approx t'$ holds in K^d, then for any finite set E of Σ-equations, say $E = \{(s_i, t_i) \mid 1 \leqq i \leqq n\}$, to determine whether $E \cup E(K) \vdash^c s \approx t$, consider the trees



Then $s' \approx t'$ in K^d iff $E \cup E(K) \vdash^c s \approx t$.

Remark. As mentioned in the introduction, the preceding corollary has been known for some time (see the introduction of [9], and is a consequence of Lemma 9.2 of [5]). Moreover, the analogous decidability result for test algebras, namely Corollary 2 to Theorem 2.1, is a consequence of this one. The test operation of the last section can be obtained as a polynomial in the quaternary discriminator and the constants $\perp$, T, F.

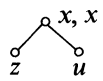
4. Modified discriminator algebras. In this section we discuss a modification of the algebras in the preceding section to allow for the presence of a constant $\perp$. An algebra A with a quaternary operation $[\ ,\ ,\ ,\ ]$ and a constant $\perp$ is called a *modified discriminator algebra* iff it satisfies

$$[a, b, c, d] = \begin{cases} \perp & \text{if } a = \perp \text{ or } b = 1, \\ c & \text{if } a = b \neq \perp, \\ d & \text{if } \perp \neq a \neq b \neq \perp. \end{cases}$$

For any type Σ containing $\perp$, we let Σ^d be the type obtained by adding the quaternary operation $[\ ,\ ,\ ,\ ]$. As in § 2, we will consider only strict Σ-algebras.

Our aim here is to give, for any equational classs K of Σ-algebras, an equational basis for the class $K^{d\perp}$ obtained by adding to each algebra A in K a new quaternary operation which makes A a modified discriminator algebra.

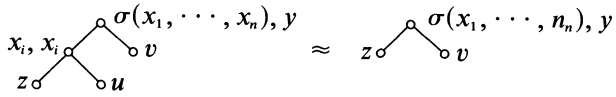
Although the treatment here parallels that of the previous sections, there is one new feature to incorporate: we will need criteria, when evaluating a tree, to know when



may be replaced by z, i.e. to know, when we reach a vertex with label x, whether the equations and negations of equations on the branch leading to this vertex already ensure that $x \neq \perp$. Thus, for example, we will have equations such as

$$[x, y, [x, x, z, u], v] \approx [x, y, z, v],$$

and, since the operations $\sigma \in \Sigma$ are strict,



Let $AX_{D_\perp}$ be the following set of axioms:

$D_\perp 1$ $[\perp, y, u, v] \approx \perp$,

$D_\perp 2$ $[x, y, \perp, \perp] \approx \perp$,

$D_\perp 2 = D3$ $[x, y, u, v] \approx [y, x, u, v]$,

$D_\perp 4 = D4$ $[x, y, x, u] \approx [x, y, y, u]$,

$D_\perp 5 = D5$ $[x, y, [x, y, u, v], w] \approx [x, y, u, w]$,

$D_\perp 6 = D6$ $[x, y, u, [x, y, v, w]] \approx [x, y, u, w]$,

$D_\perp 7 = D7$ $[x, y, [u, v, a, b], [u, v, z, w]] \approx [u, v, [x, y, a, z], [x, y, b, w]]$,

$D_\perp 8 = D8$ $[[x, y, u, v], z, a, b] \approx [x, y, [u, z, a, b], [v, z, a, b]]$,

$D_\perp 9$ $[x, y, u, u] \approx [x, x, [y, y, u, v], w]$,

$D_\perp 10$ $[x, y, u, v] \approx [x, y, u, [x, x, v, w]]$,

$D_\perp 11$ $[x, x, x, x] \approx x$.

Further, let $AX_{D_{\perp\Sigma}}$ be $AX_{D_\perp}$ with the following axioms adjoined: For each $\sigma \in \Sigma$ of arity n and each $i \leqq n$

$$D_\perp^* = D^* \quad \sigma(x_1, \cdots, x_{i-1}, [u, v, a, b], \cdots, x_n) \approx [u, v, \sigma(x_1, \cdots, a, \cdots, x_n), \sigma(x_1, \cdots, b, \cdots x_n)],$$

$$D_\perp^{**} \quad [\sigma(x_1, \cdots, x_n), y, [x_i, x_i, z, u], v] \approx [\sigma(x_1, \cdots, x_n), y, z, v],$$

$$D_\perp^{***} \quad [\sigma(x_1, \cdots, x_n), y, u, [x_i, x_i, z, v]] \approx [\sigma(x_1, \cdots, x_n), y, u, z].$$

Simple terms, binary trees and the $E(t, \alpha)$ are defined just as in the preceding section.

Moreover, since the proofs of Lemma 3.1, Propositions 3.1, 3.2 (and its corollaries) and 3.3 did not use (D1) and (D2), these results are also true, with the same proofs, for the setting described in this section. Also, we have the following additional result.

PROPOSITION 4.1. *For a tree t with vertex α, if $x \approx y \in E^-(t, \alpha)$ then*

$$AX_{D_\perp} \vdash t \approx t(\alpha \to [x, x, t_\alpha, z]).$$

Proof.

$$AX_{D_\perp} \vdash t \approx t(\alpha \to [x, y, w, t_\alpha]) \quad \text{by Proposition 3.1}$$
$$\approx t(\alpha \to [x, y, w, [x, x, t_\alpha, z]]) \quad \text{by } (D_\perp 10)$$
$$\approx t(\alpha \to [x, x, t_\alpha, z]) \quad \text{again by Proposition 3.1.}$$

PROPOSITION 4.2. *If u is a subterm of v then*

$$AX_{D_\perp} \vdash [v, v, [u, u, z, x], y] \approx [v, v, z, y].$$

Proof. The proof is by induction on the complexity of v. We may assume $u \neq v$. If $v = \sigma(v_1, \cdots, v_n)$ for $\sigma \in \Sigma$ and $u = v_i$ for some i, then the result is a consequence of axiom $D_\perp^{**}$.

If $v = \sigma(v_1, \cdots, v_n)$ and u is a subterm of v_i then

$$AX_{D_\perp} \vdash [v, v, z, y] \approx [v, v, [v_i, v_i, z, x], y] \quad \text{by the preceding argument}$$
$$\approx [v, v, [v_i, v_i, [u, u, z, x], x], y] \quad \text{by induction hypothesis}$$
$$\approx [v, v, [u, u, z, x], y] \quad \text{again by preceding argument.}$$

Now, for a set E of Σ-equations and negations of Σ-equations, we say that E is *K-inconsistent* iff either there exists $(s, t) \in E^-$ such that $E(K) \cup E^+ \vdash^c s \approx t$ or there exists $(s, t) \in E^+ \cup E^-$ such that $E(K) \cup E^+ \vdash^c s \approx \perp$. *$K$-consistent* is the negation of K-inconsistent.

A leaf α of a tree t is called *K-inaccessible* if $E(t, \alpha)$ is K-inconsistent, and *K-accessible* otherwise.

THEOREM 4.1. *For trees t, t' the following are equivalent*:

(1) *$t \approx t'$ holds in $K^{d\perp}$.*

(2) *For all leaves α and α' in t and t' respectively, labelled u and u' respectively, such that $E(t, \alpha) \cup E(t', \alpha')$ is K-consistent,*

(i) $E_K^+(t, \alpha) \cup E_K^+(t', \alpha') \vdash^c u \approx u'$

(ii) *If* $x \approx y$ *or* $x \not\approx y \in E(t', \alpha')$, *and occurs at vertex* γ' *then there is a vertex* γ, *on the path in* t *from the root to* α, *with label* (z, w), *such that*

$$E_K^+(t, \alpha) \cup E_K^+(t', \gamma') \cup \{x \approx \bot\} \vdash^c z \approx \bot \quad \text{or } u \approx \bot,$$

and vice versa.

(3) $AX_{D_\bot\Sigma} \vdash t \approx t'$.

Proof. (1) $\Rightarrow$ (2). Suppose $t \approx t'$ holds in $K^{d\bot}$, and suppose α and α' are leaves in t and t' respectively, labelled u and u' respectively, such that $E(t, \alpha) \cup E(t', \alpha')$ is K-consistent. Let θ be the congruence on $F_\Sigma X$ generated by $E_K^+(t, \alpha) \cup E_K^+(t', \alpha')$, and let $g: F_\Sigma X \to F_\Sigma X/\theta$ be the quotient map. Since $E(K) \subseteq \theta$ it follows that $F_\Sigma X/\theta \in K$.

Define A to be the modified Σ-discriminator algebra whose underlying Σ-algebra is $F_\Sigma X/\theta$, and define $h: F_{\Sigma^a} X \to A$ to be the unique homomorphism such that $h(x) = g(x)$ for $x \in X$; then $h(s) = g(s)$ for all $s \in F_\Sigma X$.

Moreover, since $E(t, \alpha) \cup E(t', \alpha')$ is K-consistent, it follows that $h(t) = u$ and $h(t') = u'$ and hence $(u, u') \in \theta$ which establishes 2(i).

Now suppose x is one of the labels occurring on the path in t' from the root to α', say at vertex γ'. Let θ' be the congruence on $F_\Sigma X$ generated by $E_K^+(t, \alpha) \cup E_K^+(t', \gamma') \cup \{x \approx \bot\}$, and let A be the modified Σ-discriminator algebra on $F_\Sigma X/\theta'$. Define h as before, then $h(t') = \bot$ and hence $h(t) = \bot$, and this yields 2(ii).

(2) $\Rightarrow$ (3). The proof is analogous to the proof in the last section, with some added details to account for the presence of $\bot$.

First of all, note that if r is any tree and β is an inaccessible leaf of r then $AX_{D_\bot\Sigma} \cup E(K) \vdash r \approx r[\beta \to x]$ for any term x: If there exists $(v, w) \in E^-(r, \beta)$ with $E_K^+(r, \beta) \vdash^c v \approx w$ then

$$\begin{aligned} AX_{D_\bot\Sigma} \cup E(K) \vdash r &\approx r(\beta \to [v, v, t_\beta, x]) && \text{by Proposition 4.1} \\ &\approx r(\beta \to [v, w, t_\beta, x]) && \text{by Proposition 3.3} \\ &\approx r(\beta \to x) && \text{by Proposition 3.1.} \end{aligned}$$

If there exists $(s, t) \in E^+(r, \beta) \cup E^-(r, \beta)$ such that $E_K^+(r, \beta) \vdash s \approx \bot$, then

$$\begin{aligned} AX_{D_\bot\Sigma} \cup E(K) \vdash r &\approx r(\beta \to [s, s, t_\alpha, x]) && \text{by either Proposition 4.1 or Propositions 3.1 and 3.3} \\ &\approx r(\beta \to [s, \bot, t_\alpha, x]) && \text{by Proposition 3.3} \\ &\approx r(\beta \to \bot) && \text{by } D_\bot 1 \\ &\approx r(\beta \to [s, \bot, x, x]) && \text{by } D_\bot 1 \\ &\approx r(\beta \to [s, s, x, x]) && \text{by Proposition 3.3} \\ &\approx r(\beta \to x) && \text{as in the first step.} \end{aligned}$$

Next, we prove that $t \approx s'$ where s' is the tree obtained from t by replacing each leaf labelled u with a copy of t' with all leaves relabelled u. We replace one leaf at a time. If α is inaccessible, then the preceding remark implies the desired result. If α is accessible, then suppose the label at the root of t' is (v, w). Then by 2(ii) there is a vertex γ on the path from the root of t to α with label (z, y) such that $E_K^+(t, \alpha)$ $\{v \approx \bot\} \vdash^c z \approx \bot$ or $u \approx \bot$. Since we may without loss of generality replace each leaf in t labelled u with $[u, u, u, u]$ by $D_\bot 11$, we may assume that we are in the case $E_K^+(t, \alpha) \cup \{v \approx \bot\} \vdash z \approx \bot$. Since $E(t, \alpha)$ is K-consistent, $E_K^+(t, \alpha) \not\vdash^c z \approx \bot$. Thus there is a sequence of transitions $z = z_1 \to z_2 \to \cdots \to z_n = \bot$ using the pairs in $E_K^+(t, \alpha) \cup \{(v, \bot)\}$ as rewrite rules, to get from z to $\bot$, and the pair $(v, \bot)$ must get used at least

once. Thus we may assume that the first m transitions use only pairs from $E_K^+(t, \alpha)$ and that $z_m \to z_{m+1}$ replaces a subterm of z_m equal to v, by $\perp$. Note that by strictness we then have $E(K) \vdash z_{m+1} \approx \perp$. Thus

$$\begin{aligned} AX_{D_\perp\Sigma} \cup E(K) \vdash t &\approx t(\alpha \to [z, z, u, u]) && \text{either by Proposition 4.1 or Propositions 3.1 and 3.3} \\ &\approx t(\alpha \to [z_m, z_m, u, u]) && \text{by Proposition 3.3} \\ &\approx t(\alpha \to [z_m, z_m, [v, v, u, u], u]) && \text{by Proposition 4.2} \\ &\approx t(\alpha \to [v, v, u, u]) && \text{as in the first two steps.} \end{aligned}$$

Using the same argument, now applied to w, we obtain

$$\begin{aligned} AX_{D_\perp\Sigma} \cup E(K) \overset{c}{\vdash} t &\approx t(\alpha \to [v, v, [w, w, u, u], u]) \\ &\approx t(\alpha \to [v, w, u, u]) \quad \text{by } D_\perp 9. \end{aligned}$$

Thus we have added the first node of t', and pushed the label u down one level. Repeat the procedure to obtain the desired result.

Now, just as in the proof of (2) $\Rightarrow$ (3) of Theorem 3.1, we obtain $AX_{D_\perp\Sigma} \cup E(K) \vdash t \approx s$ where s is the tree obtained from t by replacing each leaf with a copy of t', and use the interchange law to prove that $s \approx r$, where r is obtained from t' by replacing each leaf labelled v with a copy of t with all leaves relabelled v, and then the above argument shows that $AX_{D_\perp\Sigma} \cup E(K) \vdash r \approx t'$ and hence $t \approx t'$.

(3) $\Rightarrow$ (1) is trivial.

COROLLARY. *$E(K) \cup AX_{D\perp}$ is an equational basis for K^d.*

Remark on decidability. As in the two preceding sections, if the word problem for K is decidable, then the equational theory of K^d is decidable. The converse is somewhat more delicate than in the case of the quaternary discriminator: we show below that if every equation of the form $s \approx \perp$ which is satisfied by K is a consequence of the strictness of the operations (i.e., $\perp$ is a subterm of s) then the converse does indeed hold.

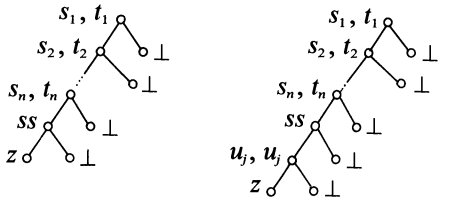
Suppose there is an algorithm to determine whether an equation $t \approx t'$ holds in $K^{d\perp}$ and suppose also that $s \approx \perp$ belongs to $E(K)$ iff $\perp$ is a subterm of s:

CLAIM 1. *There is an algorithm to determine, given any finite set E of Σ-equations and any term s, whether $E \cup E(K) \vdash^c s \approx \perp$.*

Proof. We may assume that $E = \{s_i \approx t_i \mid 1 \leqq i \leqq n\} \cup \{u_j \approx \perp \mid 1 \leqq j \leqq m\}$, where $\perp$ is not a subterm of any s_i, t_i or u_j, and that $\perp$ is not a subterm of s. Thus $\{s_i \approx t_i \mid 1 \leqq i \leqq n\} \cup E(K) \not\vdash^c s \approx \perp$, and hence if $E(K) \cup E \vdash^c s \approx \perp$ then there is a sequence of rewrites

$$s = w_0 \to w_1 \to w_2 \to \cdots \to w_k \to \perp$$

where for each i, $E \cup E(K) \vdash^1 w_i \approx w_{i+1}$. By choosing k minimal we obtain that exactly one of the rules $u_j = \perp$ is used in this sequence, and hence $E \cup E(K) \vdash^c s \approx \perp$ iff for some $j \leqq m$, $\{s_i \approx t_i \mid 1 \leqq i \leqq n\} \cup \{u_j \approx \perp\} \cup E(K) \vdash^c s \approx \perp$. However, the latter is equivalent to the following two trees being equal in $K^{d\perp}$:



CLAIM 2. *There is an algorithm to determine, given any finite set E of equations between Σ-terms, none of which contains $\perp$ as a subterm, and any terms s and t, whether $E \cup E(K) \vdash^c s \approx t$.*

Proof. $E \cup E(K) \vdash^c s \approx t$ iff the two trees s' and t' given at the end of § 3 are equal in every algebra in $K^{d\perp}$.

Now, suppose $E = \{s_i \approx t_i \mid 1 \leqq i \leqq n\} \cup \{u_j \approx \perp \mid 1 \leqq i \leqq m\}$, where $\perp$ is not a subterm of any s_i, t_i or u_j. Then $E \cup E(K) \vdash^c s \approx t$ iff either $E \cup E(K) \vdash^c s = \perp$ and $t = \perp$ or $\{s_i \approx t_i \mid 1 \leqq i \leqq n\} \cup E(K) \vdash^c s \approx t$, and so the above two claims yield an algorithm to determine whether $E \cup E(K) \vdash^c s \approx t$.

5. Further variants. There are, of course, other variants of an if-then-else operation which are amenable to the preceding methods. The last one which we shall consider, and which was also studied in [3] is the case of algebras with a ternary operation $[\ ,\ ,\ ]$ and one constant T; such an algebra is a test algebra if it satisfies

$$[x, y, z] = \begin{cases} y & \text{if } x = T, \\ z & \text{else.} \end{cases}$$

These algebras, with additional operations $\sigma \in \Sigma$, are handled just as is the case of the quaternary discriminator in § 3. An equational basis is given by

(1) $[T, x, y] \approx x$,
(2) $[x, x, y] \approx [x, T, y]$,
(3′) $[x, y, y] \approx y$,
(7) $[x, [y, u, v], [y, z, w]] \approx [y, [x, u, z], [x, v, w]]$,
(8) $[x, [x, y, u], z] \approx [x, y, z]$,
(9) $[x, z, [x, y, u]] \approx [x, z, u]$,
(10) $[[x, y, z], u, v] \approx [x, [y, u, v], [z, u, v]]$

and for each $\sigma \in \Sigma$ of arity n and each $i \leqq n$,

$$(*) \qquad \sigma(x_1, \cdots, x_{i-1}, [u, v, w], \cdots, x_n) \approx [u, \sigma(x_1, \cdots, v, \cdots, x_n), \sigma(x_1, \cdots, w, \cdots, x_n)].$$

The notions of simple terms, binary trees, $E(t, \alpha)$, K-accessibility and K-inaccessibility are defined just as in § 3, with the obvious modification because the operation is ternary rather than quaternary, so that the equations in $E^+(t, \alpha) \cup E^-(t, \alpha)$ are all of the form $u \approx T$ for some u. All of the results of § 3 carry over to this case, including the statement and proof of Theorem 3.1.

Note added in proof. Guessarian and Meseguer [*On the axiomatization of if_then_else*, this Journal, 16 (1987), pp. 332–357] deal with the problem of equational bases for test algebras, where the algebras have two sorts of elements, Bool and Data, and two tests, one mapping Bool × Bool × Bool to Bool, and one Bool × Data × Data to Data. They assume that there are precisely three Boolean elements $\perp$, T and F, in each test algebra, and that all additional operations take only Data elements as arguments. With this restriction, they obtain an equational basis for the analogue of K^t. We note that our techniques work just as well for the many-sorted case, with no restrictions on the types of additional operations, to obtain the analogues of all the results in § 2, in particular Theorem 2.1 and its corollaries.

Acknowledgments. The authors are grateful to Ralph McKenzie for several helpful comments, and to McMaster University for providing a sabbatical year for the second author.

We dedicate this paper to our mathematical predecessor, Esther Swicarcyk.

REFERENCES

[1] C. BENOIT, *Axiomatization des tests*, Theoret. Comput. Sci., 35 (1985), pp. 89–107.
[2] W. W. BOONE, *The word problem*, Ann. Math., 70 (1959), pp. 207–265.
[3] S. BLOOM AND R. TINDELL, *Varieties of "if-then-else"*, this Journal, 12 (1983), pp. 677–707.
[4] S. BURRIS AND H. P. SANKAPANNAVAR, *A Course in Universal Algebra*, Springer-Verlag, New York, Heidelberg, Berlin, 1981.
[5] S. BURRIS AND H. WERNER, *Sheaf constructions and their elementary properties*, Trans. Amer. Math. Soc., 248 (1979), pp. 269–309.
[6] T. EVANS, *The word problem for abstract algebras*, J. London Math. Soc. (2), 26 (1951), pp. 64–71.
[7] I. GUESSARIAN AND J. MESEGUER, *Axiomatization of "if-then-else" revisited*, manuscript, LITP Report 84-18, April 1984, University of Paris.
[8] R. MCKENZIE, *On spectra, and the negative solution of the decision problem for identities having a finite non-trivial model*, J. Symbolic Logic, 40 (1975), pp. 186–196.
[9] ———, *A new product of algebras and a type reduction theorem*, Algebra Universalis, 18 (1984), pp. 29–69.
[10] E. L. POST, *Recursive unsolvability of a problem of Thue*, J. Symbolic Logic, 12 (1947), pp. 1–11.
[11] H. WERNER, *Discriminator Algebras*, Akademie-Verlag, Berlin, 1978.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987

© 1987 Society for Industrial and Applied Mathematics
005

EXPECTED COMPUTATION TIME FOR HAMILTONIAN PATH PROBLEM*

YURI GUREVICH† AND SAHARON SHELAH‡

Abstract. One way to cope with an NP-hard problem is to find an algorithm that is fact on average with respect to a natural probability distribution on inputs. We consider from that point of view the Hamiltonian Path Problem. Our algorithm for the Hamiltonian Path Problem constructs or establishes the nonexistence of a Hamiltonian path. For a fixed probability p, the expected run-time of our algorithm on a random graph with n vertices and the edge probability p is $O(n)$. The algorithm is adaptable to directed graphs.

Key words. average case complexity, NP-hard Hamiltonian circuit, Hamiltonian path, probability, random graphs, expected polynomial time, expected sublinear time

AMS(MOS) subject classifications. 05G35, 60C05, 68A10, 68A20

Introduction. One way to cope with an NP-hard decision problem $\exists y R(x, y)$ is to find an algorithm that is fast on average with respect to a natural probability distribution on inputs. See in this connection Levin (1984) and Johnson (1984). A similar approach can be taken if one is interested in exhibiting an object y that witnesses $R(x, y)$, and not only in the existence of a witness. We consider from this point of view the following version of the Hamiltonian Path Problem: given a graph with distinguished vertices Start and Finish, construct or establish the nonexistence of a Hamiltonian path from Start to Finish. A Hamiltonian path from Start to Finish can be defined as a linear ordering of all vertices such that Start is the first, Finish is the last, and every pair of neighbors forms an edge. Without loss of generality, we restrict our attention to the case when the vertices of the given graph form an initial segment $\{0, 1, \cdots, n-1\}$ of natural numbers, Start $= 0$ and Finish $= n-1$.

Let $\Gamma(n, p)$ be the popular probability space of graphs with vertices $0, 1, \cdots, n-1$ and the edge probability p (Erdös and Spencer (1974)). The probability, assigned to a particular graph G with vertices $0, 1, \cdots, n-1$, is the probability that G is the outcome of the following experiment: For each pair $\{i, j\}$ of distinct vertices, toss a coin with the probability p of the head; declare $\{i, j\}$ an edge if and only if the head turns up. Let $G_{n,p}$ be a random member of $\Gamma(n, p)$.

In order to deal with instances of an arbitrary size, consider a function $p(n)$ from positive integers to the real interval $[0, 1]$. The spaces $\Gamma(n, p(n))$ can be combined into one probability space S with respect to a probability distribution δ on positive integers (the meaning of $\delta(n)$ is the probability that a random graph has exactly n vertices); however, this may not be necessary for the following reason. The expected run-time

* Received by the editors June 27, 1985; accepted for publication July 26, 1986.

† Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan 48109-2122. The work of this author was supported in part by National Science Foundation grants MCS 83-01022 and DCR 85-03275.

‡ Institute of Mathematics and Computer Science, The Hebrew University, 91904 Jerusalem, Israel. The work of this author was supported in part by National Science Foundation grant MCS 81-01560. This paper was written in the fall of 1984 when this author was a visiting professor in both the Department of Electrical Engineering and Computer Science and the Department of Mathematics, The University of Michigan, Ann Arbor, Michigan 48109-2122.

of an algorithm on a random member of S with n vertices is bounded by some function $t(n)$ if the expected run-time of the algorithm on each $G_{n,p(n)}$ is bounded by $t(n)$.

There are several algorithms for the Hamiltonian Path Problem in the literature that almost always (with probability 1 minus the reciprocal of a polynomial in n) work fast on any $G_{n,p(n)}$ where $p(n)$ is sufficiently large to ensure a high probability of the existence of a Hamiltonian path. Angluin and Valiant (1979) survey the previous algorithms of that sort and give an elegant faster algorithm of their own; almost always, their algorithm runs in time $O(n(\log n)^2)$ on their Random Access Computer. Shamir (1983) gives a further result of that sort; his algorithm is slower but the requirement on p is weaker: $p(n) \geqq n^{-1}(\log_e n + c \log_e \log_e n)$ where $c > 3$. Bollobas, Fenner and Frieze (1985) weakened the requirement on $p(n)$ to the limit; almost always, their algorithm runs in time $O(n^{4+\varepsilon})$. The expected run-time of all these algorithms is exponential with one exception, which is the Bollobas, Fenner and Frieze algorithm in the case $p(n) \geqq \frac{1}{2}$. (In the technical report of Gurevich and Shelah (1984) we wrote that, to the best of our knowledge, there are no results in the literature on expected run-time of algorithms for the Hamiltonian Path Problem. The Bollobas, Fenner and Frieze paper appeared soon after that.)

We present here our algorithm HPA (Hamiltonian Path Algorithm) for the Hamiltonian Path Problem and estimate the run-time of HPA on $G_{n,p}$ in the case when p is fixed and n is varied. The algorithm constructs or establishes the nonexistence of a Hamiltonian path from Start to Finish. The exact form of input is not important; for the sake of definiteness, we suppose that graphs are represented by the adjacency matrices. It is important that a random access computer is used. It could be, for example, the randomized Random Access Computer of Angluin and Valiant (1979).

THEOREM 1. *There is a positive real number c satisfying the following condition. For any fixed probability p, the expected run-time of* HPA *on* $G_{n,p}$ *is* $cn/p + o(n)$.

COROLLARY. *For every p there is a constant d such that the expected run-time of HPA on* $G_{n,p}$ *is bounded by* dn/p.

The algorithm HPA is composed of three algorithms HPA1, HPA2 and HPA3. First HPA1 is applied. If HPA1 fails to solve the given instance of the Hamiltonian Path Problem, then HPA2 is applied. If HPA2 fails, then HPA3 is applied; HPA3 always succeeds. Let us comment briefly on each of the three algorithms.

The algorithm HPA1 is described in § 1. It tries to construct a Hamiltonian path, and definitely fails if the desired path does not exist. For a fixed p, the expected run-time of HPA1 on $G_{n,p}$ is $cn/p + o(n)$ where c is an absolute constant. The constant c is quite a modest number; its exact value depends on the machine model. For a fixed p, the probability that HPA1 fails on $G_{n,p}$ is $2^{**}(-O(\sqrt{n}))$.

The algorithm HPA2 is described in § 2. It tries to construct or establish the nonexistence of a Hamiltonian path. The idea is to take care of the troublesome vertices (those with relatively small degree) first. The run-time of HPA2 on a graph with n vertices is $O(n^2)$. For a fixed p, the probability that HPA2 fails on HPA1($G_{n,p}$), assuming that HPA1 has failed on $G_{n,p}$, is $2^{-3n+o(n)}$ where HPA1($G_{n,p}$) is the output of HPA1 on $G_{n,p}$ in the case that HPA1 fails on $G_{n,p}$.

The fastest (in the worst case) known algorithm for the Hamiltonian Path Problem is the dynamic programming algorithm (Bellman (1960), Held and Karp (1962)) which solves the Hamiltonian Path Problem in time $2^n \cdot n^{O(1)}$ where n is the number of vertices. We could adapt it as our third algorithm HPA3. A negative feature of the dynamic programming algorithm is that it requires exponential space. Our third algorithm HPA3, described in § 3, solves the Hamiltonian Path Problem for graphs with n vertices in time $2^{2n} \times 2^{O(l)}$ and in space $O(n)$; here $l = \log_2 n$. (This time bound

is still substantially lower than the time bound for the straightforward exhaustive search.)

It is easy to see that, for a fixed probability p, the expected run-time of HPA on $G_{n,p}$ is the expected run-time of HPA1 on $G_{n,p}$ plus $o(n)$.

The following claim shows that, in a sense, our algorithm has the best possible expected run-time.

CLAIM. *Let A be any algorithm for the Hamiltonian Path Problem whose input is provided by an oracle: given a pair of vertices u, v as a query, the oracle answers whether $\{u, v\}$ is an edge. For every positive ε there is n_0 such that for every $n \geqq n_0$ and every $p > (3 \log_e n)/n$, the expected number of queries during the run of A on $G_{n,p}$ is at least $(1-\varepsilon)n/p$.*

Proof. Let Q be the number of different queries, and P be the number of different queries answered positively. The expectation $E(P)$ equals $p \times E(Q)$; hence $E(Q) = p^{-1}E(P)$. If there is a Hamiltonian path (from the designated Start to the designated Finish) then $P \geqq n-1$. Since $p > (3 \log_e n)/n$, the probability of the existence of a Hamiltonian path converges to 1 when n grows to infinity (Bollobas (1979), Posa (1976)). □

The algorithm HPA can be adapted to directed graphs. For definiteness, we work with the analogue $\Delta(n, p)$ of $\Gamma(n, p)$ which comprises directed graphs D with vertices $0, 1, \cdots, n-1$ and at most one edge between any two vertices. The probability, assigned to a particular D, is the probability that D is the result of the following experiment: Given distinct vertices u and v, toss an unbiased coin to determine which of the two ordered pairs (u, v), (v, u) is a candidate to be an edge. Having a candidate (x, y), toss a coin with the probability p of the head; declare (x, y) an edge if and only if the head turns up. Let $D_{n,p}$ be a random member of $\Delta(n, p)$. In § 4 we indicate changes necessary to turn HPA to an algorithm HPA_d for solving the Hamiltonian Path Problem for directed graphs.

THEOREM 2. *There is a positive real number c satisfying the following condition. For a fixed probability p, the expected run-time of HPA_d on $D_{n,p}$ is $cn/p + o(n)$.*

The expected run-time of HPA on a $G_{n,p(n)}$ remains polynomial if $p(n)$ changes with n but does not decrease too fast. We do not want to state explicitly the rate of decrease that leaves the expected run-time of HPA on a $G_{n,p(n)}$ polynomial: the algorithm HPA and its exposition were designed with the case of a fixed p in mind. A proper treatment of the case of decreasing $p(n)$ will require a different algorithm.

Conjecture. There is an algorithm for the Hamiltonian Path Problem with polynomial (in n) expected run-time on any $G_{n,p}$.

This paper was published first as a technical report (Gurevich and Shelah (1984)).

0. Preliminaries. Consider two coloring algorithms.

The first coloring algorithm. Given a graph do the following. For every pair $\{u, v\}$ of distinct vertices, toss an unbiased coin (which will be referred to as the red coin). If the coin turns up a head and $\{u, v\}$ is an edge, then declare $\{u, v\}$ a red edge.

The second coloring algorithm. Given a graph do the following. For every pair $\{u, v\}$ of distinct vertices, toss three unbiased coins: an orange coin, a yellow coin, and a green coin. If the orange (resp. yellow, green) coin turns up a head and $\{u, v\}$ is an edge then declare $\{u, v\}$ an orange (resp. yellow, green) edge.

The first coloring algorithm turns the given adjacency matrix M into a "red" adjacency matrix M'. For each pair $\{u, v\}$ of distinct vertices, M' contains one bit of information which tells us whether $\{u, v\}$ is a red edge or not. We introduce a new probability space $\Gamma'(n, p)$. Let G be a graph on $\{0, 1, \cdots, n-1\}$ with adjacency matrix

M and red adjacency matrix M'. The probability of G in $\Gamma'(n, p)$ is the product of the following:

The probability that M is the adjacency matrix of a random member of $\Gamma(n, p)$, and

The probability that, given M, the first coloring produces M'.

The two coloring algorithms together turn the given adjacency matrix M into an extended adjacency matrix M^*. For each pair $\{u, v\}$ of distinct vertices, M^* contains five (dependent) bits of information: the original bit saying whether $\{u, v\}$ is or is not an edge, the "red bit" saying whether $\{u, v\}$ is or is not a red edge, and three other "color bits." We introduce a new probability space $\Gamma^*(n, p)$. Members are graphs with vertices $0, 1, \cdots, n-1$ and both uncolored and colored edges; an edge may have any subset of the colors red, orange, yellow, green. (Formally, members of $\Gamma'(n, p)$ are extended adjacency matrices.) Let G be a graph on $0, 1, \cdots, n-1$ with adjacency matrix M and extended adjacency matrix M^*. The probability of G in $\Gamma^*(n, p)$ is the product of the following:

The probability that M is the adjacency matrix of a random member of $\Gamma(n, p)$, and

The probability that, given M, the two coloring algorithms produce M^*.

Given the adjacency matrix of a graph, we would like to apply the two coloring algorithms first and then to apply the rest of HPA to the resulting extended adjacency matrix. Unfortunately, coloring takes too much time. To get around this problem, HPA tosses a C-coin for a pair of different vertices when the pair is examined for being a C-edge for the first time. Logically, coloring precedes the other parts of the algorithm HPA, but operationally its work will be divided between HPA1 and HPA2: HPA1 will do some of the red coloring, and HPA2 will do all orange, yellow and green coloring.

Remark. One may alter the coloring in different ways. Since HPA1 uses only red edges to construct a Hamiltonian path and since the expected running time of HPA is defined largely by the expected running time of HPA1, one may like to increase the probability of red color. Actually, red coloring can be skipped altogether because only a small fraction of edges is examined by HPA1, but then HPA2 should be substantially modified.

DEFINITION. A *primitive boolean combination* of events $E_1, \cdots, E_m$ is the intersection of events $X_1, \cdots, X_m$ where each X_i is either the event E_i or the complement of the event E_i. An event E is *realizable* if its probability is positive.

The probability that an arbitrary pair $\{i, j\}$ of distinct vertices is a red (respectively, orange, yellow, green) edge is $p/2$.

LEMMA 0.1. *For every $k \in \{1, 2, 3\}$ there is a positive real number c_k (not depending on p or n) which satisfies the following conditions. Suppose that $J = \{$red, orange, yellow, green$\}$, $C \in J$, K is a subset of $J - \{C\}$ of cardinality k, G is a random member of $\Gamma^*(n, p)$, $\{u, v\}$ is a pair of distinct vertices of G, and E is the event "$\{u, v\}$ is an edge of color C." Then*

(a) *The probability of E, assuming any primitive boolean combination of the events "$\{u, v\}$ is an edge of color D," where $D \in K$, is at least $c_k \cdot p$;*

(b) *The probability of E, assuming any realizable boolean combination of the events "$\{u, v\}$ is an edge of color D," where $D \in K$, is at least $c_k \cdot p$;*

(c) *Let H be any realizable boolean combination of the events "$\{x, y\}$ is of color D" excluding E. Then $\mathbf{P}[E|H] \geqq c_k \cdot p$.*

Proof. (a) An easy and direct computation of conditional probabilities establishes the statement.

(b) A realizable boolean combination of some events equals a disjoint sum of primitive boolean combinations of those events. Now use the fact that if Y, Z are disjoint realizable events and $\mathbf{P}[X \mid Y] \geqq r$, $\mathbf{P}[X \mid Z] \geqq r$ then $\mathbf{P}[X \mid Y \cup Z] \geqq r$. We verify

here this simple fact.

$$\begin{aligned}\mathbf{P}[X \mid Y \cup Z] &= \mathbf{P}[X(Y \cup Z)] : \mathbf{P}[Y \cup Z] = (\mathbf{P}[XY] + \mathbf{P}[XZ]) : (\mathbf{P}[Y] + \mathbf{P}[Z]) \\ &= \mathbf{P}[XY] : (\mathbf{P}[Y \cup Z]) + \mathbf{P}[XZ] : (\mathbf{P}[Y \cup Z]) \\ &= \mathbf{P}[X \mid Y] \times \mathbf{P}[Y \mid Y \cup Z] + \mathbf{P}[X \mid Z] \times \mathbf{P}[Z \mid Y \cup Z] \geqq r.\end{aligned}$$

(c) Without loss of generality, H is a primitive boolean combination. Therefore H is the intersection AB where A is an event about $\{u, v\}$ and B is an event about other pairs of vertices. Thus,

$$\mathbf{P}[E \mid AB] = \mathbf{P}[EAB]/\mathbf{P}[AB] = (\text{by virtue of the independence}),$$

$$(\mathbf{P}[EA] \times \mathbf{P}[B])/(\mathbf{P}[A] \times \mathbf{P}[B]) = \mathbf{P}[E \mid A] \geqq c_k \cdot p. \qquad \square$$

The algorithm HPA will use the four colors in the order of spectrum. This motivates the following definition. Let c_1, c_2 and c_3 be the maximal numbers satisfying Lemma 0.1. Then $p_{\text{red}} = p/2$, $p_{\text{orange}} = c_1 \cdot p$, $p_{\text{yellow}} = c_2 \cdot p$, and $p_{\text{green}} = c_3 \cdot p$. For each color C, let $q_C = 1 - p_C$. It is easy to check that $p_{\text{red}} > p_{\text{orange}} > p_{\text{yellow}} > p_{\text{green}}$.

In the rest of this section we prove a couple of auxiliary propositions about probabilities. The first one helps to deal with families of events that are independent only to a certain degree.

PROPOSITION 0.1. *Let* $0 < r < 1$ *and* $0 \leqq \beta \leqq 1$.

(a) *Consider m Bernoulli trials with the probability r of success in a single trial. Then*

$$\mathbf{P}[\textit{at most } (1-\beta)mr \textit{ successes}] \leqq e^{**}(-\beta^2 mr/2),$$

$$\mathbf{P}[\textit{at least } (1+\beta)mp \textit{ successes}] \leqq e^{**}(-\beta^2 mr/3).$$

(b) *Let* $E_1, \cdots, E_m$ *be events in a probability space. Suppose that the probability of each* E_j, *assuming any primitive boolean combination of the events* E_j *with* $j < i$, *is at least r. Then*

$$\mathbf{P}[\textit{at most } (1-\beta)mr \textit{ events } E_i \textit{ happen}] \leqq e^{**}(-\beta^2 mr/2).$$

Proof. (a) We borrow the two inequalities from Angluin and Valiant (1979). They follow from Chernoff's bound (Chernoff (1952)).

(b) Let $D_{j,k,l}$ be the collection of sample points s such that s belongs to at most k among l events $E_{j+1}, \cdots, E_{j+1}$. In virtue of (a), it suffices to prove that for all j, k, l and H, if $1 \leqq l \leqq m$, $0 \leqq j \leqq m - l$, $0 \leqq k \leqq l$ and H is realizable then $\mathbf{P}[D_{j,k,l} \mid H] \leqq \Sigma\{b_{i,l} : 0 \leqq i \leqq k\}$ where $b_{i,l}$ is the probability of i successes in L Bernoulli trials with probability r of success in a single trial.

The proof is by induction on l. Basis of induction:

$$\mathbf{P}[D_{j,0,1} \mid H] = \mathbf{P}[\bar{E}_{j+1} \mid H] \leqq 1 - r = b_{0,1}; \qquad \mathbf{P}[D_{j,1,1} \mid H] \leqq 1 = b_{0,1} + b_{1,1}.$$

Induction step. Let $H_0 = H$, $H_{i+1} = H_i \cap \bar{E}_{j+i+1}$ for $i < l$. If $D_{j,0,l}$ is realizable then all H_i are realizable and $\mathbf{P}[D_{j,0,l} \mid H] = \Pi\{\mathbf{P}[\bar{E}_{j+i+1} \mid H_i : i < 1\} \leqq (1-r)^l = b_{0,l}$; otherwise $\mathbf{P}[D_{j,0,l} \mid H] = 0 \leqq b_{0,l}$. It remains to prove that $\mathbf{P}[D_{j,k+1,l+1} \mid H] \leqq \Sigma\{b_{i,l+1} : 0 \leqq i \leqq k+1\}$. If $\alpha = \mathbf{P}[E_{j+1} \mid H] < 1$ then H_1 is realizable and

$$\begin{aligned}\mathbf{P}[D_{j,k+1,l+1} \mid H] &= \alpha \times \mathbf{P}[D_{j+1,k,l} \mid H \cap E_{j+1}] \\ &\quad + (1-\alpha) \times \mathbf{P}[D_{j+1,k+1,l} \mid H_1] \quad (\text{by the induction hypothesis})\end{aligned}$$

$$\leqq \alpha \times \Sigma\{b_{i,l} : 0 \leqq i \leqq k\} + (1-\alpha) \times \Sigma\{b_{i,l} : 0 \leqq i \leqq k+1\}$$

(which decreases with α and therefore)

$$\leqq r \times \Sigma\{b_{i,l} : 0 \leqq i \leqq k\} + (1-r) \times \Sigma\{b_{i,l} : 0 \leqq i \leqq k+1\}$$

= (as in the case when the events E_i are truly independent)

$$\Sigma\{b_{i,l+1} : 0 \leqq i \leqq k+1\}.$$

If $\alpha = 1$ skip the second expression in the above computation. □

Note. One of the referees suggested as reference (Graham (1983)) in connection to the proof of Proposition 0.1.

The next proposition is generalization of the Bottleneck Lemma of Angluin and Valiant (1979). It can be used often as an alternative to Proposition 0.1. If S is a set of strings, let $\text{Pref}(S) = \{s' : s' \text{ is a prefix of some string } s \in S\}$. A prefix s' of a string s is *proper* if $|s'| < |s|$.

PROPOSITION 0.2. *Let*

S be a set of strings over some finite alphabet A,

k be a natural number and $0 \leqq p_1, \cdots, p_k \leqq 1$,

E be an arbitrary realizable event,

$\alpha_1, \alpha_2, \alpha_3, \cdots$ be random variables taking values in A.

Suppose that for each $s \in S$ there are proper substrings $s_1, \cdots, s_k$ of s such that $|s_1| < \cdots < |s_k|$, and for each s_i, if $l = |s_i| + 1$ then $\mathbf{P}[s_i \alpha_l \in \text{Pref}(S) \mid E] \leqq p_i$.

Then $\Sigma_{s \in S} \mathbf{P}[\alpha_1 \cdots \alpha_{|s|} = s \mid E] \leqq p_1 \bullet \cdots \bullet p_k$.

Proof. The proof follows by induction on k. The case $k = 0$ is trivial: the product of zero factors is supposed to be equal to 1.

Suppose $k > 0$ and the proposition is proved for $k-1$. For each $s \in S$ fix some appropriate substrings $s_1, \cdots, s_k$; let $T = \{s_k : s \in S\}$. If $t \in T$ then there are proper substrings $t_1, \cdots, t_{k-1}$ of t such that $|t_i| < \cdots < |t_{k-1}|$, and for each t_i, if $l = |t_i| + 1$ then $\mathbf{P}[t_i \alpha_l \in \text{Pref}(T) \mid E] \leqq p_i$. If $t \in T$ then $\mathbf{P}[t\alpha_{|t|+1} \in \text{Pref}(S) \mid E] \leqq p_k$. Thus,

$$\Sigma_{s \in S} \mathbf{P}[\alpha_1 \cdots \alpha_{|s|} = s \mid E] \leqq \Sigma_{t \in T} (\mathbf{P}[\alpha_1 \cdots \alpha_{|t|} = t \mid E] \times \mathbf{P}[t\alpha_{|t|+1} \in \text{Pref}(S) \mid E])$$

$$\leqq (\Sigma_{t \in T} \mathbf{P}[\alpha_1 \cdots \alpha_{|t|} = t \mid E]) \times p_k \leqq p_1 \bullet \cdots \bullet p_{k-1} \cdot p_k. \quad \square$$

Following Angluin and Valiant, we explain the intuition behind Proposition 0.2. Ignore event E and suppose that $p_1 = \cdots = p_k = p$. Think about $\text{Pref}(S)$ as the set of, say, violet nodes on the tree of strings over A. Say that a violet node t is a *bottleneck* if the probability of drawing a violet successor of t is at most p. The proposition says that if there are at least k bottlenecks on the way to each S-node then the probability to arrive from the root to S is at most p^k.

1. Algorithm HPA1. The algorithm HPA1, described in this section, attempts to construct an (all-red) Hamiltonian path in a given graph from one distinguished vertex Start to another distinguished vertex Finish. HPA1 does not attempt to establish the nonexistence of a Hamiltonian path.

To simplify the exposition, we assume that the given graph is colored red and the red adjacency matrix is given. The assumption will be removed at the end of the section.

The description of HPA1 is interleaved with some analysis. We fix a probability p and analyse the run of HPA1 on a random member of some $\Gamma'(n, p)$. We will prove that the expected run-time is $cn/p + o(n)$ for some absolute (i.e. not depending on p) constant c, and that, almost surely, HPA1 succeeds in constructing an all-red Hamiltonian path. In this section, an event $E(n)$ will be called *negligible* if its probability is $2^{**}(-O(/(\sqrt{n}))$, and *almost sure* if its complement $\bar{E}(n)$ is negligible.

Remark. The bound $2^{**}(-O(\sqrt{n}))$ is chosen somewhat arbitrarily. One can play with this bound. For example, $\sqrt{n}$ can be replaced by $(\log_2 n)^2$ or any n^ε with $\varepsilon < 1$; HPA1 is easily adjustable. However, the algorithm HPA1 definitely fails if G does not have a Hamiltonian path from Start to Finish. Thus the probability of failure is not less than the probability $(1-p)^{n-1}$ of a vertex to be isolated.

The algorithm HPA1 works in several stages.

Stage 0. Create four lists:
Even, consisting of only one vertex, namely the vertex Start;
Odd, consisting of only one vertex, namely the vertex Finish;
Even Outback, consisting of all remaining even vertices; and
Odd Outback, consisting of all remaining odd vertices.

To simplify exposition, we ignore rounding reals (i.e. we ignore necessary applications of the floor and ceiling functions $\lfloor \cdots \rfloor$ and $\lceil \cdots \rceil$).

Stage 1. Extend Even by means of sweeps through Even Outback. During one sweep, for each Even Outback vertex v in turn, try to extend Even by means of v: if the pair {Last (Even), v} is a red edge then remove v from Even Outback and make it the new Last (Even). Halt when an idle sweep (a sweep that does not extend Even) is discovered or Even Outback is emptied. If Even Outback contains $\geqq\sqrt{n}$ vertices then go to HPA2.

LEMMA 1.1. *The expected time of Stage* 1 *is* $cn/p + o(n)$ *for some* c, *and almost surely* HPA1 *succeeds on Stage* 1, *i.e. almost surely* $<\sqrt{n}$ *vertices are left in Even Outback.*

Proof. On stage 1, HPA1 repeatedly makes attempts to extend Even by an Even Outback vertex. The total number of attempts is bounded by n^2. The expected number of attempts is $n/p_{\text{red}} + o(n)$. For, if an idle sweep is not discovered within n/p_{red} attempts, then almost surely Even acquires all even vertices: use Proposition 0.1 with $m = n/p_{\text{red}}$, $\beta = \frac{1}{2}$ and $r = p_{\text{red}}$. Since every attempt takes only bounded many steps, the expected time of Stage 1 is $cn/p + o(n)$ for some c.

Suppose that $\geqq\sqrt{n}$ vertices are left in Even Outback after Stage 1. It means that there is a vertex x (the final Last (Even)) such that examining $\geqq\sqrt{n}$ different pairs $\{x, v\}$, not examined earlier, reveals that none of them is a red edge. The probability of such an event is $\leqq n \times (q_{\text{red}}{}^{**}\sqrt{n})$ so that the event is negligible. □

Stage 1′. Extend Odd similarly, but place Odd Outback vertices at the beginning of Odd rather than at the end. In particular, Finish remains the last member of Odd.

Stage 2. Concatenate an initial segment of the current list Even with a final segment of the current list Odd into one path P. Let x_i be the ith element of Even from the end, and let y_j be the jth element of Odd (from the beginning). Try pairs $\{x_0, y_0\}$, $\{x_0, y_1\}$, $\{x_1, y_0\}$, etc. one after another to find one that is a red edge. The order is first by the sum of indices and then lexicographically. Halt when a red edge $\{x_i, y_j\}$ comes along or when $i+j$ reaches $\sqrt{n}$. If a red edge $\{x_i, y_j\}$ is discovered, then use the edge to concatenate the initial segment [Start, x_i] of Even and the final segment [y_j, Finish] of Odd into one Path P, then return $x_1, \cdots, x_{i-1}$ to Even Outback, and return $y_1, \cdots, y_{j-1}$ to Odd Outback. If $i+j$ reaches $\sqrt{n}$ go to HPA2.

LEMMA 1.2. *Almost surely,* HPA1 *succeeds on Stage* 2 (*if it arrives there*). *The expected time of Stage* 2 *is bounded by a constant* (*which depends on* p).

Proof. The proof is clear. □

Stage 3. Insert Even Outback vertices to P, one by one, by means of one sweep through P. In the following procedure, Pred is the predecessor function defined by P.

1. Set $x :=$ Last (P);
2. If Even Outback is empty then halt else set $v :=$ First (Even Outback);

3. If x is one of the first 4 vertices in P then go to HPA2;
4. If both $\{\text{Pred}(x), v\}$ and $\{v, x\}$ are red edges
   then place v into P between Pred (x) and x, set $x := \text{Pred}(x)$,
   drop v from Even Outback, go to 2
   else if $\{\text{Pred}(x), v\}$ is a red edge then set $x := \text{Pred}(x)$, go to 3
   else set $x := \text{Pred}(\text{Pred}(x))$, go to 3.

LEMMA 1.3. *The run-time of Stage* 3 *is bounded by some cn where c does not depend on p. Almost surely,* HPA1 *succeeds on Stage* 3 *(if it arrives there).*

Proof. The first statement is obvious. We prove the second.

First let us review the situation at the beginning of Stage 3. The Even Outback contains less than $2\sqrt{n}$ vertices which are all even. P has a tail of at odd vertices plus Finish, the length of the tail is at least $(n/2)-2\sqrt{n}$. If v is an Even Outback vertex and x belongs to the tail of P then the pair $\{v, x\}$ was not looked at earlier, hence $\mathbf{P}[\{v, x\} \text{ is a red edge}] = p_{\text{red}}$. Let $m = (1/2) \times ((n/2) - 2\sqrt{n})$.

On Stage 3, HPA1 repeatedly makes attempts to insert an Even Outback vertex into P. Let E_i be the event that either HPA1 does not make the ith attempt (i.e. Even Outback was emptied during previous attempts) or HPA1 makes the ith attempt and the ith attempt is successful. Obviously, the probability of E_i, assuming any primitive boolean combination of the previous events, is at least $r = p_{\text{red}}^2$. By Proposition 0.1, almost surely more than $mr/2$ events E_i happen. Hence, almost surely HPA1 succeeds on Stage 3. □

Stage 3′. Similarly insert the Odd Outback vertices in P.

Let us summarize the analysis of HPA1 in the following theorem.

THEOREM 1.1. *There is a positive real number c satisfying the following condition: Fix a probability p. Let G be a random member of the probability space* $\Gamma'(n, p)$ *for some n. The expected run-time of* HPA1 *on G is* $cn/p + o(n)$, *and the probability that* HPA1 *fails on G is* $2^{**}(-O(\sqrt{n}))$.

Proof. The proof is clear. □

To simplify the exposition, we have supposed above that the input to HPA1 is a colored graph whereas in effect the input is an ordinary graph and HPA1 should perform the red coloring.

THEOREM 1.2. *There is a positive real number c satisfying the following condition. Fix a probability p. Let G be a random member of the probability space* $\Gamma(n, p)$ *for some n. The expected run-time of* HPA1 *on G is* $cn/p + o(n)$, *and the probability that* HPA1 *fails on G is* $2^{**}(-O(\sqrt{n}))$.

Proof. It is easy to see that our analysis of HPA1 remains valid; in particular the three lemmas remain valid. The only difference is that the minimal appropriate constant c should be a bit larger to account for time spent for red coloring. □

2. Algorithm HPA2. The polynomial time algorithm HPA2, described in this section, attempts to construct or to establish the nonexistence of a Hamiltonian path in a given graph from one distinguished vertex Start to another distinguished vertex Finish.

To simplify the exposition, we suppose that the given graph is colored and is given by means of the extended adjacency matrix. The assumption will be removed at the end of this section. Our estimations of the run-time of HPA2 and the chances of HPA2 to succeed will remain true under the assumption of any realizable boolean combination of the events "$\{u, v\}$ is a red edge." In particular, they will remain true under the assumption that HPA1 fails on G, which is a boolean combination of the events "$\{u, v\}$ is a red edge" because HPA1 works only with red edges.

Again, the description of the algorithm is interleaved with some analysis. We fix a probability p and analyse the run of HPA2 on a random member G of the probability space $\Gamma^*(n, p)$ where n is arbitrary but not too small; we will take for granted, for example, that $n > 2 + 3/(-\log_2 q_{\text{green}})$.

HPA2 uses orange, yellow and green colors to construct a Hamiltonian path, and works in several stages. As in § 1, we will often ignore rounding reals in order to simplify exposition. Let V be the set of vertices of G and $l = \log_2 n$.

Stage 1. Compute the set T (for "troublesome") of vertices with small orange, yellow or green fan. For each vertex v, let $M(v) = \min$ {the number of C-edges incident to v: C is orange, yellow or green}. Compute $T' = \{v: M(v) < nl^{-2}\}$. If $|T'| \geqq 3/(-\log_2 q_{\text{green}})$ then go to HPA3, else set $T := T' \cup$ {Start, Finish} and output T.

LEMMA 2.1. *The time of Stage* 1 *is* $O(n^2)$. *The probability that* HPA2 *fails on Stage* 1 (*i.e.* $|T'| \geqq 3/(-\log_2 q_{\text{green}})$) *is bounded by* $2^{-3n+o(n)}$.

Proof. The first statement is obvious; we prove the second.

Let E be any event "$\{u, v\}$ is a C-edge" where u, v are distinct vertices and $C \in$ {orange, yellow, green}, and let $q = q_{\text{green}}$. By Lemma 0.1 and the definition of q_{green}, the probability of $\bar{E}$ (which is the complement of E), assuming any realizable boolean combination of the events "$\{x, y\}$ is a D-edge" different from E, is at most q.

Let $m = nl^{-2}$. For each vertex $v \in T'$, there are a color $C \in$ {orange, yellow, green} and a set $S \subseteq V - \{v\}$ of at most $m - 1$ vertices such that there are no C'-edges between v and $(V - \{v\}) - S$. Hence,

$$\mathbf{P}[v \in T'] \leqq 3\binom{n-1}{m-1} q^{n-m} \leqq 3n^{m-1} q^{n-m},$$

$$\mathbf{P}[|T'| \geqq k] \leqq \binom{n}{k} (3n^{m-1} q^{n-m})^k \leqq n^k (3n^{m-1} q^{n-m})^k \leqq (3n^m q^{n-m})^k,$$

and

$$\log_2 \mathbf{P}[|T'| \geqq k] \leqq k[\log_2 3 + ml - (n-m)(-\log_2 q)].$$

Hence $\log_2 \mathbf{P}[|T'| \geqq 3/(-\log_2 q)] \leqq -3n + o(n)$. □

Stage 1 of HPA2 is a special procedure which outputs the set T of troublesome vertices. The extended adjacency matrix and the set T constitute the input for the rest of HPA2. Let $\Gamma^*_T(n, p)$ be the subspace of $\Gamma^*(n, p)$ defined by the given set T; a member A of $\Gamma^*(n, p)$ belongs to $\Gamma^*_T(n, p)$ if and only if the given T is the output of Stage 1 applied to A. In the rest of this section, we work with a random member G of $\Gamma^*_T(n, p)$.

PROPOSITION 2.1. *Let numbers* c_1, c_2 *and* c_3 *satisfy Lemma* 0.1. *Suppose that*

$J =$ {red, orange, yellow, green} *and* $C \in$ {orange, yellow, green},

K *is a subset of* $J - \{C\}$ *and* $k = |K|$,

$\{u, v\}$ *is a pair of distinct vertices in* $V - T$,

E *is the event "*$\{u, v\}$ *is an edge of color* C,"

H *is a realizable boolean combination of events "*$\{x, y\}$ *is of color* D" *excluding* E.

Then $\mathbf{P}[E \mid H] \geqq c_k \cdot p$.

Proof. Without loss of generality, we may assume that H is a primitive boolean combination: see the proof of statement (b) of Lemma 0.1. Let a be the cardinality of the set $\{x \in V - \{u, v\}: \{u, x\}$ is an edge of color C with respect to $H\}$, and b be the cardinality of the set $\{x \in V - \{u, v\}: \{v, x\}$ is an edge of color C with respect to $H\}$. Let $m = nl^{-2}$. If $a < m - 1$ then $u \in T$ which is impossible; hence $a \geqq m - 1$. Similarly, $b \geqq m - 1$. If $a = m - 1$ or $b = m - 1$ then $\mathbf{P}[E \mid H] = 1$. Suppose $a \geqq m$ and $b \geqq m$. Without loss of generality, in the probability space $\Gamma^*(n, p)$, hypothesis H implies the

event $G \in \Gamma_T^*(n, p)$. Therefore the desired conditional probability $\mathbf{P}[E \mid H]$ can be computed in $\Gamma^*(n, p)$. Now use statement (c) of Lemma 1 in § 0. □

In order to motivate the next definition, let us notice that any Hamiltonian path P and any set U of vertices give rise to a family $\{P_u: u \in U\}$, where P_u is the maximal segment of P such that $u \in P$ and U contains at least one of any two neighboring elements of P.

DEFINITION. Let U be a set of vertices. A family F of disjoint (having no common vertices) paths is an *envelope* for U if

(a) $U \subseteq \cup F$;

(b) Every U-vertex, different from Start and Finish, is an internal vertex of some F-path;

(c) If Start belongs to U then it is the first element of some F-path, if Finish belongs to U then it is the last element of some F-path, and if both Start and Finish belong to U then they belong to different F-paths unless $U = V$ and $|F| = 1$;

(*d*) If u, v are neighbors in some F-path then $u \in U$ or $v \in U$.

Stage 2. Construct an envelope F for T, or establish the nonexistence of such an envelope. The desired envelope is constructed by brute force but we should be a little careful: even though T is small, there may be relatively many vertices adjacent to T-vertices. For each vertex v, let Fan (v) be the number of all edges incident to v.

1. Set $X := T$ and List := Empty.
2. While there is $v \in X$ with Fan $(v) \geqq 3|X|$ do the following:
   Make one such v the head of List and set $X := X - \{v\}$.
3. Find an envelope F for X or the nonexistence of such an envelope by exhaustive search. (This is possible because there are $\leqq 3|T|^2$ vertices v such that v belongs to X or is adjacent to a vertex in X.) In the negative case write "There is no Hamiltonian path from Start to Finish," and halt.
4. While List $\neq$ Empty do the following:
   4.1. Set $v :=$ Head (List), List := Tail (List).
   4.2. Extend F to an envelope for $X \cup \{v\}$. Notice that $\cup F$ contains at most $3|X|$ vertices whereas v is adjacent to at least $3(|X|+1)$ vertices. If $v =$ Start and v is the first vertex of some $P \in F$, or $v =$ Finish and v is the last vertex of some $P \in F$, or v is an internal vertex of some $P \in F$ then do nothing, otherwise do the following. If v is an end of some P in F then pick $u \in V - \cup F$ adjacent to v, and extend P by the edge $\{v, u\}$. If $v \notin \cup F$ then pick
   $u \in \{x: x \in V - \cup F, x \text{ is adjacent to } v, \text{ and } x \neq \text{Finish}\}$,
   $w \in \{x: x \in V - \cup F, x \text{ is adjacent to } v, x \neq \text{Start}, \text{ and } x \neq u\}$
   and extend F by a new path (u, v, w).
   4.3. Set $X := X \cup \{v\}$.

LEMMA 2.2. *The time of Stage* 2 *is* $O(n)$.

Proof. The proof is clear. □

Remark. In connection to the case of decreasing $p(n)$, one of the referees was interested in the run-time of step 3. We describe a variation of the dynamic programming algorithm (Bellman (1960); Held and Karp (1962)) to carry out step 3. Let $U = \{v \in V - T: v \text{ is adjacent to a vertex in } T\}$. A chain P of vertices from $T \cup U$ will be called a *pseudo-path* if (a) Start is the first vertex in P, (b) if v is the successor of u in P but $\{u, v\}$ is not an edge then $\{u, v\} \subseteq U$, there is an edge between u and its P-predecessor, and there is an edge between v and its P-successor unless v is the last in P, (c) if v is the successor of u in P and $\{u, v\}$ is an edge then $\{u, v\} \cap T \neq \emptyset$, (d) if Finish belongs to P then $T \subseteq P$ and Finish is the last in P. The idea is to compute

a function f whose properties are described below. The domain of f consists of triples (X, Y, v) where $\{\text{Start}\} \subseteq X \subseteq T$, $Y \subseteq U$, $|Y| \leqq 2|X|$, and $v \in X \cup Y$. Each $f(X, Y, v)$ is either a pseudo-path P such that $X \subseteq P$ and v is the last in P, or the string "The desired pseudo-path does not exist." Compute f by induction on $|X|$. Turn any pseudo-path $f(T, Y, \text{Finish})$ into the desired envelope for T; if no $f(T, Y, \text{Finish})$ is a pseudo-path then write "There is no Hamiltonian path from Start to Finish" and halt. The run-time of this algorithm is bounded by |domain (f)| times a polynomial in n.

Stage 3. Extend F-paths using orange edges. Let $P_1, \cdots, P_k$ be the F-paths where Start = First (P_1), and Finish = Last (P_k). Obviously, $k \leqq |T|$. The orange fans of vertices in $V - T$ are at least n/l^2; in particular, the orange fans of Last $(P_1), \cdots,$ Last (P_{k-1}), First $(P_2), \cdots,$ First (P_k) are at least n/l^2. The union of the paths $P_1, \cdots, P_k$ contains less than $3|T|$ vertices. Find the maximal m such that $3|T| + 2m(k-1) \leqq n/l^2$. Using only orange edges, extend each path $P_2, \cdots, P_k$ by an additional final segment of length m and extend each path $P_1, \cdots, P_{k-1}$ by an additional initial segment of length m in such a way that the extended paths $P_1, \cdots, P_k$ remain disjoint.

LEMMA 2.3. *The time of Stage* 3 *is* $O(n)$.

Proof. The proof is obvious. □

Stage 4. Sew the paths $P_1, \cdots, P_k$ into one path Q_0 using yellow edges. For $i < k$, let A_i be the final tenth part of P_i; for $j > 1$, let B_j be the initial tenth part of P_j. For each $i < k$, find $x_i \in A_i$ and $y_i \in B_{i+1}$ such that $\{x_i, y_i\}$ is a yellow edge, then throw from P_i the successors of x_i and throw from P_{i+1} the predecessors of y_i. In the case of failure (for at least one i) go to the algorithm HPA3; in the case of success sew the (possibly shortened) paths $P_1, \cdots, P_k$ into one path Q_0 using the yellow edges $\{x_i, y_i\}$.

LEMMA 2.4. *The time of Stage* 4 *is* $O(n^2/l^{-4})$. *The probability of failure is bounded by* $2^{**}O(-n^2/l^{-4})$.

Proof. The proof is clear. The expected computation time is bounded by a constant (which depends on p). □

Stage 5. Partition $V - Q_0$ into disjoint paths $Q_1, Q_2, \cdots$ satisfying the following condition: either Q_{i+1} is a circuit (a closed path) of length at least $nl^{-2}/2$ or each of the two end-points of Q_{i+1} is adjacent to at least $nl^{-2}/2$ vertices in $\bigcup_{j \leqq i} Q_j$. Assuming that $Q_1, \cdots, Q_i$ have been constructed, $U_i = \bigcup_{j \leqq i} Q_j$, and $V - U_i \neq \varnothing$, we construct Q_{i+1} using, say, yellow edges.

Pick a vertex in $V - U_i$ and set X equal to the path consisting of that vertex only. While there is a yellow edge from Last (X) to a vertex in $V - (U_i \cup X)$, append X by such a vertex. While there is a vertex in $V - (U_i \cup X)$ with a yellow edge to First (X), pick one such vertex and make it the new first vertex of X.

If there are at least $nl^{-2}/2$ yellow edges between First (X) and U_i as well as between Last (X) and U_i, then set $Q_{i+1} := X$. Otherwise let u be an end-point of X with less than $nl^{-2}/2$ yellow edges between u and U_i. Without loss of generality, $u =$ First (X). There are more than $nl^{-2}/2$ vertices v of X such that $\{v, u\}$ is a yellow edge; let w be the last (with respect to X) among these vertices. Turn X into a circuit Q_{i+1} by throwing away the successors of w and adding the edge $\{w, u\}$.

LEMMA 2.5. *The time of Stage* 5 *is* $O(n)$.

Proof. The proof is clear. □

Before we explain the last stage of HPA2, let us introduce some terminology and notation. If P is an open path P, $v \in P$ and $v \neq$ Last (P) then v' is the successor of v with respect to P. We suppose that every closed path (circuit) comes with a distinguished member, called the *initial member*, and a specified direction of traversing the path. Since our vertices are natural numbers, we can use, for example, the following rule:

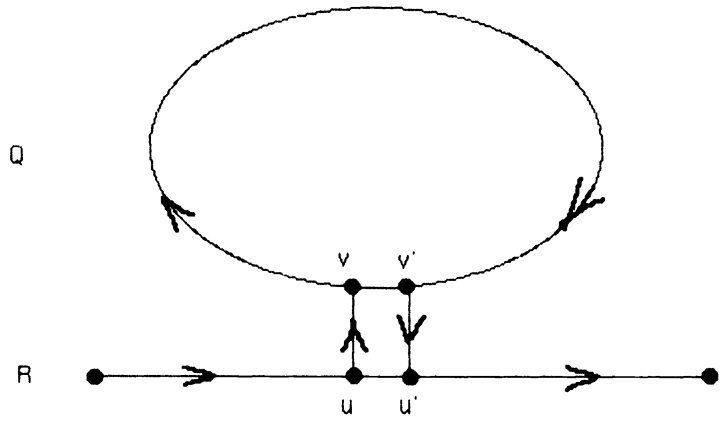


FIG. 1

the member of a circuit with the least numerical value is the initial member, and the lesser of the two neighbors of the initial member defines the direction of traversing. If P is a circuit and $v \in P$ then v' is the predecessor of v with respect to P.

After Stage 5 we have a sequence $Q_0, Q_1, \cdots, Q_m$ of paths. Let Q_{m+1} be the empty set. If $Q = Q_i$ and $i \leqq m$ let $Q' = Q_{i+1}$.

Stage 6. Merging the paths $Q_0, Q_1, \cdots, Q_m$ into the desired Hamiltonian path.

1. Set $R := Q_0$ and $Q := Q_1$.
2. If Q is empty then halt.
3. If Q is a circuit then do the following:
   3.1. Set u equal to the first (with respect to R) member of $\{x \in R - T: x' \notin T\}$, and v equal to the initial member of Q.
   3.2. If $\{u, v\}$ and $\{v', u'\}$ are green edges, then set R equal to the concatenation of:
   the initial segment [Start, u] of R,
   the segment [v, v'] of Q (which includes all vertices of Q), and
   the final segment [u', Finish] of R,
   set $Q := Q'$, and go to 2. (See Fig. 1.)
   3.3. Find or establish the nonexistence of a pair (x, y) such that $x \in R - T$, $x' \notin T$, $y \in Q$, and neither of the pairs $\{x, y\}$, $\{y', x'\}$ has yet been examined for being a green edge. In the positive case set $u := x$, $v := y$ and go to 3.2; in the negative case go to HPA3.
4. If Q is an open path, then do the following.
   4.1. Set
   $A := \{w \in R - T: w' \notin T$, and $\{w, \text{First}(Q)\}$ is a yellow edge$\}$,
   $B := \{w \in R - T: w' \notin T$, and $\{\text{Last}(Q), w\}$ is a yellow edge$\}$,
   I equal to the least initial segment of R that contains one half of A or one half of B.
   If I does not contain one half of A, then set Q equal to the reverse of Q, and swap A and B.
   (When step 4.1 is completed, $A = \{w \in R - T: w' \notin T$, and $\{w, \text{First}(Q)\}$ is a yellow edge$\}$, I is an initial segment of R which contains at least one half of A, $B = \{w \in R - T: w' \notin T$, and $\{w, \text{Last}(Q)\}$ is a yellow edge$\}$, and $R - I$ contains at least one half of B.)

4.2. Set
u equal to the first (with respect to R) member of $A \cap I$,
v equal to the first (with respect to R) member of $B - I$.
4.3. If $\{u', v'\}$ is a green edge then
set R equal to the concatenation of:
the segment [Start, u] of R,
the path Q,
the reverse of the segment $[u', v]$ of R,
the segment $[v', \text{Finish}]$ of R,
set $Q := Q'$, and go to 2. (See Fig. 2.)
4.4. Find or establish the nonexistence of a pair (x, y) such that $x \in A$, $y \in B$, and the pair $\{x', y'\}$ was not examined on being a green edge yet. In the positive case set $u := x$, $v := y$ and go to 4.3; in the negative case go to HPA3.

LEMMA 2.6. *The computation time of Stage* 6 *is* $O(n^2)$. *The probability of the failure on Stage* 6 *is bounded by* $2^{**}O(-n^2/l^{-4})$.

Proof. On Stage 6, HPA2 repeatedly makes attempts to combine the current paths Q and R into a new path R. Each attempt is uniquely identified by a pair (u, v) of vertices. Therefore the number of attempts is bounded by n^2. This gives the bound on the computation time.

It remains to prove the bound on the probability of failure. There is a fraction m of n/l^2 such that in no case HPA2 transfers the control to HPA3 within first m attempts. For $i = 1, \cdots, m$ let E_i be the event that either HPA2 does not make the ith attempt (i.e. the desired Hamiltonian path is constructed during previous attempts) or HPA2 makes the ith attempt and the ith attempt is successful. The probability of E_i, assuming any primitive boolean combination of events E_j with $j < i$, is at least $r = p^2_{\text{green}}$. By Proposition 0.1, the probability that at most $mr/2$ events E_i happen, is $2^{**}O(-n^2/l^{-4})$. Hence the probability that HPA2 fails is $2^{**}O(-n^2/l^{-4})$. □

Note. Reading the technical report (Gurevich and Shelah (1984)), one of the referees expressed the following concern. Suppose that we are merging R with $Q = Q_i$ which happened to be an open path. The set $A \cap I$ contains $\Omega(n/l^2)$ elements, and the set $B \cap (R-1)$ contains $\Omega(n/l^2)$ elements. Thus we have $\Omega(n^2/l^4)$ pairs to search on step 4.4. However, many of those pairs could be examined earlier, when we tried to merge R with those Q_j that $j < i$ and Q_j is an open path. Are there enough pairs left to guarantee the high chances of success in merging R and Q_i?

It is true that Stage 6 splits into the substages of merging R and Q_1, R and Q_2, etc. But consider Stage 6 as one sequence of attempts to merge the current R and the current Q. If HPA2 fails on Step 6, then the great majority of the attempts were

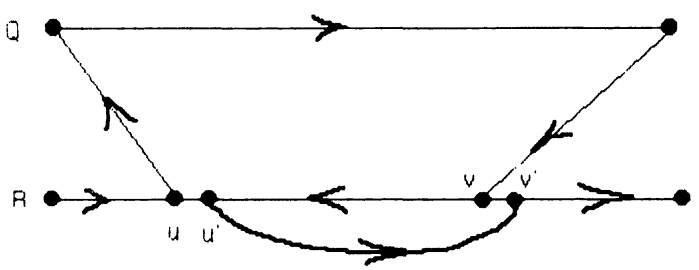


FIG. 2

unsuccessful. We do not care how the unsuccessful attempts were spread over different substages. By Proposition 0.1, such an event is unlikely.

Let us summarize the analysis.

THEOREM 2.1. *The run-time of* HPA2 *on a graph with n vertices is* $O(n^2)$.

Proof. Use Lemmas 2.1–2.6. □

THEOREM 2.2. *Fix a probability p. The probability that* HPA2 *fails on a random member G of a probability space* $\Gamma^*(n, p)$ *is* $2^{-3n+o(n)}$.

Proof. Use Lemmas 2.1, 2.4 and 2.6. □

To simplify the exposition, we have assumed that the input is a random member of $\Gamma^*(n, p)$ whereas in effect we are interested in an input which is the output of HPA1. Instead of a fully colored graph, given by the extended adjacency matrix, we have a graph which is only partially colored red. In addition, we have important information that HPA1 has failed on the corresponding uncolored graph. Let HPA1(G) be the outcome of HPA1 on the input G in the case that HPA1 fails of G.

THEOREM 2.3. *Fix a probability p. Let G be a random member of some* $\Gamma(n, p)$. *The probability that* HPA2 *fails on* HPA1(G), *assuming that* HPA1 *has failed on G, is* $2^{-3n+o(n)}$.

Proof. The idea is that the analysis above remains valid. The coloring is completed on Stage 1 of HPA2. This does not change our estimation of the computation time of Stage 1, and it does not matter for the rest of HPA2 when the graph was colored.

Further, HPA2 never looks to the red part of the extended adjacency matrix. It is easy to see that all six lemmas remain valid under the assumption of any realizable boolean combination of events "$\{u, v\}$ is a red edge." In particular, they remain valid under the assumption that HPA1 fails on the original input. This particular assumption is a boolean combination of events "$\{u, v\}$ is a red edge" because HPA1 utilizes only red edges. □

3. Algorithm HPA3. One obvious algorithm for the Hamiltonian Path Problem is a straightforward exhaustive search: examine in turn all permutations of the vertices different from Start and Finish. Stirling's formula gives an upper bound 2^{nl} on the run-time of the straightforward exhaustive search algorithm on graphs with n vertices; in this section $l = \log_2 n$.

The fastest known algorithm for the Hamiltonian Path Problem is a version of the so-called dynamic programming algorithm (Bellman (1960), Held and Karp (1962)). We sketch the idea roughly. Suppose for simplicity that Start $\neq$ Finish. We will compute a function f satisfying the following conditions. The domain of f consists of pairs (X, v) where X is a set of vertices, Start belongs to X, Finish does not belong to X unless X contains all vertices, v is an element of X different from Start, and if X contains all vertices then $v =$ Finish. Each $f(X, v)$ is either a Hamiltonian path in X from Start to v or the string "The desired Hamiltonian path does not exist." The computation proceeds by induction in $|X|$. The run-time of this algorithm on graphs with n vertices is bounded by 2^n times a polynomial in n. Unfortunately, this algorithm requires exponential space (and a computer with addresses of length about n).

Combining the idea of the dynamic programming algorithm with the idea of Savitch's theorem (Savitch (1970)), we give here a recursive algorithm HPA3 for the Hamiltonian Path Problem. The run-time of HPA3 on a graph with n vertices is bounded by 2^{2n} times a polynomial in n; it is slower than the dynamic programming algorithm but still much faster than the exhaustive search algorithm. The work space of HPA3 on graphs with n vertices is $O(n)$.

We will describe HPA3 informally. Let $(G, \text{Start}, \text{Finish})$ be an instance of Hamiltonian Path Problem, and V be the set of vertices of the graph G. Without loss of generality, $V = \{0, 1, \cdots, n-1\}$ for some n. As in the previous sections, we will often ignore rounding reals in order to simplify exposition.

If $n \leqq 3$ then use the exhaustive search algorithm. Suppose that $n > 3$.

Let Center range over the set $V - \{\text{Start}, \text{Finish}\}$,

$U = V - \{\text{Start}, \text{Center}, \text{Finish}\}$,

A range over subsets of U such that $|A| = |U|/2$ and $\min(U) \in A$ unless $U = \varnothing$,

$B = U - A$,

A' be the subgraph of G with the universe $A \cup \{\text{Start}, \text{Center}\}$,

B' be the subgraph of G with the universe $B \cup \{\text{Center}, \text{Finish}\}$.

For each pair (Center, A) in turn recursively apply HPA3 to the triples $(A', \text{Start}, \text{Center})$ and $(B', \text{Center}, \text{Finish})$. If HPA3 constructs a Hamiltonian path in A' from Start to Center and a Hamiltonian path in B' from Center to Finish, then concatenate the two paths into a Hamiltonian path in G from Start to Finish and halt. Otherwise go to the next pair (Center, A). If there is no next pair, then write "There is no Hamiltonian path from Start to Finish" and halt.

Obviously, the algorithm HPA3 solves the Hamiltonian Path Problem.

THEOREM 3.1. *Let $S(n)$ be the work space and $T(n)$ be the time that* HPA3 *needs in order to handle graphs with n vertices.*

(a) *In an appropriate implementation of* HPA3, $S(n) = O(n)$.

(b) $T(n) = 2^{2n} \times n^{O(1)}$.

Proof. (a) The idea is to pass A (resp. B) to the recursive procedure in the form of the characteristic function which is a boolean vector. The recursive procedure passes further a shorter boolean vector which is a characteristic function that refers to the natural order of A (resp. B). There is a constant b such that for all $n > 3$,

$$S(n) \leqq bn + S((n-3)/2+2) \leqq bn + S(n/2+1).$$

Choose a constant c such that $S(n) \leqq cn$ for $n \leqq 3$, and $bn + c(n/2+1) \leqq cn$ for $n > 3$.

Check by induction on n that $S(n) \leqq cn$ for all n.

(b) There are $n-2$ possible values of Center. For each Center, the number of possible values for A or B is bounded by 2^{n-3}. For each pair (Center, A), HPA3 calls the recursive procedure twice. There is a constant b such that for all $n > 3$,

$$T(n) \leqq 2^{n+bl} \times T(n/2+1).$$

Choose a constant c such that $T(n) \leqq 2^{2n+cl}$ for $n \leqq 3$, and $2^{n+bl} \times 2^{(2n+cl)/2+1} \leqq 2^{n+cl}$ for $n > 3$.

Check by induction on n that $T(n) \leqq 2^{n+cl}$ for all n. □

Remark. Why is HPA3 faster than the straightforward exhaustive search algorithm? To answer this question we use the notation of HPA3. For given Center and A, we spend some time on A *plus* some time on B whereas the straightforward exhaustive search goes through all permutations of A *times* all permutations of B.

Theorems 1.2, 2.2 and 3.1 imply Theorem 1 of the Introduction.

Remark. It is important that the product of the probability of HPA2 to fail and the time bound of HPA3 is $o(n)$. This explains the expression $3/(-\log_2 q_{\text{green}})$ on Stage 1 of HPA2; it guarantees the probability $2^{-3n+o(n)}$ of HPA2 to fail.

4. The case of directed graphs. In this section we prove Theorem 2 of the Introduction. The adaptation of HPA to directed graphs is pretty obvious almost everywhere. The exceptions are Stage 2 and step 4 of Stage 6 of HPA2 which are treated below.

Before we turn to the two exceptions let us notice that Stage 1 of HPA2 should be modified to ensure that T contains all vertices v such that either the orange, yellow or green fan-in of v is less than n/l^2 or the orange, yellow or green fan-out of v is less than n/l^2.

First, we modify Stage 2 of HPA2. Notice that if $v \in X$, and Fan-in $(v) \geqq 3|X|$ unless $v =$ Start, and Fan-out $(v) \geqq 3|X|$ unless $v =$ Finish, then every envelope for $X - \{v\}$ can be easily extended to an envelope for X. Hence the problem reduces to finding or establishing the nonexistence of an envelope for some subset S of T such that for every $v \in S$, either Fan-in $(v) < 3|S|$ or Fan-out $(v) < 3|S|$. Let Y range over minimal sets of edges such that (a) if $v \in S$, $v \neq$ Start and Fan-in $(v) < 3|S|$ then Y contains an edge $\{u, v\}$, and (b) if $v \in S$, $v \neq$ Finish and Fan-out $(v) < 3|S|$ then Y contains an edge $\{v, w\}$. There are only bounded many of different Y's. The problem reduces to the following: given a set Y, find or establish the nonexistence of an envelope F for S such that every Y-edge is a part of an F-path. The Y-edges form a family of paths. If one of the paths is a circle, then the task is impossible, otherwise the task is easy.

Second, we modify step 4 of Stage 6. This time we cannot reverse the order of Q or any segment of R. Set A, B and I as in 4.1. We are forced to consider two cases.

Case 1. I contains one half of A (and therefore $R - I$ contains at least one half of B). Find or establish the nonexistence of a triple (u, v, w) such that $u \in A \cap I$, v is a nonfinal vertex of Q, $w \in B \cap (R - I)$, and (v, u'), (w, v') are edges. In the negative case go to HPA3. In the positive case set R equal to the concatenation of segment [Start, u] of R, segment [First (Q), v] of Q, segment [u', w] of R, segment [v', Last (Q)] of Q, and segment [w', Finish] of R. (See Fig. 3.)

Case 2. I contains one half of B (and therefore $R - I$ contains at least one half of A). Find or establish the nonexistence of a triple (u, v, w) of vertices such that

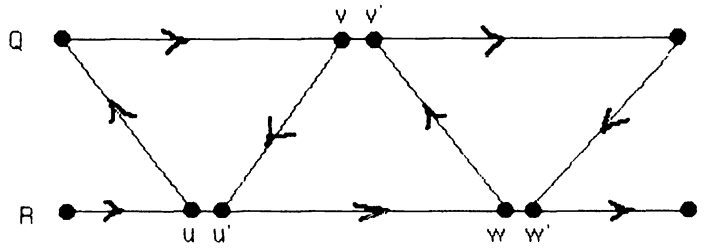


FIG. 3

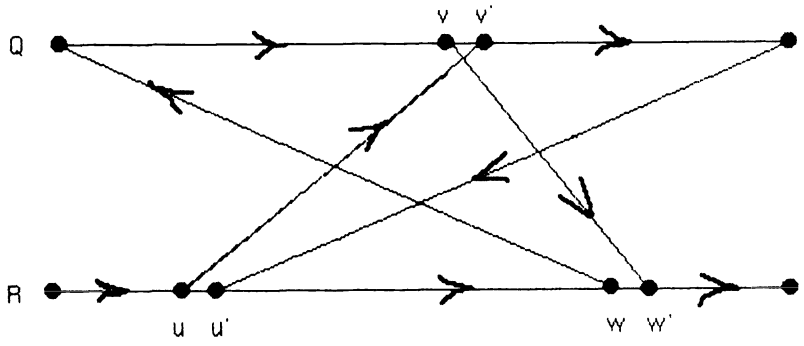


FIG. 4

$u' \in B \cap I$, v is a nonfinal vertex in Q, $w \in A \cap (R - I)$, and (u, v'), (v, w') are edges. In the negative case go to HPA3. In the positive case set R equal to the concatenation of segment [Start, u] of R, segment [v', Last (Q)] of Q, segment [u', w] of R, segment [First (Q), v] of Q, and segment [w', Finish] of R. (See Fig. 4.)

Theorem 2 of the Introduction is proved. □

REFERENCES

D. ANGLUIN AND L. G. VALIANT (1979), *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18, pp. 155-193.

R. BELLMAN (1960), *Combinatorial processes and dynamic programming*, in Proc. of the 10th Symposium in Appl. Math., Amer. Math. Soc., Providence, RI.

B. BOLLOBAS (1979), *Graph Theory*, Springer-Verlag, New York, Berlin.

B. BOLLOBAS, T. I. FENNER AND A. M. FRIEZE (1985), *An algorithm for finding Hamilton cycles in a random graph*, in Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 430-439.

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

P. ERDÖS AND J. SPENCER (1974), *Probabilistic Methods in Combinatorics*, Academic Press, New York.

M. R. GAREY AND D. S. JOHNSON (1979), *Computers and Intractability*, W. H. Freeman, San Francisco.

R. L. GRAHAM (1983), *Applications of the FGK inequality and its relatives*, in Mathematical Programming, The State of Art, Bonn 1982, Springer-Verlag, New York, Berlin.

Y. GUREVICH AND S. SHELAH (1984), *Expected computation time for Hamiltonian Path Problem and clique problem*, Tech. Report CRL-TR-50-84, Univ. of Michigan, Ann Arbor.

M. HELD AND R. M. KARP (1962), *A dynamic programming approach to sequencing problems*, J. Soc. Indust. Appl. Math., 10, pp. 196-210.

D. S. JOHNSON (1984), *The NP-completeness column: an ongoing guide*, J. Algorithms, 5, pp. 284-299.

L. LEVIN (1984), *Problems, complete in "average" instance*, in Proc. 16th Annual ACM Symposium on Theory of Computing, p. 465.

L. POSA (1976), *Hamiltonian circuits in random graphs*, Discrete Math., 14, pp. 359-364.

W. J. SAVITCH (1970), *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4, pp. 177-192.

E. SHAMIR (1983), *How many random edges make a graph Hamiltonian?* Combinatorica, 3, pp. 123-131.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987

© 1987 Society for Industrial and Applied Mathematics
006

WEIGHTED LEAF AVL-TREES*

VIJAY K. VAISHNAVI†

Abstract. A new data structure called a weighted leaf AVL-tree is presented for representing a collection of weighted items drawn from a totally ordered set. It is shown that using this data structure, the average access time of an item is optimal to within a constant factor while the worst-case access time is "logarithmic." The data structure is a generalization of a leaf AVL-tree to the weighted-item case and retains its restructuring simplicity. It is shown that the data structure supports a number of update operations in a simple and worst-case efficient manner.

Key words. algorithms, data structures, search trees, AVL-trees, weighted trees, weighted dynamic dictionaries, globally biased trees, information storage and retrieval, dynamic databases

AMS(MOS) subject classifications. 68A10, 68A20

1. Introduction. An ordered list of n items can be organized into a (standard) *search tree* [6]. Alternatively, it can be organized into a *leaf* (i.e., *extended*) *search tree* [6]. In a leaf search tree, the leaf (external) nodes store items in the left to right order, and the internal nodes contain *keys* for directing the search. If the tree is suitably "balanced," then the worst-case access time of any item as well as the worst-case time needed for performing an update operation is $O(\log n)$. An AVL-tree [2] (standard or leaf) is a prominent example of *balanced binary trees.*

If the items have positive integer weights associated with them, reflecting, for example, the frequency of their access, then we have a more general problem of organizing an ordered list of weighted items into a suitably balanced search tree, such that the items can be accessed as well as updated efficiently in the worst case. The update operation may be the insertion of an item, the deletion of an item, or a change in the weight of an item. Such a search tree can, for example, be used in a real-time application as a dynamic self-organizing search tree that adapts itself to the access pattern.

For any *weighted leaf binary search tree* storing n items, i.e., a leaf binary search tree in which weights are associated with items, it can be shown, using an entropy argument [1], that the weighted path length of the tree is bound below by $\Sigma_i w_i \log_2 (W/w_i)$, where w_i is the weight of item i, and $W = \Sigma_i w_i$. Thus, the best that can be expected from a weighted leaf binary search tree is "logarithmic behavior," i.e., the access time for item i is $O(\log_2 (W/w_i))$, in the worst case. This guarantees minimal average access time of the tree to within a constant factor, i.e., nearly optimal behavior. $O(\log_2 (W/w_i))$ has been called the *ideal access time* [3] for item i.

In this paper, we provide a generalization of leaf AVL-trees to the weighted-item case, and investigate it. We call such trees *weighted leaf AVL-trees.* We show that in the worst case these trees support the access operation in ideal time. We also provide simple and worst-case efficient algorithms for various update operations. We show the following worst-case update results for these trees: (a) Item i can be inserted in $O(\log_2 (W/w_i) + \log_2 (W/(w_{i-} + w_{i+})))$ time, and deleted in $O(\log_2 (W/w_i) + \log_2 (\bar{W}/(w_{i-} + w_{i+})))$ time, where w_{i-} is the weight of the largest item smaller than i,

* Received by the editors April 19, 1984; accepted for publication (in revised form) August 14, 1986. This research was partially supported by a grant from the College of Business Administration, Georgia State University, Atlanta, Georgia 30303. An early version of part of this paper was presented at the 21st Annual Allerton Conference on Communication, Control and Computing.

† Department of Computer Information Systems, Georgia State University, Atlanta, Georgia 30303.

w_{i+} is the weight of the smallest item larger than i, and $\bar{W}$ is the sum of weights after the update. (b) Item i can be *promoted* (i.e., its weight increased) to $\bar{w}_i$ in $O(\log_2(W/w_i))$ time, and *demoted* (i.e., its weight decreased) to $\bar{w}_i$ in $O(\log_2(W/w_i)+\log_2(\bar{W}/\bar{w}_i))$ time.

There have been other similar attempts at generalizing balanced search trees to the weighted-item case [3]–[5], [8] and [9]. The work reported in this paper is, however, to our knowledge, the first successful attempt at generalizing AVL-trees. Moreover, the data structure is simple and the worst-case running time for the update operations investigated are the same or better than any data structure available in the literature. We will return to this point in the final section.

We define weighted leaf AVL-trees in § 2. We also show in that section that the worst-case access time for these trees is ideal. In §§ 3 and 4, we present and analyze algorithms for increasing and decreasing the rank of an item. The update results presented in § 5 depend almost entirely on the algorithms presented in §§ 3 and 4. In the final section, we discuss various issues and draw conclusions.

2. Weighted leaf AVL-trees and their access time. Let T be a "weighted" leaf (i.e., extended) binary tree. In such a tree, leaf nodes store "weighted" items, i.e., items having positive integer weights associated with them. Let item i have weight w_i associated with it; let it be stored in the ith leaf node (in order from the left). The item can obviously be represented as (i, w_i).

The *rank* of a node X in T, $rk(X)$, is defined recursively as follows:

$rk(X)=F(w_i)+1$, if X is a leaf node storing item i, where $F(w_i)=\lfloor 2.464\log_2(w_i+1.5)-0.56\rfloor$, an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and w_i leaf nodes (see [7] and Lemma 2.6).

$rk(X)=1+\max\{rk(Y)\mid Y \text{ is a son of } X\}$, if X is an internal node.

If V is a son of U (in T) and $rk(U)-rk(V)>2$, then U and V form a *major gap* (*major rank gap*). The major gap is *supported* by W, a neighboring[1] leaf node of V, if $rk(W)\geqq rk(U)-2$.

A *weighted leaf AVL-tree* is a weighted leaf binary tree in which every major gap is supported.

A weighted leaf AVL-tree can be used as a (leaf) search tree in an obvious manner; we will always mean it to be as such.

Example 2.1. Figure 2.1 shows a weighted leaf AVL-tree. Observe that there are two major gaps in the tree, one between A and C, and the other between F and E. B and D are the left neighboring leaf nodes of C and E, respectively. D is also the right neighboring leaf node of A. Similarly, D and G are the right neighboring leaf nodes of C and E, respectively. Note that both major gaps in the tree are supported by D.

Note that the rank of a node in a weighted leaf AVL-tree is the same as its height[2] *if* the leaf node storing item i, $1\leqq i\leqq n$, is *placed* at height $1+F(w_i)$. It follows that a leaf AVL-tree can also be defined as a weighted leaf binary tree in which all weights are the same and there are no major gaps. The following result shows that a weighted

[1] The *left* (resp., *right*) *neighboring leaf node* (if any) of V is the rightmost (resp., leftmost) leaf node to the left (resp., right) of the leftmost (resp., rightmost) leaf descendant of V; a node is a descendant of itself. (See Example 2.1.)

[2] The *height* of a tree is zero if the tree is empty; otherwise, it is one plus the height of the tallest subtree at the root of the tree. The *height* of a node in a tree is the height of the tree rooted at the node.

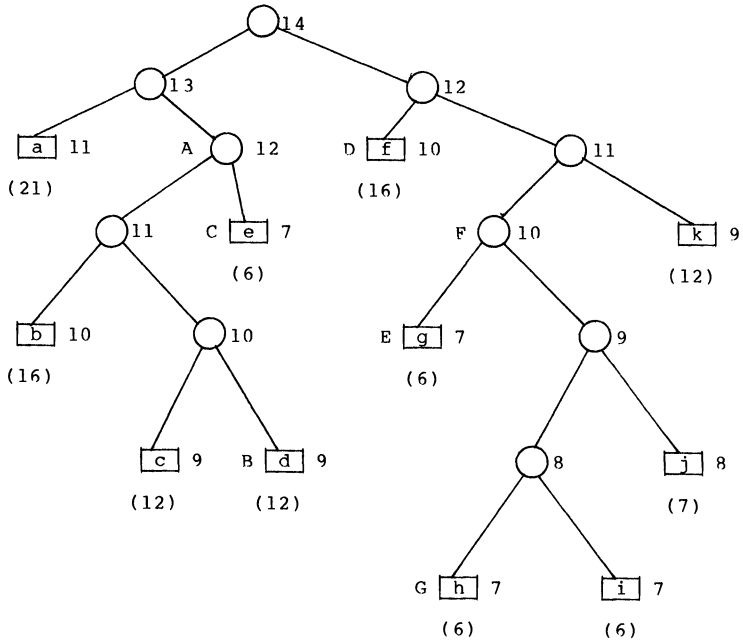


FIG. 2.1. *A weighted leaf AVL-tree. The items are shown within leaf nodes and their weights are shown underneath within parentheses. The rank of a node is shown on its right side.*

leaf AVL-tree is a generalization of a leaf AVL-tree to the case where the weights of items are not necessarily the same.

LEMMA 2.1. *Let all weights be the same in a weighted leaf binary tree T. T is a weighted leaf AVL-tree if and only if it is a leaf AVL-tree.*

Proof. If T is a leaf AVL-tree, then it does not have any major gap. T is then clearly a weighted leaf AVL-tree.

If T is a weighted leaf AVL-tree, then its major gaps must be supported. Let there be a major gap between nodes U and V which is supported by W, a neighboring leaf node of V. This implies that the rank of W must be greater than that of V—a contradiction, since all weights are the same. Thus, T cannot have any major gap. T is, therefore, a leaf AVL-tree. □

Let T be a weighted leaf AVL-tree. Let there be a major gap in the tree between U and V, which is supported by W. The following are some useful definitions related to the rank of a node, the major gap between U and V, and how the major gap is supported.

A node is *higher* (resp., *lower*) than another node if the rank of the former is greater (resp., smaller) than that of the latter.

The *rank* of a subtree (in T) is the rank of its root.

The major gap (between U and V) is a *left* major gap *at* U if V is the left son of U; otherwise, it is a *right* major gap *at* U.

If W is a brother or a son of a brother of V, then W supports the major gap *directly*; otherwise, W supports the major gap *indirectly*.

The left and the right neighboring leaf nodes of V are the *left* and the *right neighboring leaf nodes* of the major gap, respectively.

The *width* of the major gap (between U and V) is $rk(U) - rk(V) - 2$. The *r-width* of the major gap is the larger of zero and $\min\{rk(U) - rk(V) - 2, rk(U) - r - 2\}$. Note

that if r is the larger of the ranks of the neighboring leaf nodes of V, then the major gap is supported if and only if its r-width is zero. Also, if r is $rk(V)$ or less, then the r-width of the major gap is the same as its width.

Example 2.2. Consider the weighted leaf AVL-tree shown in Fig. 2.1. The major gap between A and C is a right major gap at A; D is its right neighboring leaf node. Both of the major gaps at A and F are indirectly supported. The width of the major gap at A is 3, while its 10-width is zero.

An upper bound on the access time. We now show, through a sequence of results, that the access time of any item in a weighted leaf AVL-tree is ideal. The general strategy that we use for proving this result is as follows:

(a) We relate the desired upper bound on the time needed for accessing item i in a weighted leaf AVL-tree storing n items with weights w_i, $1 \leqq i \leqq n$, to (i) $F(w_i)$ (defined earlier), and to (ii) an upper bound on the "height" of an "almost two-dimensional leaf AVL-tree" with $W = \Sigma_i w_i$ "1-level" leaf nodes, implied by the fact that $F(w_i)$ is an upper bound on the height of a leaf height-balanced tree with height unbalance (i.e., the difference in the heights of subtrees at a node in the tree) up to 4 and w_i leaf nodes (Lemma 2.2).

(b) We show that an upper bound on the height of a "modified two-dimensional leaf AVL-tree" with n "1-level" leaf nodes is also an upper bound on the height of an "almost two-dimensional leaf AVL-tree" with the same number of "1-level" leaf nodes (Lemma 2.4).

(c) We show how an upper bound on the height of a "modified two-dimensional leaf AVL-tree" with n "1-level" leaf nodes can be derived from an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 [7] and n leaf nodes (Lemma 2.7).

(d) We derive an upper bound on the height of a "modified two-dimensional leaf AVL-tree" with W "1-level" leaf nodes implied by the fact that $F(w_i)$ is an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and w_i leaf nodes. The desired result is then derived using Lemmas 2.2, 2.4 and 2.7 (Theorem 2.8).

An *almost two-dimensional leaf AVL-tree* (*a2AVL-tree*) is a leaf binary tree with the following extensions and restrictions to its structure:

(a) Its nodes are called 2-*level* nodes; its leaf nodes further point to leaf binary trees which are called its 1-*level* subtrees. The nodes of 1-level subtrees are called 1-*level* nodes. V is a (left/right) *son* of U only if either both U and V are 2-level nodes or both U and V are 1-level nodes. (Observe that a path in an a2AVL-tree contains a certain number of 2-level nodes followed by a certain number of 1-level nodes.)

(b) Each of the 1-level subtrees is a leaf height-balanced tree with height unbalance up to 4 [7].

(c) The *rank* of its 2-level node is the height[3] of the node.

(d) Every major gap formed by its 2-level nodes is "supported" (as defined earlier) by a 2-level leaf node.

An a2AVL-tree can be used for storing items (and searching them); the above definition, however, specifies only its structure.

Example 2.3. Figure 2.2 shows an a2AVL-tree. Nodes A through I are 2-level nodes. The rest of the nodes are 1-level nodes. The rank of each 2-level node is the

[3] The definitions of the height of an a2AVL-tree and the height of a node in an a2AVL-tree are the same as for leaf binary trees. (See Example 2.3.)

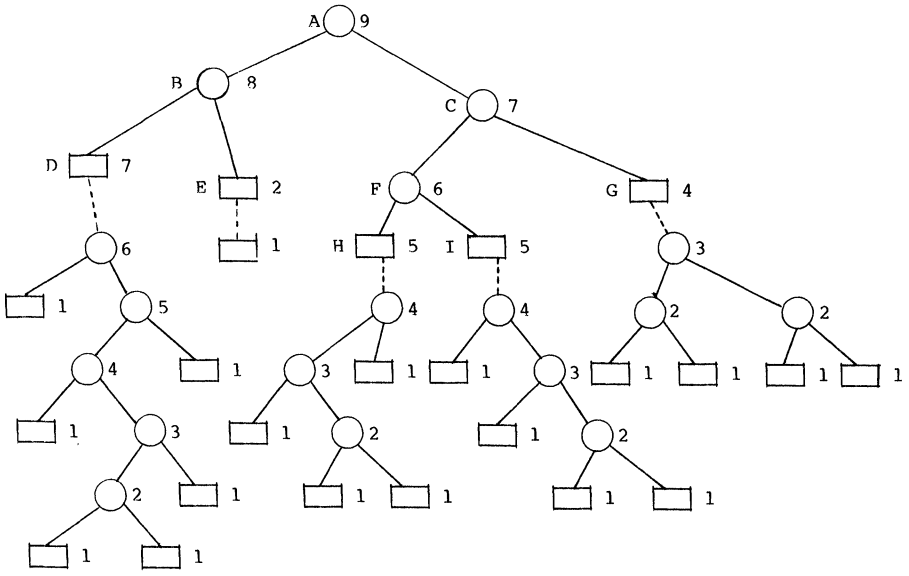


FIG. 2.2. *An almost two-dimensional leaf AVL-tree. The height of each node is shown on its right side.*

same as its height. The only major gaps formed by its 2-level nodes are between B and E, and between C and G. Both of these major gaps are directly supported.

Let $A(i)$ be the access-time of item i in a weighted leaf AVL-tree T_1 which stores n items, item i having weight w_i. Let $W = \Sigma_i w_i$.

Let $U(W)$ be an upper bound on the height of an a2AVL-tree with W 1-level leaf nodes, implied by the fact that $F(n)$ $(= \lfloor 2.464 \log_2 (n+1.5) - 0.56 \rfloor)$ is an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and n leaf nodes.

LEMMA 2.2. $A(i) = O(U(W) - F(w_i))$.

Proof. Let T_2 be an a2AVL-tree such that (a) it stores in its 2-level leaf nodes the same weighted items as T_1 (item i with weight w_i being stored in the ith (from the left) 2-level leaf node), and (b) the number of 1-level leaf nodes in the subtree pointed to by the 2-level leaf node storing item i, $1 \leqq i \leqq n$, is w_i.

The height of T_2 is bound from above by $U(W)$ even if its 2-level leaf node storing item i, $1 \leqq i \leqq n$, is "placed" at height $1 + F(w_i)$. If we actually do such placing of its 2-level leaf nodes, then its 2-level nodes constitute a weighted leaf AVL-tree which stores the same weighted items as T_1. The result follows. □

Let P be a 2-level nonleaf node at height h in an a2AVL-tree. The *H-condition* of P is "r" if either (a) one of the sons of P is a (2-level) leaf node at height $h-1$, or (b) the left son of the right son of P or the right son of the left son of P is a (2-level) leaf node at height $h-2$. Observe that if a major gap between U and V (V, a son of U) is directly supported, then the H-condition of U must be "r."

A *modified two-dimensional leaf AVL-tree* (*m2AVL-tree*) is an a2AVL-tree with the following modifications: (a) 2-level nodes U and V, such that V is a son of U, form a major gap only if the heights (i.e., ranks) of U and V differ by more than 5, and (b) every major gap in the tree is directly supported.

It is easy to see than an m2AVL-tree can alternatively be defined as a leaf binary tree (all its nodes being called 2-*level* nodes) whose 2-level leaf nodes further point to leaf height-balanced trees (whose nodes are called 1-*level* nodes) with height

unbalance up to 4, such that either the H-condition of a 2-level nonleaf node in the tree is "r" (the definition of the H-condition is carried over from a2AVL-trees) or the difference in the heights of the subtrees at the node is at most 4.

LEMMA 2.3. *Given an a2AVL-tree with n 1-level leaf nodes, it can be transformed into an m2AVL-tree with the same height and n or fewer 1-level leaf nodes.*

Proof. See Appendix A. □

LEMMA 2.4. *An upper bound on the height of an m2AVL-tree with n 1-level leaf nodes is also an upper bound on the height of an a2AVL-tree with the same number of 1-level leaf nodes.*

Proof. The proof follows from Lemma 2.3. □

Let $N(h, 2)$ be the least number of 1-level leaf nodes in an m2AVL-tree of height h. Similarly, let $N(h, 1)$ be the least number of leaf nodes in a leaf height-balanced tree of height h with height unbalance up to 4.

LEMMA 2.5. $N(h, 2) \geqq N(h-3, 1)+1$, $h \geqq 2$.

Proof. See Appendix B. □

LEMMA 2.6. $F(n) = \lfloor 2.464 \log_2 (n+1.5) - 0.56 \rfloor$, *is an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and n leaf nodes.*

Proof. $2.464 \log_2 (n+2.5) - 1.56$ is an upper bound on the height of a (standard) height-balanced tree with height unbalance up to 4 and n nodes [7]. An upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and n leaf nodes is therefore $2.464 \log_2 (n-1+2.5) - 1.56+1$ (note that a leaf tree with n nodes has $n-1$ nonleaf, i.e., internal nodes). Hence the result. □

LEMMA 2.7. *If $k_1 \log_2 (n+k_2) - k_3$ is an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4 and n leaf nodes, then $k_1 \log_2 (n+k_2-1) - k_3+3$ is an upper bound on the height of an m2AVL-tree with n 1-level leaf nodes.*

Proof. $N(h-3, 1)$ is the least number of 1-level leaf nodes in a leaf height-balanced tree of height $h-3$ with height unbalance up to 4. Thus,

$$\begin{aligned} h-3 &\leqq k_1 \log_2 (N(h-3, 1)+k_2) - k_3 \\ &\leqq k_1 \log_2 (N(h, 2)+k_2-1) - k_3, \end{aligned}$$

because $N(h-3, 1) \leqq N(h, 2)-1$ (Lemma 2.5). The result follows. □

THEOREM 2.8. *A weighted leaf AVL-tree has ideal access-time for all items.*

Proof. From Lemmas 2.6 and 2.7, $\lfloor 2.464 \log_2 (W+0.5)+2.44 \rfloor$, where $W = \Sigma_i w_i$, is an upper bound on the height of an m2AVL-tree with W 1-level leaf nodes, implied by the fact that $F(w_i)$ is an upper bound on the height of a leaf height-balanced tree with height unbalance up to 4. From Lemma 2.4, this is also a value for $U(W)$ (defined earlier).

From Lemma 2.2, the following equality applies to $A(i)$, the time needed for accessing item i in a weighted leaf AVL-tree storing n items (item i having weight w_i):

$$\begin{aligned} A(i) &= O(U(W) - F(w_i)) \\ &= O(\lfloor 2.464 \log_2 (W+0.5)+2.44 \rfloor - \lfloor 2.464 \log_2 (w_i+1.5) - 0.56 \rfloor) \\ &= O(\log_2 (W/w_i)). \end{aligned}$$

□

3. Rank-increase restructuring algorithm. Let T be a weighted leaf AVL-tree. Let the rank of a leaf node P be increased by a positive integer x (see Fig. 3.1). As a result, T may no longer be a valid weighted leaf AVL-tree. In this section, we describe and analyze an algorithm, *Algorithm* I, for restructuring T into a weighted leaf AVL-tree. We intersperse the description of the algorithm with explanations and results which have bearing on the correctness of the algorithm.

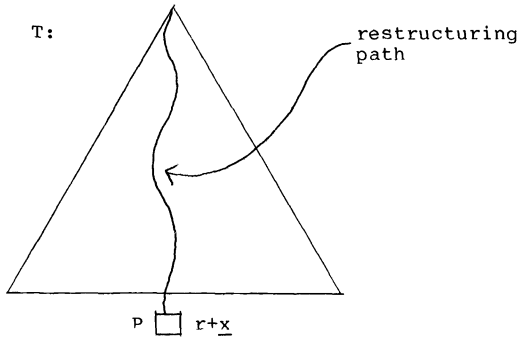


FIG. 3.1. *Rank of P has been increased by x to r+x.*

Let the path between P and the root of T be called the *restructuring path* (see Fig. 3.1). This path is traversed starting from the father of P. As each node is visited, the rank of the "current" node (i.e., the node visited) is updated, and an "action" is performed according to the *restructuring rules* given below. The action is to terminate the algorithm, to simply "climb" (i.e., to go up on) the restructuring path with a rank increase, or to "rebalance" the subtree rooted at the current node into a weighted leaf AVL-tree followed by either terminating the algorithm or climbing the restructuring path with a rank increase. The algorithm is terminated if the restructuring path cannot be climbed any further. The *rebalancing operations* used in this algorithm are *Rotation* and *Double Rotation* (Fig. 3.2 and its symmetric variant). These are the same operations as used in the AVL-tree restructuring [2]. Note that rebalancing at a node may change the nodes on the restructuring path, but the restructuring path will always be the upward path from P.

A basic principle behind this algorithm is that as each node is visited, the subtrees at the node are valid weighted leaf AVL-trees. We call this as the *invariant property* of the algorithm.

We prove results (Lemmas 3.2 and 3.4) assuming that the invariant property is true at the current step. These results imply that the restructuring rules *maintain* the property for the next step, if any. Thus, the property can be proved by induction.

In stating the restructuring rules, we will consider as understood the fact that the algorithm is terminated if the rule prescribes visiting the next higher node which, however, does not exist.

If the rank of the current node does not get increased, then there is no change to the current node as well as to the nodes above it on the restructuring path, as far as major gaps are concerned. Thus, we have the first restructuring rule:

Rule 1. If the rank of the current node does not increase, then the algorithm is terminated.

If the rank of the current node does get increased by z, $z \geqq 1$, there are three possibilities: (a) There still may not be a major gap (at the current node). (b) A new major gap may be created (where there was none before). (c) The width of an existing major gap may be increased.

If (a) is the case, then the restructuring path must simply be climbed with a rank increase of z. In such a case, we have the following lemma.

LEMMA 3.1. *If there is no major gap at the current node in spite of an increase in its rank by z, then z must be* 1.

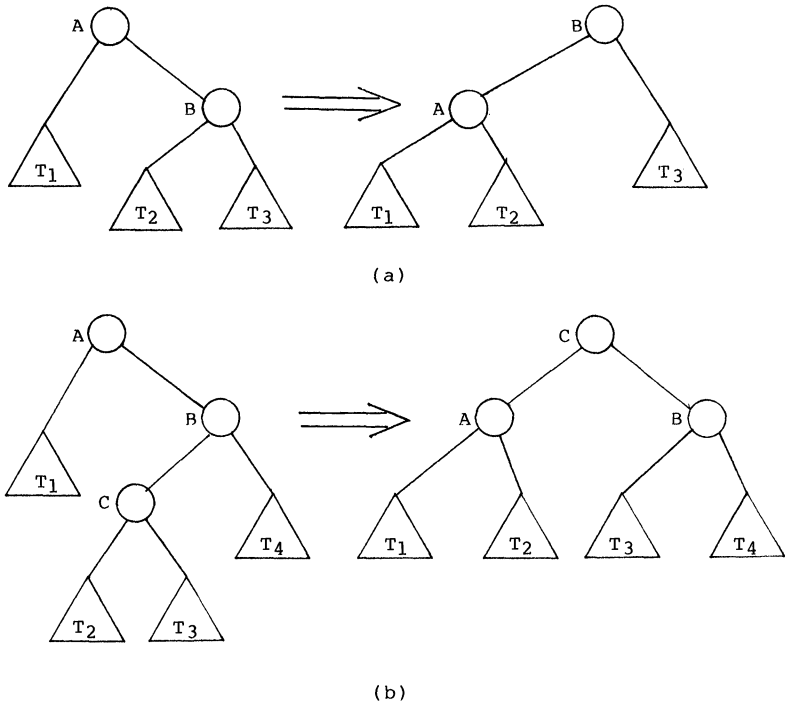


FIG. 3.2. (a) *Rotation at A.* (b) *Double rotation at A.*

Proof. Figure 3.3 and its symmetric variant show the only situations where an increase in the rank of the current node does not result in a major gap. Note that the rank of T_1 cannot be more than $r-1$. Thus, an increase in the rank of A by more than 1 will create a left major gap at A. □

If (b) or (c) is the case, then there is actually a need for rebalancing at the current node only if the major gap is not supported. To keep the algorithm simple and efficient, we, however, insist that a rebalancing be performed at the current node unless the major gap is directly supported. The rules according to which no rebalancing is performed at the current node, in spite of an increase in its rank, are, therefore, as follows.

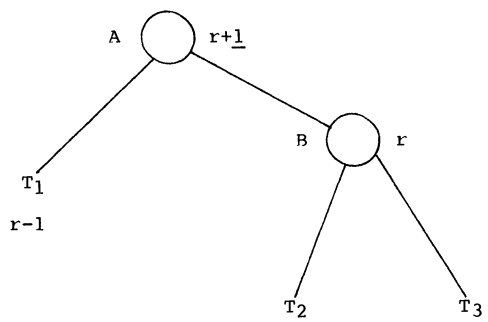


FIG. 3.3. *Increase in the rank of A (by 1) to r+1 does not result in a (left) major gap at A. T_1, T_2 and T_3 are the names of subtrees. Rank of T_1 is $r-1$.*

Rule 2. If the rank of the current node increases (by 1) but there is still no major gap at the node, then no rebalancing is performed. The father of the current node is visited next with an increase in the rank of its son by 1.

Rule 3. If the rank of the current node increases by z, $z \geqq 1$, and there is a major gap at the current node that is directly supported, then no rebalancing is performed (see Fig. 3.4). The father of the current node is visited next with a rank increase of its son by z.

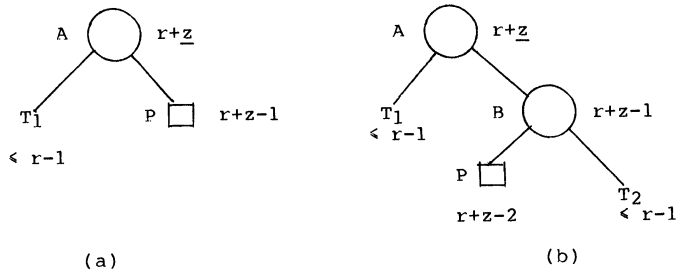


FIG. 3.4. *Major gap at A, if any, is directly supported.*

Now, we are left with the case where a rebalancing operation must be applied at the current node. We must, however, specify whether rotation or double rotation must be applied. We make a distinction between the cases, z (the rank increase of the current node) $= 1$, and $z > 1$. The corresponding rules are as follows.

Rule 4. If $z = 1$ and there is a major gap at A, the current node, which is not directly supported, then rotation (see Fig. 3.5) is applied at A if the situation is as shown in Fig. 3.5(a) or its symmetric variant. Double rotation (see Fig. 3.6) is applied if the situation is as shown in Fig. 3.6(a) or its symmetric variant. The father of the current node is visited next, with an increase in the rank of its son by 1, if rotation has been applied and it has not decreased the rank of the subtree ($u = r+1$, in Fig. 3.5). In all other cases, the algorithm is terminated.

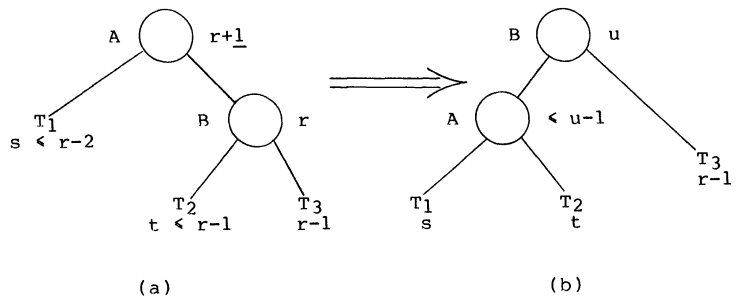


FIG. 3.5. *Rotation at A. Rank of A has increased (by 1) to $r+1$. There is a left major gap at A (in (a)) which is not directly supported. $u = r+1$, if $t = r-1$; otherwise $u = r$.*

Rule 5. If $z > 1$ and there is a major gap at A which is not directly supported, then rotation is applied at A if the "direction of the last two steps is the same," i.e., if the two tree edges on the restructuring path immediately below A are both left edges or both right edges. Double rotation is applied if the direction of the last two steps is opposite. (Figures 3.7, 3.8 and their symmetric variants show the application of this

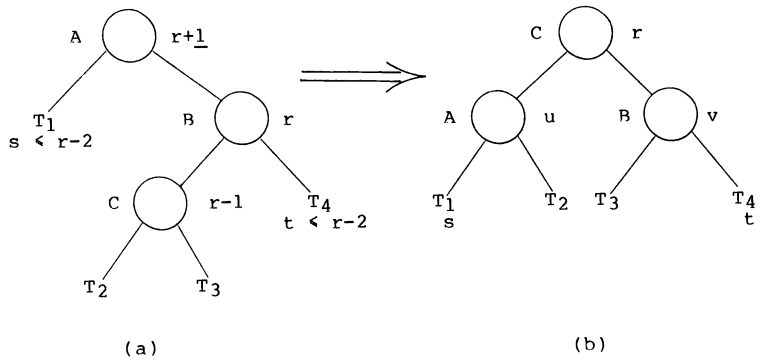


FIG. 3.6. *Double rotation at A. Rank of A has increased (by 1) to r+1. There is a left major gap at A in* (a) *which is not directly supported. Either u or v is r−1.*

rule.) The father of the current node is visited next, with an increase of $z-1$ in the rank of its son.

Notice the similarity of Rule 5 to that of the corresponding rule in the insertion algorithm for AVL-trees [2]. Rotation is applied if the direction of the last two steps is the same; double rotation is applied otherwise. Except for one special case, the same

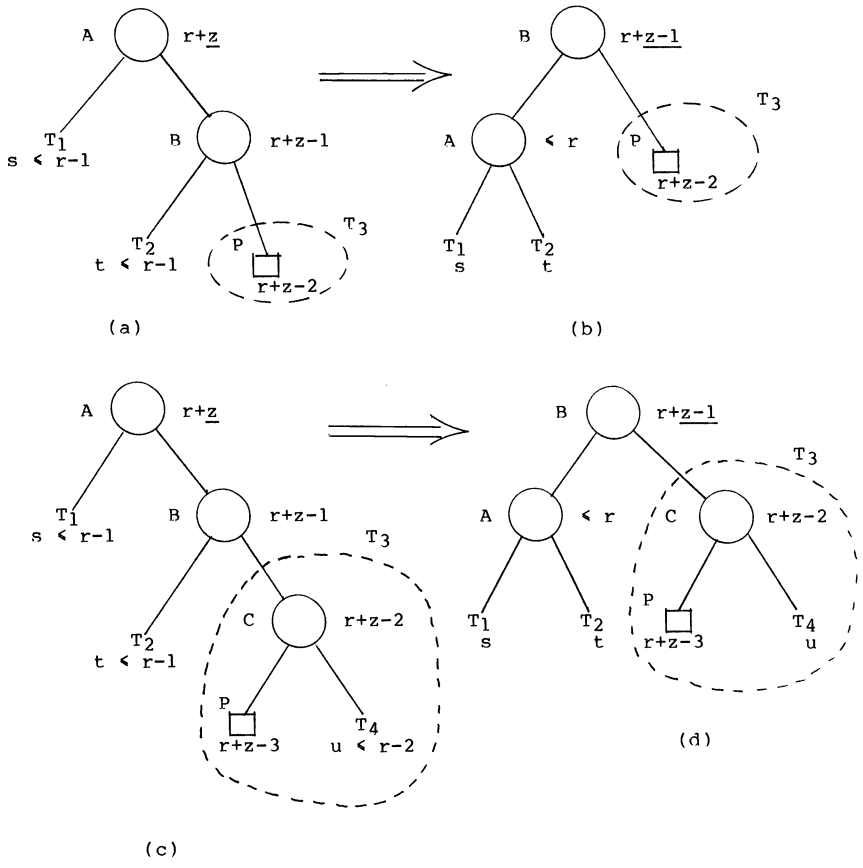


FIG. 3.7. *Rotation at A. Rank of A has increased (by z, z>1) to r+z. There is a left major gap at A* (*in* (a)) *which is not directly supported. Increase in the rank of the rebalanced subtree is z−1.*

can be said for the case $z=1$ (Rule 4). This special case is the following. The ranks of the subtrees at B (see Fig. 3.5) are the same. If this is the case, then rotation rather than double rotation is applied even if the direction of the last two steps is opposite (i.e., if the restructuring path passes through T_2). This is also the case responsible for no decrease in the rank of the rebalanced subtree. It might be of interest to note that this case can occur if a rebalancing operation has been applied in the previous step, and the current node in the previous step had a rank-increase of 2 which resulted in a major gap at the node with a width of 1. Observe that if in Fig. 3.7(a) or (c), $z=2$ and $s=r-1$, then in the respective rebalanced subtrees, the ranks of the subtrees at B are the same. A similar observation can be made for Fig. 3.8.

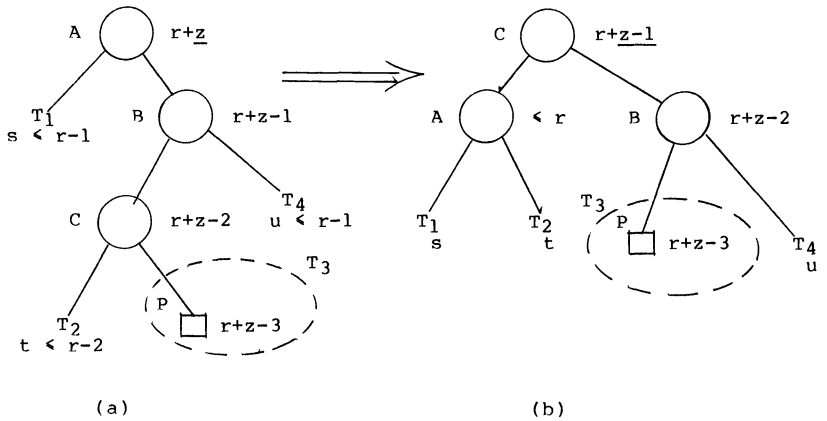


FIG. 3.8. *Double rotation at A. Rank of A has increased (by z, $z>1$) to $r+z$. There is a left major gap at A (in* (a)) *which is not directly supported. Increase in the rank of the rebalanced subtree is $z-1$.*

Rule 4 is *correct* if the following is true:

(a) The application of this rule can be represented by Fig. 3.5, 3.6 or a symmetric variant;

(b) The rebalanced subtree is free from any possibly unbalanced major gaps.

LEMMA 3.2. *Rule 4 is correct.*

Proof.

Case 1. *Rotation is applied* (see Fig. 3.5(b)). The rank of B is either $r+1$ (if t, the rank of T_2, is $r-1$) or r (if $t \leqq r-2$). In the former case, there is no major gap between B and A. In the latter case, there is a major gap between B and A only if $s \leqq r-4$ and $t \leqq r-4$. But then there is a left major gap at B (Fig. 3.5(a)) which must be supported by its right neighboring leaf node. This leaf node will support the major gap between B and A. All other major gaps continue to be supported.

Case 2. *Double rotation is applied* (see Fig. 3.6(b)). The rank of T_2 or T_3 (or both) is $r-2$. Thus, the rank of C is r. If the rank of each of T_2 and T_3 is $r-2$, then $u=r-1$, $v=r-1$. In such a case, no new major gaps get created and no (already existing) major gaps get widened. In fact, irrespective of the ranks of T_2 and T_3, the widths of any major gaps at A (before its rank increase) and B (Fig. 3.6(a)) get reduced in the rebalanced subtree.

Suppose the rank of T_2 is $r-2$ and that of T_3 is $r-3$ or less. There is a major gap between C and B only if the rank of T_3 is $r-4$ or less and $t \leqq r-4$. If this is the

case, then there is a right major gap at B (Fig. 3.6(a)) which must be supported by its right neighboring leaf node. This leaf node will also support the right major gap at C.

Suppose, the rank of T_2 is $r-3$ or less and that of T_3 is $r-2$. There is a major gap between C and A only if $s \leqq r-4$ and the rank of T_2 is $r-4$ or less. If this is the case, then there is a left major gap at A (Fig. 3.6(a)) even without the rank increase of A. This major gap must be supported by its left neighboring leaf node. This leaf node will also support the left major gap at C. □

Rule 5 applied at the current node is *correct* if it rebalances the subtree rooted at the node such that the resulting subtree has the following properties:

(a) The rank of the subtree is one less than what it was before;

(b) The subtree is free from any possibly unsupported major gaps.

A subtree of the tree being restructured is in *special form* if it is a leaf node, or if it can be represented as one of the trees shown in Fig. 3.9 or their symmetric variants.

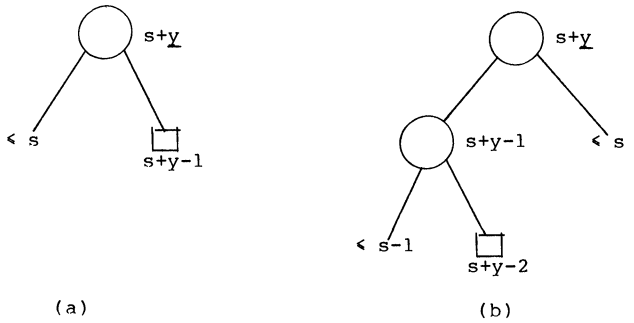


FIG. 3.9. *Trees in special form. y is the increase in the rank of the tree.*

The *form assumption is valid at the current node* if the subtree rooted at the son of the node on the restructuring path (e.g., the subtree rooted at B in Fig. 3.7(a), if A is the current node) is in special form.

We first prove that Rule 5 is correct, provided the form assumption is valid at the current node at the time of application of the rule. We later prove that this assumption is, in fact, valid (Corollary 3.5.1).

LEMMA 3.3. *If the rank of current node A increases by z, $z>1$, and the form assumption is valid at A, then the subtree rooted at A can be represented as one of the trees shown in Figs.* 3.4, 3.7(a) *and* (c) *and* 3.8(a) *and their symmetric variants.*

Proof. Let B be the son of A on the restructuring path. The subtree rooted at A can obviously be represented as one of the trees shown in Fig. 3.10 and their symmetric variants. Consider the tree shown in Fig. 3.10(d). Clearly, $s+y=r+z-1$ and $y \geqq z$. Thus, $s \leqq r-1$. This tree can, therefore, be represented as Fig. 3.8(a). The rest of the proof is similar, and is left to the reader. □

LEMMA 3.4. *If the form assumption is valid at current node A, then Rule* 5 *is correct and the rebalanced subtree is in special form.*

Proof. Let z be the rank increase of A. Since Rule 5 is applicable, $z>1$ and there is a major gap at A which is not directly supported. Thus, in view of Lemma 3.3, the subtree rooted at A can be represented as Fig. 3.7(a), (c), 3.8(a) or one of their symmetric variants. It is easy to see that Fig. 3.7 and its symmetric variant correctly represent the application of rotation at A. Similarly, Fig. 3.8 and its symmetric variant

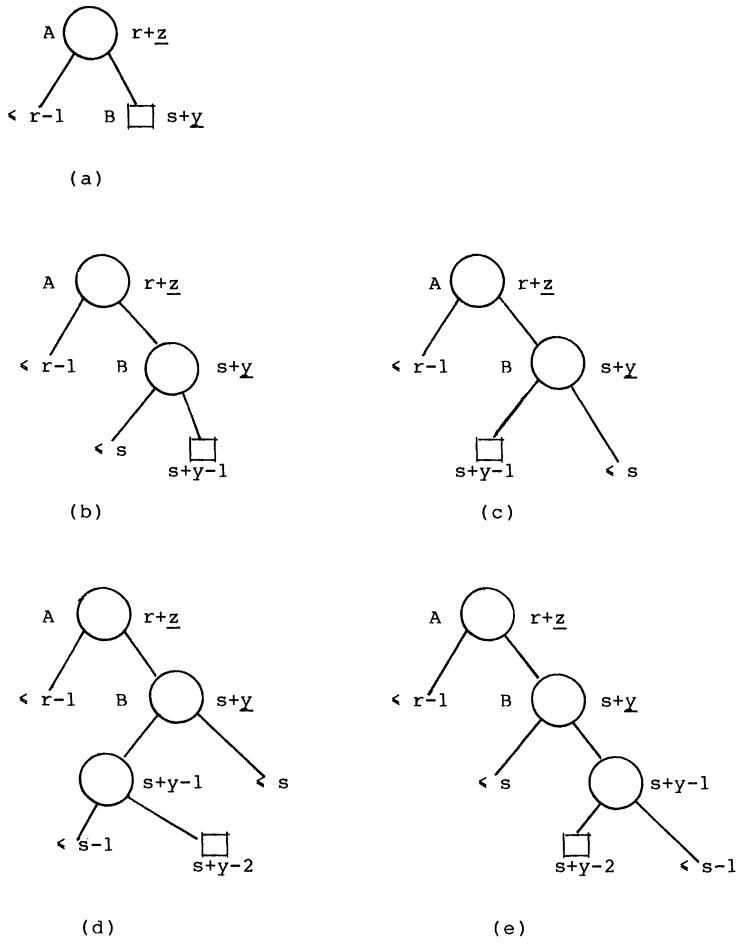


FIG. 3.10. *A is the current node. B is on the restructuring path.* $z > 1$, *and the form assumption is valid at A.*

correctly represent the application of double rotation at A. Observe that the rank of the rebalanced subtree decreases by 1, and the rebalanced subtree is in special form. Thus, the only thing that remains to be shown is that the rebalanced subtree is free from any possibly unsupported major gaps. This is, in fact, the case, because the only new major gap that may get created in the rebalanced subtree (between B and A in Fig. 3.7(b) or (d), or between C and A in Fig. 3.8(b)) is directly supported. □

LEMMA 3.5. *If the rank increase of current node N is greater than one, then the form assumption is valid at the node.*

Proof. (By induction on m, the number of nodes below the current node on the restructuring path.)

Basis. $m = 1$. In this case, the subtree rooted at the current node is as shown in Fig. 3.4(a) or its symmetric variant. The form assumption is clearly valid at the node.

Induction Hypothesis. The result is true for $1 \leqq m \leqq k$.

Induction Step. The result is true for $m = k + 1$.

Trace back to the immediate previous step, and consider "current node" A, in that step, with a rank increase of z. $z = 0$ implies that Rule 1 must have been applied, resulting in the termination of the algorithm—a contradiction. $z = 1$, implies that Rule

2, 3 (with $z = 1$) or 4 must have been applied. This would have led to either no increase in the rank of N or an increase by 1—a contradiction. Thus, $z > 1$, and Rule 3 (with $z > 1$) or 5 must have been applied at A. By the induction hypothesis, the form assumption must have been valid at A.

If Rule 3 has been applied, then the subtree rooted at A must have been as shown in Fig. 3.10(a) or 3.10(c) (which can be represented as Fig. 3.4(a) or 3.4(b), in view of Lemma 3.3), or a symmetric variant; such a subtree is clearly in special form. This proves the induction step.

If Rule 5 has been applied, then the subtree rooted at A must have been as shown in Fig. 3.10(b), (d) or (e) (or a symmetric variant), which in turn can be represented as Fig. 3.7(a), (c) or 3.8(a), in view of Lemma 3.3 In each case, the rebalanced subtree (Fig. 3.7(b), (d), 3.8(b) or a symmetric variant) is in special form (Lemma 3.4), which proves the induction step. □

COROLLARY 3.5.1. *The form assumption is valid at the current node at the time Rule 5 is applied.*

THEOREM 3.6. *Algorithm* I *is correct, i.e., it restructures the tree into a valid weighted leaf AVL-tree.*

Proof. The result follows from the invariant property, and the fact that the termination of the algorithm is correct (see Lemma 3.2). □

THEOREM 3.7. *The worst-case time complexity of Algorithm* I *is O(length of the restructuring path at the start of the algorithm).*

Proof. The time taken in applying a restructuring rule is constant. In the worst case, a restructuring rule may be applied at each node of the restructuring path, as it stands at the start of the execution of the algorithm. □

COROLLARY 3.7.1. *The rank of a leaf node P in a weighted leaf AVL-tree can be increased in O(time needed for accessing P), in the worst case.*

THEOREM 3.8. *Using Algorithm* I, *the number of rebalancing operations applied for restructuring a weighted leaf AVL-tree, in which the rank of a leaf node has been increased by* 1, *is at most* 1.

Proof. The proof follows from the algorithm. □

Example 3.1. The reader may verify that the tree shown in Fig. 3.11 is the result of applying Algorithm I to the tree shown in Fig. 2.1, after the rank of its leaf node, B, is increased to 19.

4. Rank decrease restructuring algorithm. Let T be a weighted leaf AVL-tree. Let the rank of a leaf node P be decreased by a positive integer x to $r - x$, where $r - x > 0$ (see Fig. 4.1). As a result, T may no longer be a valid weighted leaf AVL-tree. In this section, we describe and analyze an algorithm, *Algorithm* D, for restructuring T into a weighted leaf AVL-tree. As in the previous section, we intersperse the description of the algorithm with explanations and results which have bearing on the correctness of the algorithm.

Figure 4.1 (and its symmetric variant) introduces the following concepts related to T: *restructuring path*, *lower neighboring path*, *upper neighboring path*, *lower junction node*, and *upper junction node* (of P). Observe that J_l, the lower junction node, is the father of P (whose rank has decreased by x). J_l does not exist if and only if P is the only leaf node in T. J_u does not exist if P has only one neighboring leaf node (see Fig. 4.2). Figure 4.3 (and its symmetric variant) introduces the concepts of *activity paths* 1, 2, 3 and 4 (of P) in T. In this figure, all the activity paths exist, but this may not always be the case. The highest and the lowest node on an activity path are called its *head node* and *tail node*, respectively. Thus, in Fig. 4.3, L, K, N and U are the

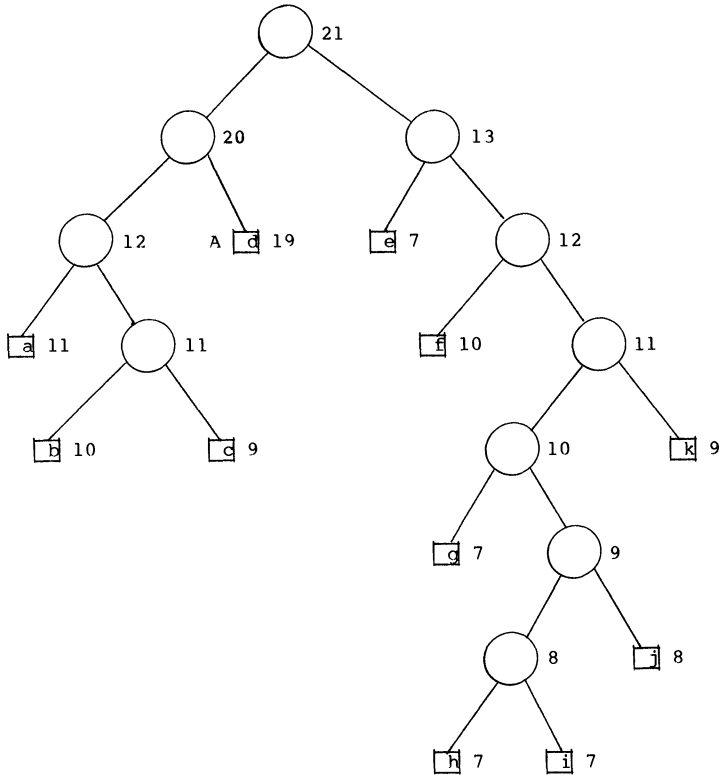


FIG. 3.11. *Result of applying Algorithm* I *to the tree shown in Fig.* 2.1, *after the rank of its leaf node, B, is increased to* 19.

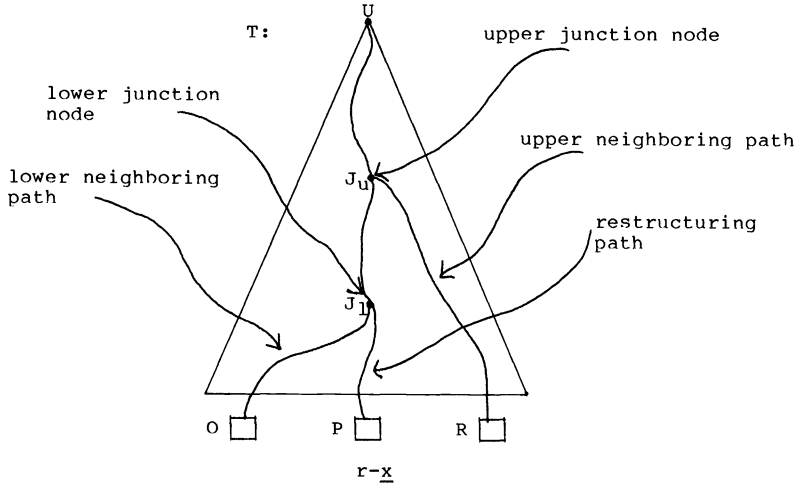


FIG. 4.1. *Rank of P has been decreased by x. Q and R are the neighboring leaf nodes of P. Paths, upwards from Q, P and R are the lower neighboring path, restructuring path and upper-neighboring path, respectively. P is a son of* J_l. J_l *and* J_u *are distinct nodes.*

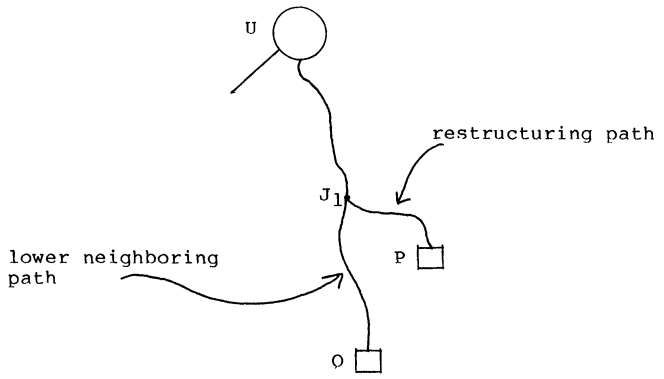


FIG. 4.2. *A special case. Upper junction node (and hence upper neighboring path) does not exist. P has only one neighboring leaf node.*

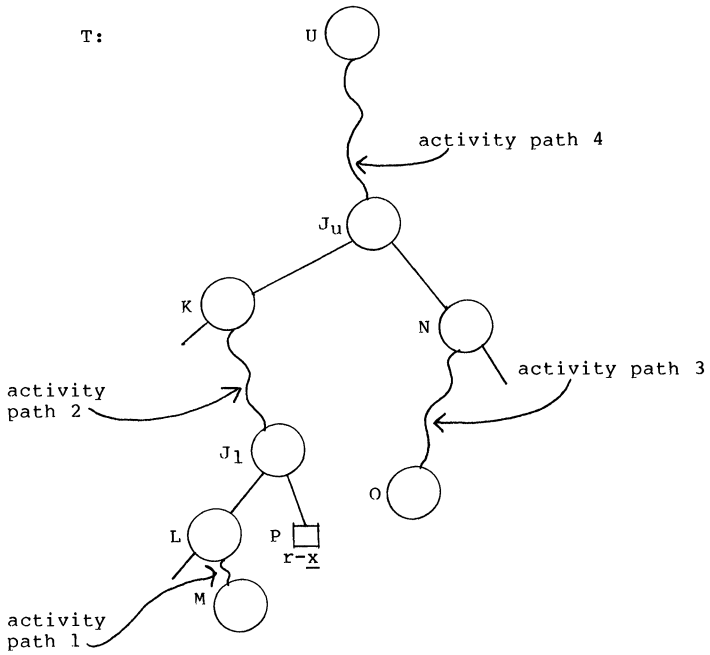


FIG. 4.3. *A curved line indicates zero or more tree edges. M is a nonleaf node on the lower neighboring path whose rank is closest to but greater than $r-x$. O is a nonleaf node on the upper neighboring path whose rank is closest to but greater than $r-x$. Paths between L and M, K and J_l, N and O, and U and J_u, are activity paths* 1, 2, 3 *and* 4, *respectively.*

head nodes of activity paths 1, 2, 3 and 4, respectively; M, J_l, O and J_u are the tail nodes of the activity paths 1, 2, 3 and 4, respectively.

The decrease in the rank of P may result in unsupported major gaps at nodes on some or all of the activity paths (see Fig. 4.4). Such major gaps can only be on the restructuring path, lower neighboring path, or upper neighboring path. Let us call major gaps on these paths at nodes of the activity paths as *relevant.*

Example 4.1. In the weighted leaf AVL-tree shown in Fig. 4.4, the rank of P has decreased from 7 to 2. There are relevant major gaps at L and M on activity path 1, and at J_l on activity path 2.

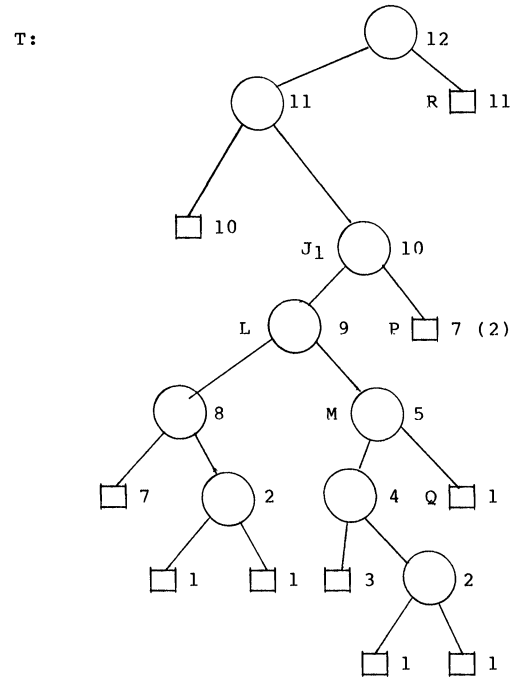


FIG. 4.4. *Decrease on the rank of P from 7 to 2 causes unsupported major gaps at nodes L and M (on activity path 1). It also causes an increase in the width of the right major gap at J_1 (which is on activity path 2).*

Activity paths 1, 2, 3 and 4 are "processed" in order. For processing an activity path, the path is traversed starting from its tail node. As node *A* (the "current" node) is visited, its rank is updated, and an "action" is performed according to the *restructuring rules* given below. The action is to terminate the processing, to climb the activity path without change, or to "rebalance" the subtree rooted at *A* followed by (a) visiting *A* again, if there still is a possibly unsupported major gap at *A*, and otherwise (b) visiting the father of *A*. Of course, the processing is terminated if the node to be visited next does not exist.

The rebalancing operations used in this algorithm are *rotation, double rotation* and *modified double rotation* (Figs. 3.2, 4.5 and their symmetric variants). We call the

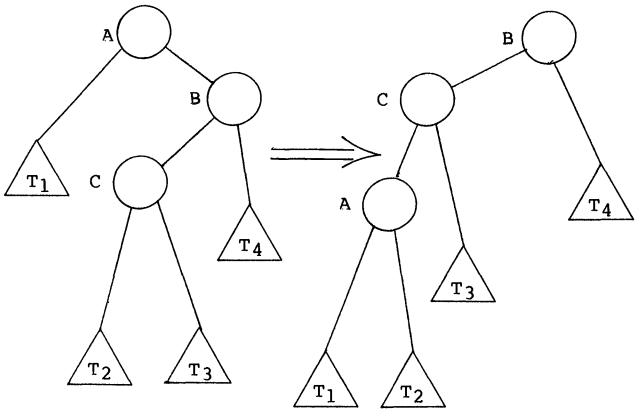


FIG. 4.5. *Modified double rotation at A.*

subtree resulting from an application of a rebalancing operation *rebalanced* even though there still may be an unbalanced major gap in the subtree. Observe that a rebalancing operation may "restructure" an activity path which, in this algorithm, means adding *new* nodes to the path (see Fig. 4.6). By an activity path (at any stage of the algorithm), we will thus mean a possibly restructured activity path. A node on the path is "original" if it is not new, i.e., if it was on the path at the start of the execution of the algorithm.

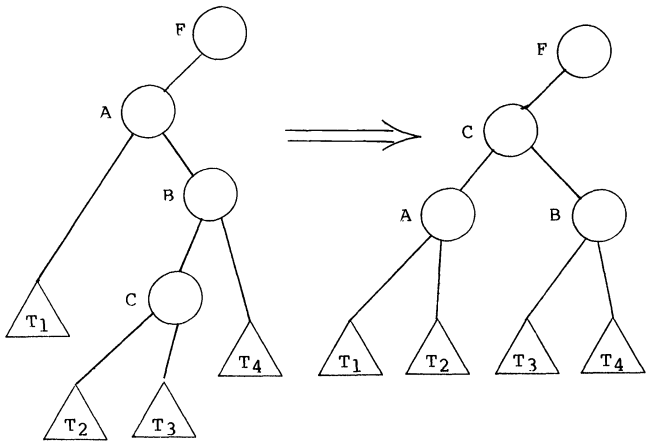


FIG. 4.6. *Activity path passes through* T_1. *After rebalancing, node C gets added to the (restructured) activity path.*

The *invariant property* of the algorithm is that as each node is visited (for the application of a restructuring rule), the subtrees at the node are valid weighted leaf AVL-trees.

We prove results (Lemmas 4.1 through 4.5) assuming that the invariant property is true at the current step. These results imply that the rules *maintain* the property for the next step, if any. Thus, the property can be proved by induction.

In stating the following restructuring rules, we consider as understood the fact that the processing (of the activity path) is terminated if the rule prescribes visiting a node that does not exist.

The following rule is for the situation where the major gap at the current node, if any, is supported. Observe that if the $(r-x)$-width of the relevant major gap at the current node is zero, then the major gap is supported by P (with rank $r-x$).

Rule 1. If one of the following is true:

(a) There is no relevant major gap at the current node,

(b) The relevant major gap at the current node is directly supported or its $(r-x)$-width is zero,

then the processing (of the activity path) is terminated if the rank of A, the current node, has not decreased and A is original; else the father of A is visited.

Observe that after Rule 1 is applied, the father of the current node is visited if the current node is new. We will explain this after introducing the following restructuring rule which rebalances the subtree rooted at the current node.

Rule 2. If there is a relevant major gap at A, the current node, which is not directly supported and whose $(r-x)$-width is not zero, then the following is done.

(i) (a) Rotation is applied if the situation is as shown in Fig. 4.7(a) or its symmetric variant.

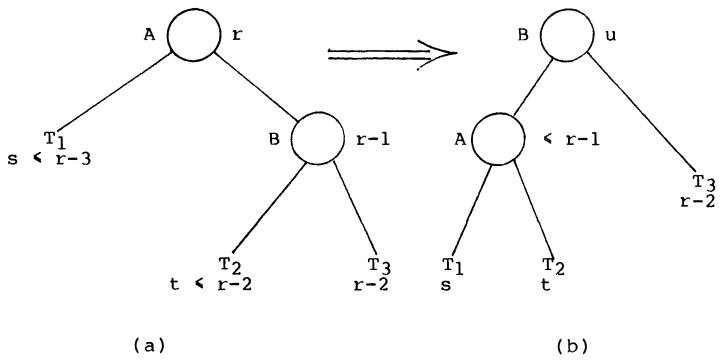


FIG. 4.7. *Rotation at A. There is a left major gap at A (in* (a)*) with nonzero* $(r-x)$*-width which is not directly supported.* $u = r$*, if* $t = r-2$*; otherwise,* $u = r-1$*. In* (b)*, there may be a possibly unsupported left major gap at A, which is on the (restructured) activity path.*

(b) Double rotation is applied if the situation is as shown in Fig. 4.8(a) or its symmetric variant.

(c) Modified double rotation is applied if the situation is as shown in Fig. 4.9(a) or its symmetric variant.

(ii) If there still is a relevant major gap at A in the rebalanced subtree which is not directly supported and whose $(r-x)$-width is not zero, then A is visited again. Otherwise, the father of A is visited next.

Let A, the current node, be visited for the first time, and Rule 2 be applied at the node. Let F be the father of A. F is obviously an original node. Note that after rebalancing the subtree rooted at A, A is visited again if there still is a relevant major gap with nonzero $(r-x)$-width at the node which is not directly supported. This process will continue until there is no relevant major gap at A, or there is one but it is directly supported, or the $(r-x)$-width of the major gap is zero. At such a time, there will be a certain number of new nodes between A and F (see Figs. 4.7-4.9). F must be eventually visited because there may have been a decrease in the rank of the rebalanced subtree, at the time A was visited for the first time and Rule 2 applied. This explains why the father of the current node is visited if the current node is new, after Rule 1 is applied.

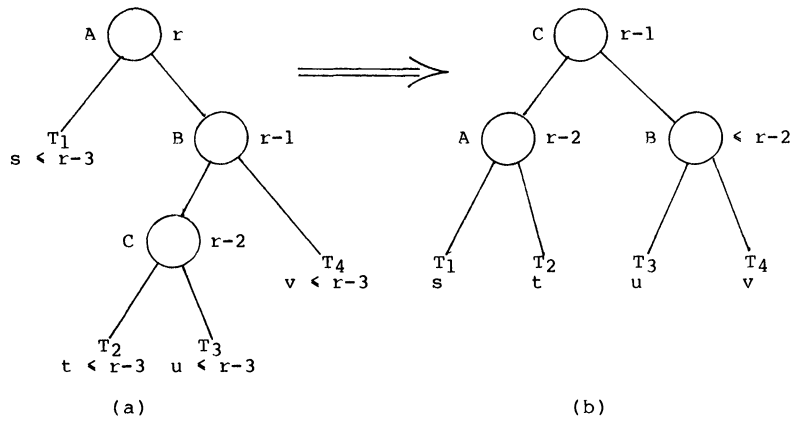


FIG. 4.8. *Double rotation at A. There is a left major gap at A (in* (a)*) with nonzero* $(r-x)$*-width which is not directly supported.* $s = r-3$ *or* $t = r-3$*. In* (b)*, there may be a possibly unsupported left major gap at A, which is on the (restructured) activity path.*

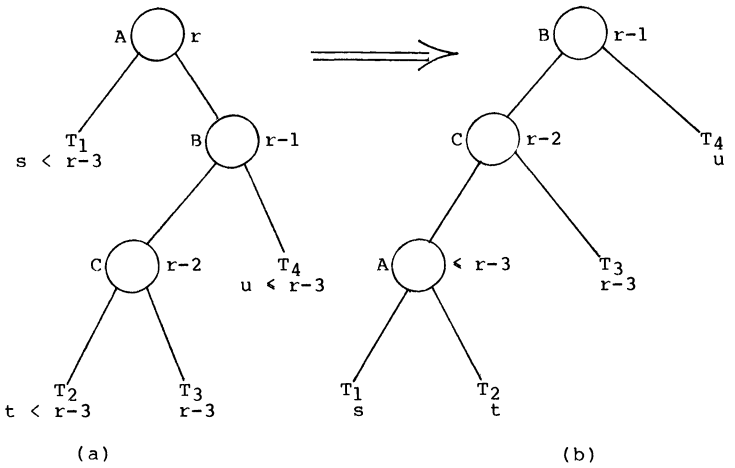


FIG. 4.9. *Modified double rotation at A. There is a left major gap at A (in* (a)*) with nonzero* $(r-x)$*-width which is not directly supported. In* (b)*, there may be a possibly unsupported left major gap at A, which is on the (restructured) activity path.*

A rebalancing operation applied in Rule 4 is *correct*, if (a) it can be represented as Figs. 4.7-4.9 or a symmetric variant; (b) the rebalanced subtree is a weighted leaf AVL-tree, except that a possibly unsupported relevant major gap may still remain at what was the current node but with reduced $(r-x)$-width; (c) at least one new node is introduced into the activity path above what was the current node.

LEMMA 4.1. *Rotation is correct.*

Proof. See Fig. 4.7(b). B is a new node on the (restructured) activity path above A. If the rank of A differs from that of B by 2 or more, then the rank of B must be $r-1$ and the rank of A must be $r-3$ or less. In such a case, $s \leqq r-4$ and $t \leqq r-4$. But then there is a left major gap at B (Fig. 4.7(a)) which must be supported by the right neighboring leaf node of the major gap, with rank $r-3$ or more. This leaf node directly supports the major gap between B and A, if any. The major gap at A may still be unsupported but its $(r-x)$-width has decreased. All other major gaps continue to be supported. □

LEMMA 4.2. *Double rotation is correct.*

Proof. See Fig. 4.8(b). Either $s=r-3$, or $t=r-3$; otherwise, modified double rotation must be applied. The rank of C (Fig. 4.8(a)) is $r-2$. Thus, either t or u is $r-3$. C is a new node introduced into the activity path above A. The only new major gap that may get introduced into the subtree is between C and B. This will happen only if $u \leqq r-5$ and $v \leqq r-5$. But then there is a right major gap at B (Fig. 4.8(a)) which must be supported by its right neighboring leaf node. This node supports the major gap between C and B. The relevant major gap at A may still be unsupported but its $(r-x)$-width has decreased. All other major gaps continue to be supported. □

LEMMA 4.3. *Modified double rotation is correct.*

Proof. See Fig. 4.9(b). B and C are new nodes introduced into the activity path above A. If the rank of A differs from that of C by 2 or more, then $s \leqq r-5$ and $t \leqq r-5$. In such a case, there is a left major gap at C (Fig. 4.9(a)) which must be supported by the right neighboring leaf node of the major gap, with rank $r-4$ or more. This leaf node directly supports the major gap between C and A, if any. The relevant major gap at A may still be unsupported but its $(r-x)$-width has decreased. All other major gaps continue to be supported. □

Rule 2 is *correct* if the rebalancing operation applied in the rule is correct.

LEMMA 4.4. *Rule* 2 *is correct.*

Proof. The proof follows from Lemmas 4.1-4.3. □

THEOREM 4.5. *Algorithm* D *is correct, i.e., the algorithm restructures the tree into a valid weighted leaf AVL-tree.*

Proof. The result follows from (a) the correctness of the invariant property; (b) the fact that it is correct to terminate the processing of an activity path if the rank of the current node (which is original) does not change and any major gap at the node is supported; (c) the repeated application of Rule 2 at a node cannot go on indefinitely (Lemma 4.4). □

THEOREM 4.6. *The worst-case time complexity of Algorithm* D *is* O(*sum of the lengths of the activity paths in the restructured tree*).

Proof. In the worst case, the restructuring rule is applied in order at each node of the restructured activity path, starting from its tail node. A restructuring rule (Rule 2) may, however, be applied at a node, say A, more than once; all such applications occurring one after another. But, for each such additional application of the rule, at least one new node is introduced into the activity path above A (Lemmas 4.1 through 4.3). Thus, the length of the restructured activity path is an upper bound on the number of such additional applications of Rule 2.

The result follows in view of the fact that an application of a restructuring rule takes constant time. □

Example 4.2. The reader may verify that the tree shown in Fig. 4.10 is the result of applying Algorithm D to the tree shown in Fig. 3.11, after the rank of its leaf node, A, is decreased to 9.

5. Update results. Let T be a weighted leaf AVL-tree. Let us use the following notation related to T:

W and $\bar{W}$ are the sum of the weights of the items (in T) before and after an update, respectively. w_i is the weight of item i. $\bar{w}_i$ is the weight of item i after its weight is changed. $i-$ is the largest item smaller than i, and w_{i-} is its weight. $i+$ is, similarly, the smallest item larger than i, and w_{i+} is its weight.

The update results given below use Algorithms I and D in either a direct or an adapted form.

THEOREM 5.1. *The weight of item* i *can be increased to* $\bar{w}_i$ *in* $O(\log_2(W/w_i))$ *time, in the worst case.*

Proof. Follows from Corollary 3.7.1. □

THEOREM 5.2. *The weight of item* i *can be decreased to* $\bar{w}_i$ *in* $O(\log_2(W/w_i) + \log_2(\bar{W}/\bar{w}_i))$ *time, in the worst case.*

Proof. Let R be the rank of the tree before the weight decrease, r be the rank of leaf node P storing item i, and x be the rank decrease of P because of the weight decrease of item i. From the definition of rank, $x = O(\log_2(w_i/\bar{w}_i))$. Item i is accessed in $O(\log_2(W/w_i))$ time. The tree is restructured using Algorithm D in $O(\log_2(\bar{W}/\bar{w}_i))$ time in the worst case, in view of Theorem 4.6; hence the result. □

THEOREM 5.3. *Item* i *can be inserted in* T *in* $O(\log_2(W/w_i) + \log_2(W/(w_{i-} + w_{i+})))$ *time, in the worst case.*

Proof. In order to insert item i, item $i-$ or $i+$ must be searched in order to find the place where item i must be inserted. Let us call the time taken in this phase the *location time.* The item is then inserted and the tree is restructured in the *restructuring time.* The total insertion time is the sum of the location time and the restructuring time.

Let us first discuss a method for insertion which is simple but whose time complexity is worse than what is claimed in the result (see Fig. 5.1(a)). This method

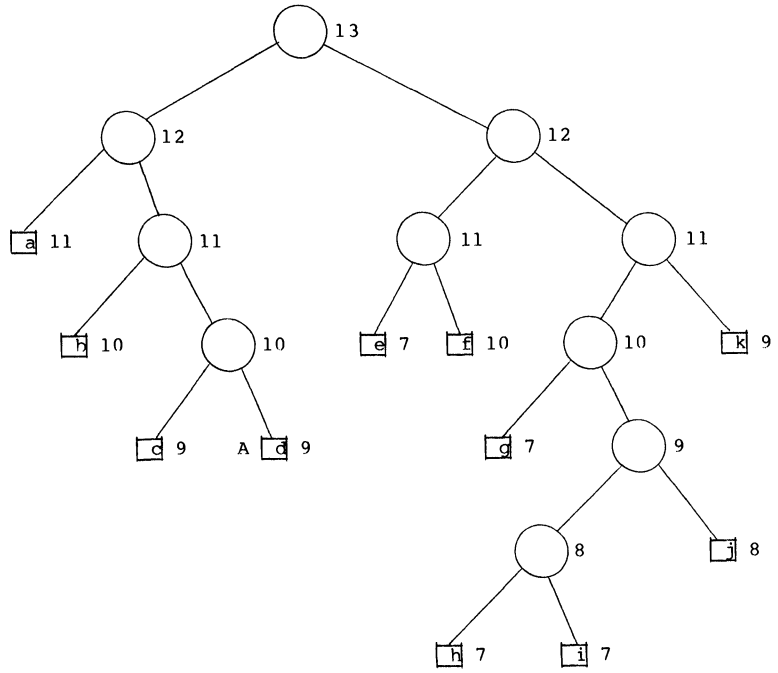


FIG. 4.10. *Result of applying Algorithm* D *to the tree shown in Fig.* 3.11, *after the rank of its leaf node, A, is decreased to* 9.

is based on the fact that Q cannot be supporting a major gap on the path between J_l and R.

Items $i-$ and $i+$ are searched concurrently descending level-by-level till Q, the lower of the two nodes storing items $i-$ and $i+$ is located. Thus, the location time is $O(\log_2 W/(\min\{w_{i-}, w_{i+}\}))$. Item i is then inserted (Fig. 5.1(a)). After insertion, the subtree rooted at P replacing leaf node Q will have a rank at least one greater than that of Q. The upward path from P is then traversed with the rank increase, using Algorithm I. The restructuring time is thus $O(\log_2(W/\min\{w_{i-}, w_{i+}\}))$.

The key to improving the time complexity is to insert at R, the higher of the two nodes Q and R (see Fig. 5.1(b)). A complicating factor, however, is that if the rank of S (storing item i) is less than that of R, then certain major gaps on the path between Q and J (the common ancestor of Q and R) may lose their support. For this reason, before performing the insertion, the affected "activity paths" are processed in appropriate order, "assuming" that the rank of R gets decreased by $x = (rk(R) - rk(S))$. This can be done in $O(\log_2(W/w_i))$ time, in view of Theorem 4.6. This is followed by performing the insertion and the needed restructuring using Algorithm I, in $O(\log_2(W/rk(R)))$ time. The result follows. □

THEOREM 5.4. *Item i in T can be deleted in* $O(\log_2(W/w_i) + \log_2(\bar{W}/(w_{i-} + w_{i+})))$ *time, in the worst case.*

Proof. Assume, without loss of generality, that the situation is as shown in Fig. 5.2. The deletion is performed using a variation of Algorithm D for decreasing the rank of P (storing item i) to $r - x$, where $x = \max\{\text{zero}, rk(P) - \max\{rk(Q), rk(R)\}\}$.

First, P and J_l are deleted. The rank of S (on activity path 2) is updated. If the updated rank of S is $r - x$ or more, then S is marked as the new tail node of activity path 2; otherwise, T, the father of S on activity path 2, is marked as the new tail node

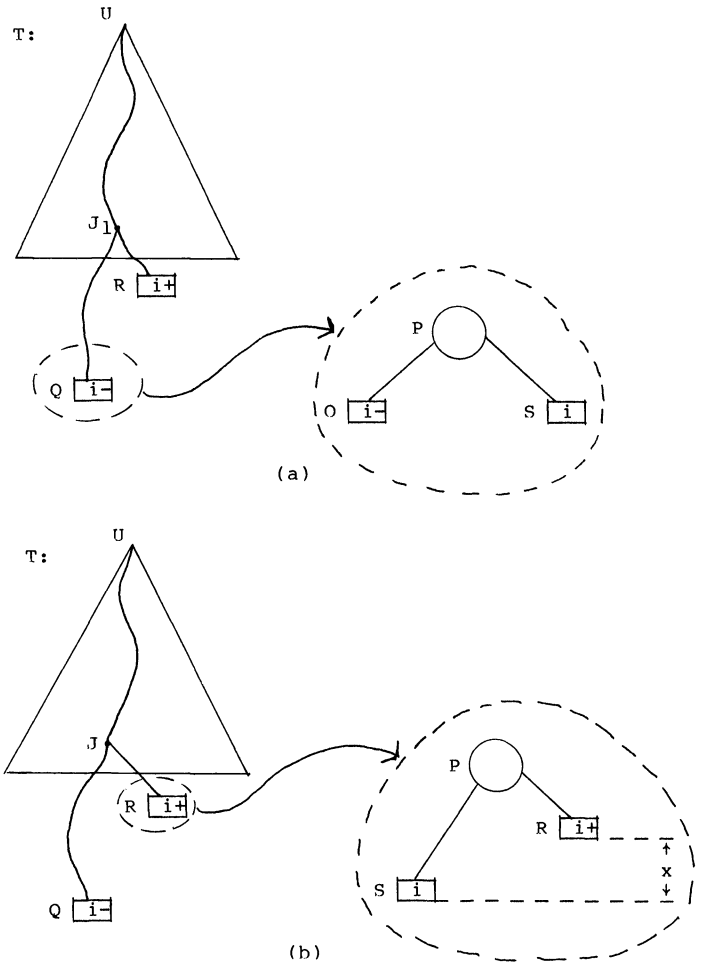


FIG. 5.1. *Two methods for inserting item i. Rank of Q is assumed to be less than R. In* (b), *x is zero or* $(rk(R)-rk(S))$, *whichever is larger.*

of the path. Activity paths 1, 2, 3 and 4 are now processed assuming that the rank of P has decreased to $r-x$; activity path 2 is processed starting at its new tail node.

After P and J_l are deleted, R is the neighboring leaf node of relevant major gaps on activity paths 1 and 2, and Q is the neighboring leaf node of relevant major gaps on activity path 3. If the rank of R is greater than that of Q, then $rk(R) \geqq r-x$. Similarly, if the rank of Q is greater than that of R, then $rk(Q) \geqq r-x$. Thus, it suffices to process activity paths 1, 2 and 3 as if the rank of a leaf node, which is a neighboring leaf node of relevant major gaps on activity paths 1, 2 and 3, has decreased to $r-x$. Again, while processing activity path 4, the rank of J_u cannot become less than $\max\{rk(Q), rk(R)\}$, and hence it cannot become less than $r-x$. Also, the $(r-x)$-width of the major gap at J_u being zero means that the major gap is supported by Q and R. Thus, it suffices to process activity path 4 as if the rank of a neighboring leaf node of the major gap at J_u, if any, is $r-x$.

The above arguments show the correctness of the deletion algorithm. In view of Theorem 4.6, the complexity of the algorithm is as stated in the result, if we can show that the following assertion is true: if T is made the new tail node of activity path 2,

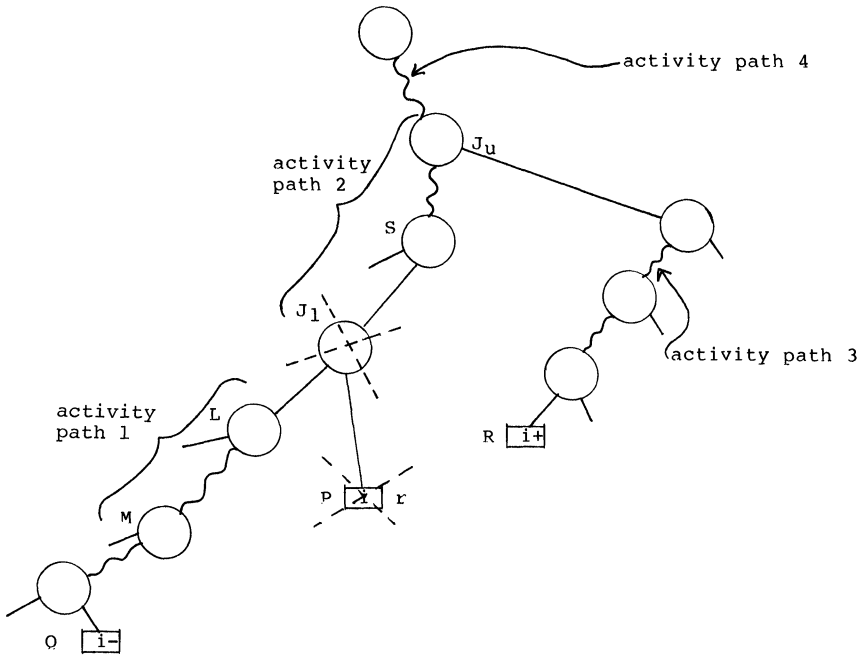


FIG. 5.2. *Deletion of item i stored in P. Q and R are the left and the right neighboring leaf nodes of P, respectively. P and J_l are deleted.*

then its updated rank, at the time it is visited first (for processing the path), is $r-x$ or more.

Suppose T is the new tail node of activity path 2 and the rank of T has been updated to a value less than $r-x(\leqq r)$ (see Fig. 5.3). This implies that the rank of S has been updated from at least $r+2$ to a value less than $r-x-1$. Consider the tree in its original form (before P and J_l are deleted). In this tree, there must be a left major gap at S because otherwise the rank of S cannot have been updated to a value less than $r-x-1(\leqq r-1)$. Again for the same reason, the left major gap at S must be supported by its left neighboring leaf node with rank r or more. Thus, there is a leaf node in the left subtree at T with rank r or more. It follows that the rank of T cannot

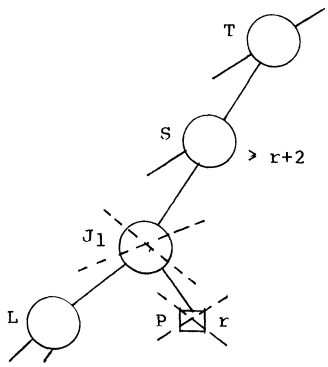


FIG. 5.3. *Proof of Theorem* 5.4.

have been updated to a value less than $r+1$—a contradiction, proving the assertion; hence the result. □

6. Concluding remarks. In this paper, we have provided a generalization of leaf AVL-trees to the weighted item case. We have proved an upper bound on the worst-case access time for an item and shown it to be ideal (logarithmic). The upper bound that we have proved is, however, not tight, and can be improved (see [12] for similar work).

We have presented two algorithms, Algorithm I and Algorithm D, which form the basis for the update results. Algorithm I is particularly fast. There is, however, scope for improving the efficiency of the algorithms. A major contribution of the paper is in analyzing these algorithms. A significant feature of the algorithms is that the "rebalancing" operations used are rotation, double rotation, and modified double rotation (a variation of double rotation). Thus, the algorithms retain to a large extent the restructuring simplicity of AVL-trees.

In terms of the worst-case time-complexities, weighted leaf AVL-trees provide an improvement over the available solutions [3]-[5], [8] and [9]. Specifically, the data structure matches the best known results for *all* of the five operations, access, insertion, deletion, promotion, and demotion (see Table 1). We also expect weighted leaf AVL-trees to support *join* and *split* operations efficiently; this is, however, under investigation.

TABLE 1

Comparison of the worst-case performances of the known data structures (including Weighted Leaf AVL-Trees) for the various operations. A "" beside a time-complexity indicates that it is the best known result.*

Data structure	Access	Insertion	Deletion	Promotion	Demotion
Globally Biased 2, b Trees [3]	A (*)	B	C (*)	D	D (*)
Globally Biased Binary Trees [3]	A (*)	B	C (*)	D	D (*)
Globally Biased a, b Trees [4]	E (*)	F (*)	F	G	H
2B-Trees [5]	I (*)	J	J	I (*)	K
D-Trees [8], [9]	A (*)	L	M	N	P
Weighted Leaf AVL-Trees	A (*)	Q (*)	C (*)	A (*)	D (*)

$A = O(\log_2(W/w_i))$. $B = O(\log_2(\bar{W}/w_i) + \log_2(W/(w_{i-} + w_{i+})))$.
$C = O(\log_2(W/w_i) + \log_2(\bar{W}/(w_{i-} + w_{i+})))$. $D = O(\log_2(W/w_i) + \log_2(\bar{W}/\bar{w}_i))$.
$E = O(\log_b(W/w_i))$. $F = O(\log_a(W/\min\{w_{i-} + w_{i+}, w_i\}))$.
$G = O(\log_a(\max\{W, \bar{w}_i\}/w_i))$. $H = O(\log_a(\max\{W, \bar{w}_i\}/\bar{w}_i))$.
$I = O(\log_{d+1}(W/w_i))$. $J = O(\log_{d+1} W)$.
$K = O(\log_{d+1}(W/\bar{w}_i))$. $L = O(\log_2(\bar{W}/w_0))$. $M = O(\log_2 W)$.
$N = O(\log_2(\bar{W}/w_i))$. $P = O(\log_2(W/\bar{w}_i))$. $O = O(\log_2(W/w_i) + \log_2(W/(w_{i-} + w_{i+})))$.
w_i = Weight of Item i. $\bar{w}_i$ = New weight of Item i. w_0 = weight of the smallest item.
w_{i-} = Weight of the largest item smaller than i. w_{i+} = Weight of the smallest item larger than i.
W = Sum of the weights (before update). $\bar{W}$ = Sum of the weights (after update).

We have generalized leaf AVL-trees rather than (standard) AVL-trees to the weighted item case. It is also possible to generalize (standard) AVL-trees in a similar manner. In fact, some work has already been done in this direction [10], [12], and further work is in progress.

Appendix A.

LEMMA (Lemma 2.3). *Given an a2AVL-tree T_1 of height H with n 1-level leaf-nodes, it can be transformed into an m2AVL-tree T_2 of the same height with n or less 1-level leaf nodes.*

Proof. T_1 is transformed into T_2 by successively applying the following transformation procedure to its 2-level nonleaf nodes, in decreasing order of their heights. T_2 is an m2AVL-tree of height H with n or less 1-level leaf nodes.

Transformation procedure. The procedure is applied to a 2-level nonleaf node P at height h provided it satisfies *Property* A which is stated as follows: There are no major gaps on the "left downward path from P" (the path between P and its leftmost (2-level) leaf descendant) and the "right downward path from P," which are indirectly supported.

The procedure transforms T_P, the tree rooted at P, to $\bar{T}_P$ of height h, such that the following is true in $\bar{T}_P$:

(a) Each (2-level) son[4] of P satisfies Property A.
(b) The subtrees T_a and T_b rooted at the sons of P are a2AVL-trees.
(c) $\bar{T}_P$ is an m2AVL-tree except that T_a and T_b are not necessarily m2AVL-trees.
(d) The number of 1-level leaf nodes in $\bar{T}_P$ is less than or equal to the number of 1-level leaf nodes in T_P.

Some of the transformations of the procedure *left-balance* or *right-balance* a subtree. The transformation for *right-balancing* a subtree rooted at a 2-level nonleaf node P (see Fig. A.1) is as follows:

Let U be the right neighboring leaf node of P, and let its height be t. Let Q be a 2-level nonleaf node on the right downward path from P, if it exists, whose height is greater than or equal to, but closest to, $t+1$.

Transformation.
if the height of the rightmost (2-level) descendant leaf node of P is less than t, *then*
  if the height of P is t or less, *then* replace the subtree rooted at P by that rooted at U,
  else replace the right subtree at Q by the subtree rooted at U.

It is easy to verify that, as a result of the transformation, either P is replaced by a leaf node or all major gaps in the subtree rooted at P are supported. Thus, any major gap on the right downward path from P is directly supported. Also, if there is a left major gap at Q which is directly supported (in the subtree before the transformation), the major gap will be directly supported in the transformed subtree.

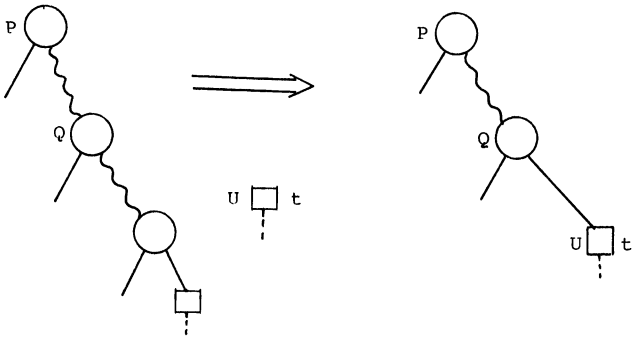


FIG. A.1. *A curved line indicates zero or more tree edges.*

[4] Recall that a 1-level node cannot be the (left/right) *son* of a 2-level node.

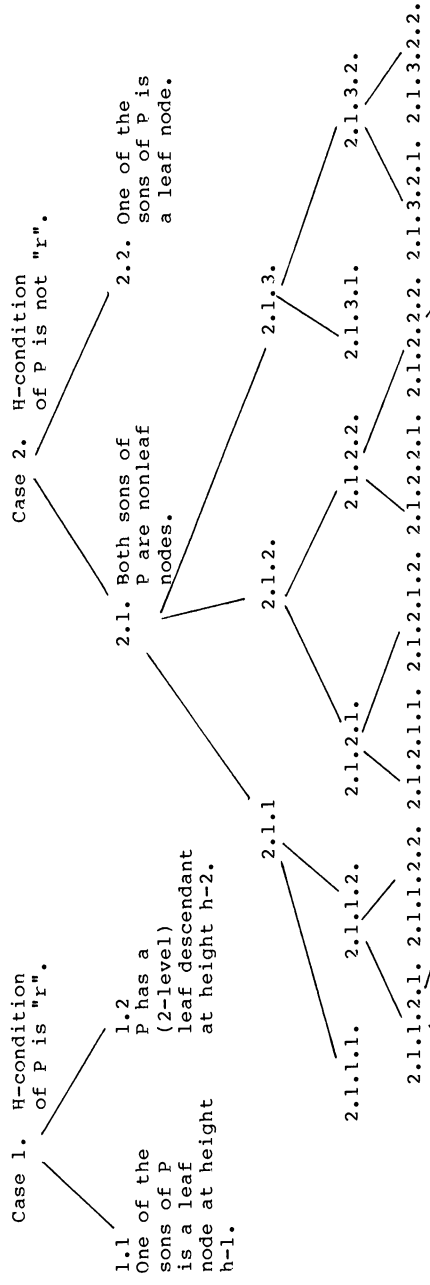

Case 1. H-condition of P is "r".
1.1 One of the sons of P is a leaf node at height h-1.
1.2 P has a (2-level) leaf descendant at height h-2.
Case 2. H-condition of P is not "r".
2.1. Both sons of P are nonleaf nodes.
2.2. One of the sons of P is a leaf node.
2.1.1
2.1.2.
2.1.3.
2.1.1.1.
2.1.1.2.
2.1.2.1.
2.1.2.2.
2.1.3.1.
2.1.3.2.
2.1.1.2.1.
2.1.1.2.2.
2.1.2.1.1.
2.1.2.1.2.
2.1.2.2.1.
2.1.2.2.2.
2.1.3.1.1.
2.1.3.1.2.
2.1.3.2.1.
2.1.3.2.2.
2.1.1.2.1.1.
2.1.1.2.1.2.
2.1.2.2.2.1.
2.1.2.2.2.2.

FIG. A.2

The transformation for *left-balancing* is symmetric and is left to the reader.

The following rules describe the transformation procedure for the different cases; their symmetric variants are omitted. Figure A.2 shows the cases considered.

(1) The H-condition of P is "r."

(1.1) One of the sons (say, right son Q) of P is a leaf node at height $h-1$ (Fig. A.3). Let D be the left neighboring leaf node of Q.

Transformation. Replace the left subtree at P by the subtree rooted at D.

(1.2) The right son of the left son of P or the left son of the right son of P is a leaf node at height $h-2$. Let the latter be the case (Fig. A.4). Let A and S be the left and the right neighboring leaf nodes of R, respectively.

Transformation. Replace the left subtree at P by the subtree rooted at A, and the right subtree at Q by the subtree rooted at S.

(2) The H-condition of P is not "r."

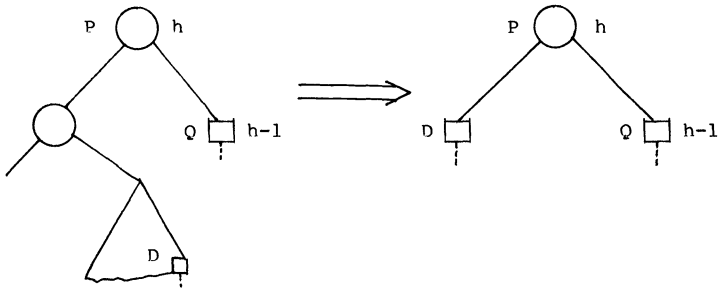


FIG. A.3

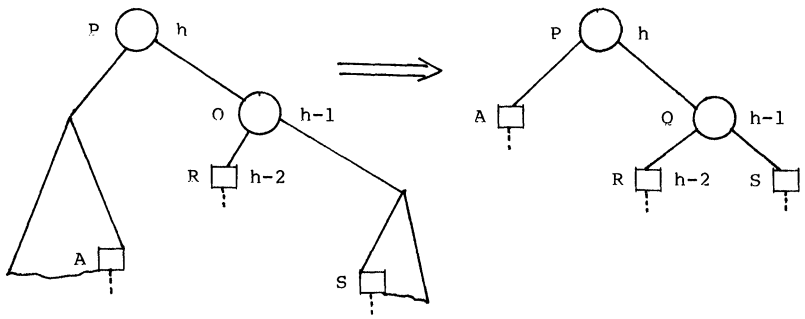


FIG. A.4

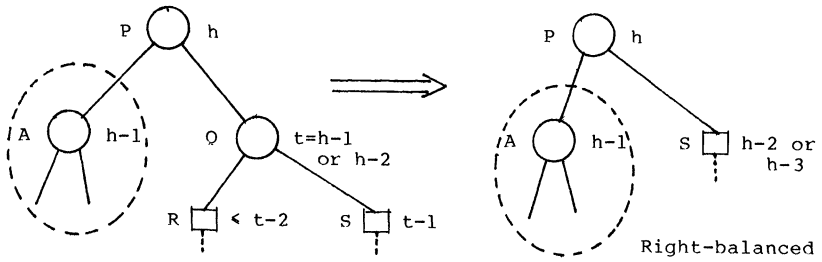


FIG. A.5

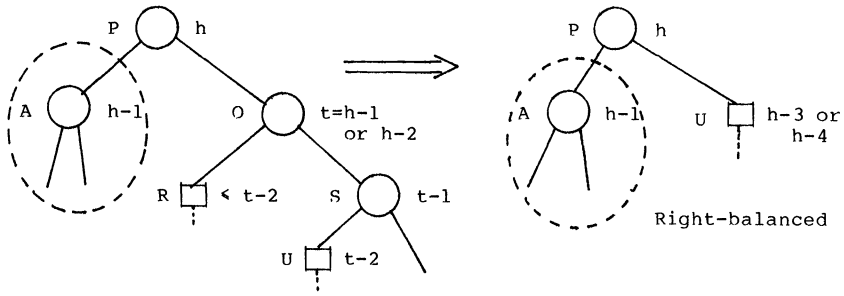


FIG. A.6

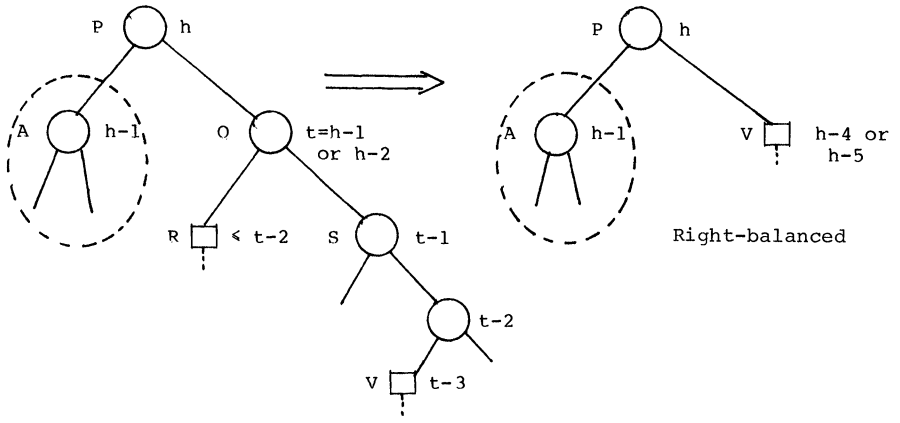


FIG. A.7

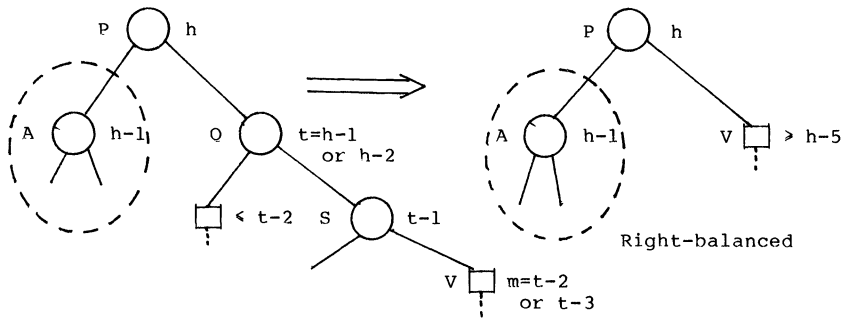


FIG. A.8

(2.1) Both sons of P are nonleaf nodes (Figs. A.5 through A.18). Let $h-1$ be the height of A, the left son of P. Let t be the height of Q, the right son of P. Note that t must be $h-1$ or $h-2$ because P satisfies Property A and the H-condition of P is not "r."

(2.1.1) The left son of Q is a leaf node at height $t-2$ or less (Figs. A.5 -A.9). Height of the right son of Q must obviously be $t-1$.

Transformation. As a partial transformation, right-balance the subtree rooted at A.

(2.1.1.1) S, the right son of Q, is a leaf node (Fig. A.5).

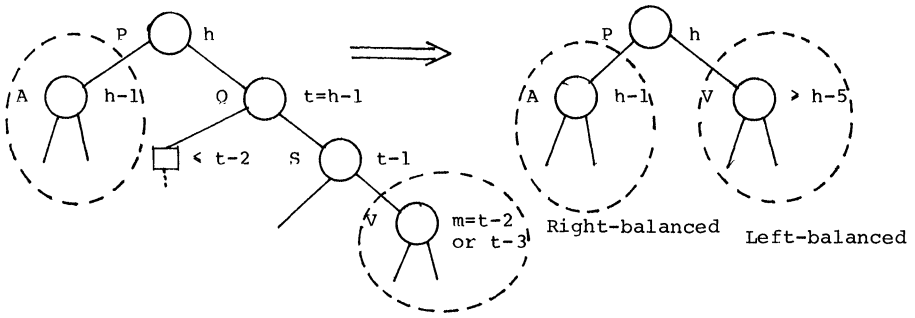


FIG. A.9

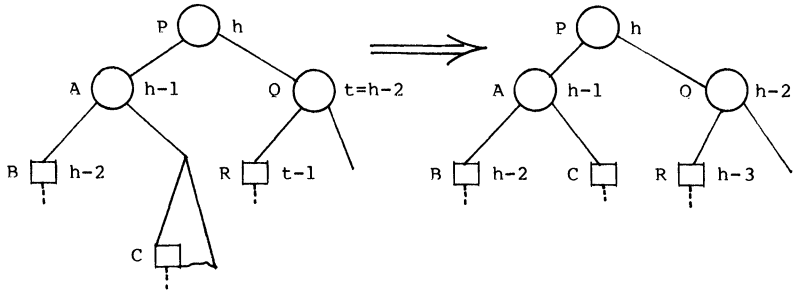


FIG. A.10

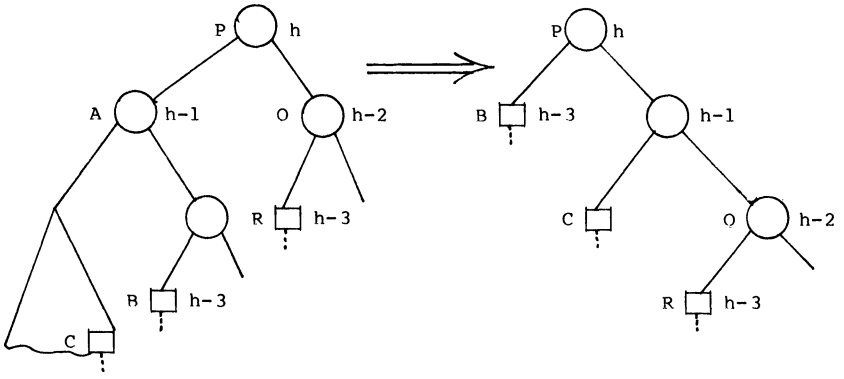


FIG. A.11

Transformation. Replace the right subtree at P by the subtree rooted at S.

(2.1.1.2) S is not a leaf node (Figs. A.6-A.9).

(2.1.1.2.1) H-condition of S is "r" (Figs. A.6 and A.7).

(2.1.1.2.1.1) U, a son of S (say the left), is a leaf node at height $t-2$ (Fig. A.6).

Transformation. Replace the subtree rooted at Q by that rooted at U.

(2.1.1.2.1.2) The right son of the left son of S or the left son of the right son of S is a leaf node at height $t-3$. Let the latter be the case (Fig. A.7).

Transformation. Replace the subtree rooted at Q by that rooted at V.

(2.1.1.2.2) The H-condition of S is not "r." V, the right son of S, is at height m (Figs. A.8 and A.9). m must be $t-2$ or $t-3$ because the H-condition of S is not "r" and P satisfies Property A.

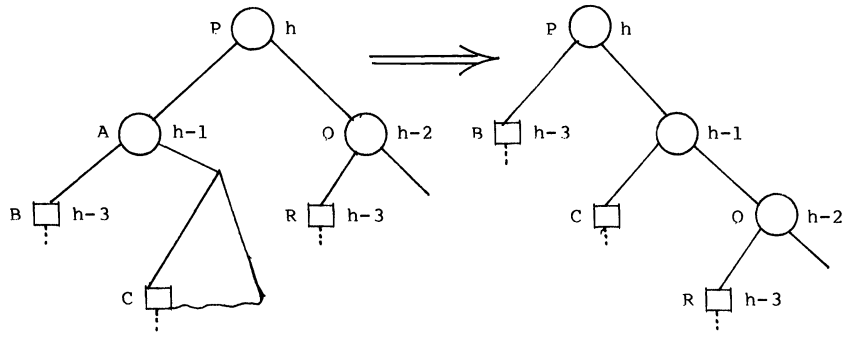


FIG. A.12

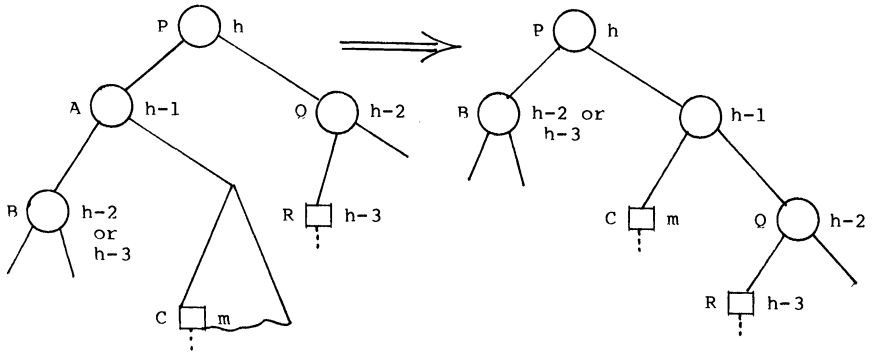


FIG. A.13

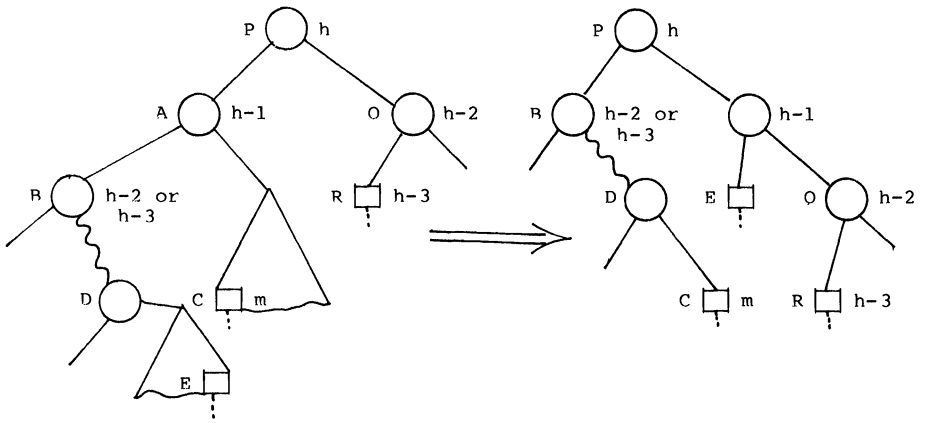


FIG. A.14

Transformation. If V is a leaf node, then replace the subtree rooted at Q by that rooted at V (Fig. A.8). If V is not a leaf node, then left-balance the subtree rooted at V, and replace the subtree rooted at Q by the resulting subtree (Fig. A.9).

(2.1.2) R, the left son of Q, is a leaf node at height $t-1$ (Figs. A.10 through A.14). t must be $h-2$ because the H-condition of P is not "r."

(2.1.2.1) The H-condition of A is "r" (Figs. A.10 and A.11).

(2.1.2.1.1) B, a son of A, is a leaf node at height $h-2$ (Fig. A.10). B must be the left son because the H-condition of P is not "r."

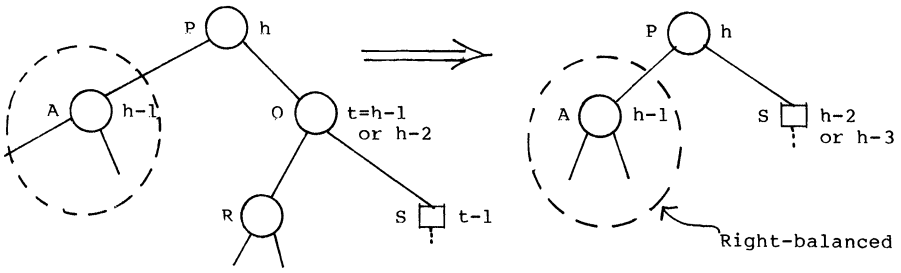


FIG. A.15

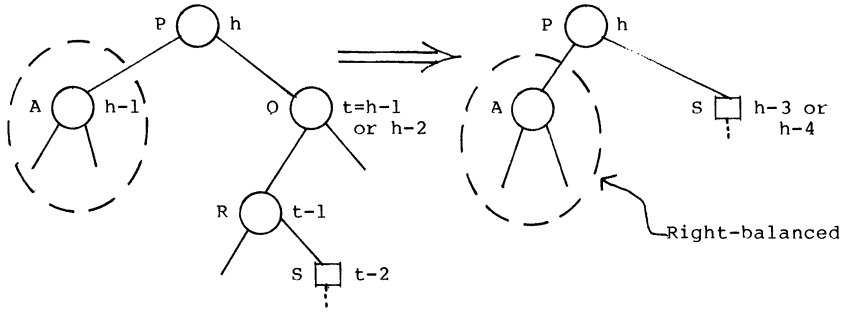


FIG. A.16

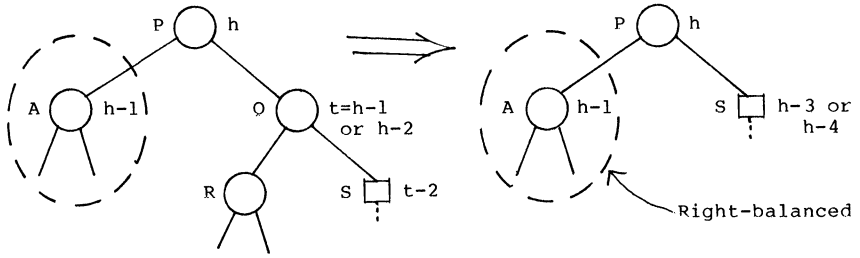


FIG. A.17

Transformation. Replace the right subtree of A by the subtree rooted at the right neighboring leaf node of B.

(2.1.2.1.2) B, the right son of the left son of A or the left son of the right son of A, is a leaf node at height $h-3$. Let the latter be the case (Fig. A.11). Let C be the left neighboring leaf node of B.

Transformation. Replace the left subtree at P by that rooted at B. Construct a tree whose left subtree is the subtree rooted at C and the right subtree is the subtree rooted at Q, and let this be the right subtree at P.

(2.1.2.2) The H-condition of A is not "r" (Figs. A.12-A.14). Let B be the left son of A. Its height must be $h-2$ or $h-3$ because the H-condition of A is not "r" and P satisfies Property A. Let C be the right neighboring leaf node of B.

(2.1.2.2.1) B is a leaf node (Fig. A.12). Its height must be $h-3$ because the H-condition of A is not "r."

Transformation. This transformation is the same as that for Case 2.1.2.1.2.

(2.1.2.2.2) B is a nonleaf node (Figs. A.13 and A.14). Let the height of C be m. $m \leqq h-4$ because the H-condition of A is not "r."

(2.1.2.2.2.1) The rightmost (2-level) leaf descendant of B is at height m or more (Fig. A.13). Thus, there is no major gap on the right downward path from B which is indirectly supported.

Transformation. This transformation is the same as that for Case 2.1.2.1.2.

(2.1.2.2.2.2) E, the rightmost (2-level) leaf descendant of B is at height less than m (Fig. A.14). Let D be a nonleaf node on the right downward path from B whose height is greater than or equal to, but closest to, $m+1$. Note that there cannot be an indirectly supported major gap on the path between B and D. Also, the right subtree at D cannot have a 2-level leaf descendant at height m or more.

Transformation. Right-balance the subtree rooted at B, and let it replace the left subtree at P. Construct a tree whose left subtree is the subtree rooted at E and the right subtree is the subtree rooted at Q. Let this be the right subtree at P.

(2.1.3) The left son of Q is a nonleaf node (Figs. A.15-A.18).

(2.1.3.1) The H-condition of Q is "r." The only possibilities are: (a) S, the right son of Q, is a leaf node at height $t-1$ (Fig. A.15); (b) S, the right son of the left son or the left son of the right son of Q, is a leaf node at height $t-2$. Let the former be the case (Fig. A.16).

Transformation. Right-balance the subtree rooted at A. Replace the subtree rooted at Q by that rooted at S.

(2.1.3.2) The H-condition of Q is not "r" (Figs. A.17 and A.18).

(2.1.3.2.1) S, the right son of Q, is a leaf node (Fig. A.17). The height of S must be $t-2$ because P satisfies Property A and the H-condition of Q is not "r."

Transformation. Right-balance the subtree rooted at A. Replace the subtree rooted at Q by that rooted at S.

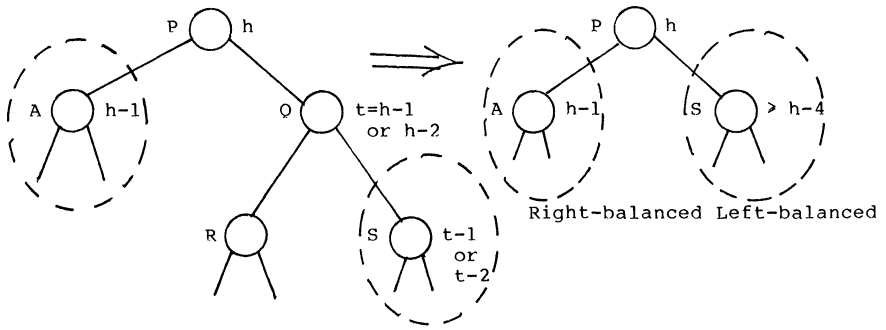


FIG. A.18

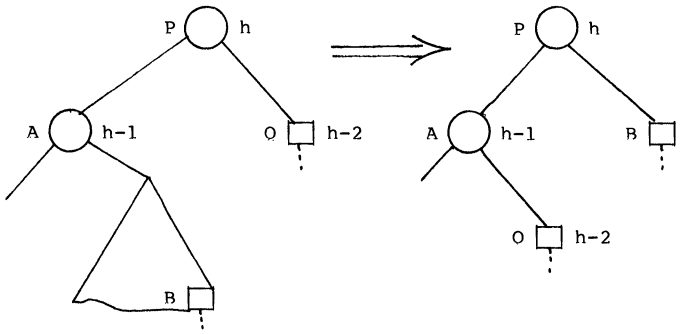


FIG. A.19

(2.1.3.2.2) S is not a leaf node (Fig. A.18). Its height must be $t-1$ or $t-2$.

Transformation. Right-balance the subtree rooted at A. Left-balance the subtree rooted at S, and let it replace the subtree rooted at Q.

(2.2) One of the sons of P, say right, is a leaf node (Fig. A.19). Its height must be $h-2$, because P satisfies Property A and its H-condition is not "r." Let B be the rightmost (2-level) leaf descendant of A.

Transformation. Replace the right-subtree at A by the subtree rooted at Q and the right subtree at P by that rooted at B.

Appendix B.

LEMMA (Lemma 2.5). $N(h, 2) \geqq N(h-3, 1)+1$, $h \geqq 2$.

Proof. The proof follows from induction on h.

Basis. $h = 2$ or 3.

$$N(2, 2) = 1 \geqq N(-1, 1)+1 = 0+1 = 1,$$

$$N(3, 2) = 2 \geqq N(0, 1)+1 = 0+1 = 1.$$

Induction hypothesis. The result is true for $2 \leqq h \leqq s$.

Induction step. To prove that the result is true for $h = s+1$, $s \geqq 3$.

In the following, note that $N(s, 1) \geqq 1 + N(s-1, 1)$, because the contribution of the lower height subtree at the root has to be at least 1.

For $s \geqq 6$:

$$N(s+1, 2) = \min\{N(s, 1), 1+N(s-1, 1), 2+N(s-2, 1), N(s, 2)+N(s-4, 2)\}$$

(see Fig. B.1; observe that the tree shown in (b) or (c) has "r" as the H-condition of its root).

$$N(s, 2)+N(s-4, 2) \geqq 2+N(s-3, 1)+N(s-7, 1) \text{ (by the induction hypothesis)}$$

$$= 2+N(s-2, 1) \quad \text{for } s \geqq 8$$

$$= 1+N(s-2, 1) \quad \text{for } s = 7 \text{ or } 6$$

because in such a case $N(s-7, 1) = 0$ and $N(s-2, 1) = 1+N(s-3, 1)$.

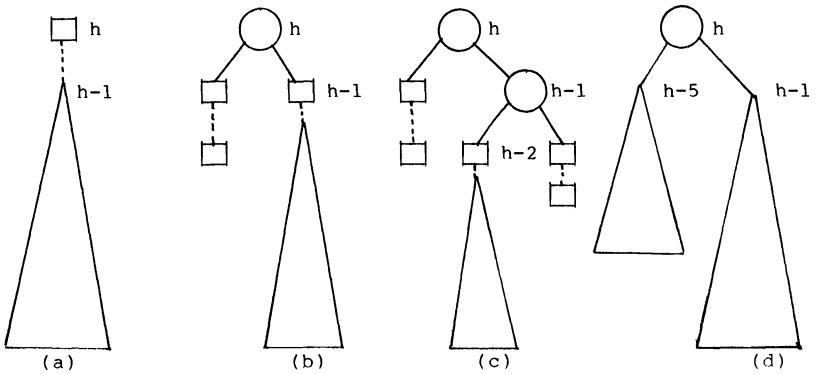


FIG. B.1

For $3 \leqq s \leqq 5$:

$$N(s+1,2) = \min\{N(s,1), 1+N(s-1,1), 2+N(s-2,1), 1+N(s,2)\},$$

$$1+N(s,2) \geqq 2+N(s-3,1) \quad \text{(by the induction hypothesis)}$$

$$= 1+N(s-2,1),$$

$$N(s,1) \geqq 1+N(s-1,1),$$

$$N(s-1,1) \geqq N(s-2,1).$$

Thus, $N(s+1,2) \geqq 1+N(s-2,1)$; hence the result.

Acknowledgment. The author is indebted to the anonymous referee for thoughtful and useful comments, to Roger Lamprey for going through an earlier version of the paper and making useful suggestions, and to Linda Davant for her excellent typing.

REFERENCES

[1] A. ABRAMASON, *Information Theory and Coding*, McGraw-Hill, New York, 1963.

[2] G. M. ADEL'SON-VELSKIJ AND Y. M. LANDIS, *An algorithm for the organization of information*, Soviet Math. Dokl., 3 (1962), pp. 1259-1263.

[3] S. W. BENT, D. D. SLEATOR AND R. E. TARJAN, *Biased search trees*, this Journal, 14 (1985), pp. 545-568.

[4] J. FEIGENBAUM AND R. E. TARJAN, *Two new kinds of biased search trees*, Bell Systems Tech. J., 62 (1983), pp. 3139-3158.

[5] R. H. GUETING AND H. P. KRIEGEL, *Dynamic K-dimensional multiway search under time-varying access frequencies*, Proc. 5th GI Conference on Theoretical Computer Science, Lecture Notes in Computer Science 104, Springer, Berlin, New York, 1981, pp. 135-145.

[6] D. E. KNUTH, *The Art of Computer Programming, Vol.* III: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[7] F. LUCCIO AND L. PAGLI, *On the height of height-balanced trees*, IEEE Trans. Comput., 25 (1976), pp. 87-90.

[8] K. MEHLHORN, *Arbitrary weight changes in dynamic trees*, Angew. Math. Inform. Bericht 78/04, Universitat des Saarlandes, 1978.

[9] ———, *Dynamic binary search*, this Journal, 8 (1979), pp. 175-195.

[10] V. K. VAISHNAVI, *Multidimensional height-balanced trees*, IEEE Trans. Comput., 33 (1984), pp. 334-343.

[11] ———, *On the worst-case efficient implementation of weighted dynamic dictionaries*, Proc. 21st Annual Allerton Conference on Communication, Control and Computing, 1983, pp. 647-655.

[12] ———, *On the height of multidimensional height-balanced trees*, IEEE Trans. Comput., 35 (1986), pp. 773-780.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987
© 1987 Society for Industrial and Applied Mathematics

THE COMPLEXITY OF RELIABLE CONCURRENCY CONTROL*

CHRISTOS H. PAPADIMITRIOU† AND MIHALIS YANNAKAKIS‡

Abstract. We define what it means for a schedule to be reliable, that is, correct in the face of possible transaction failures (assuming that aborting a transaction to restore correctness is not allowed). It turns out that the right definition is recursive, and surprisingly involved. We show that, in fact, testing a schedule for reliability is PSPACE-complete, and thus in some sense even harder than the ordinary NP-complete notions of correctness examined in the past. However, we also prove that all conflict serializable schedules are always reliable, and thus reliability should not be an extra complication for practical concurrency control systems. Finally, we examine two other notions of reliability, related to multiple versions and aborts.

Key words. concurrency control, schedule, reliable schedule, serializability, PSPACE-complete problems, games

1. Reliable schedules.

In database concurrency control we are concerned with the correctness of *schedules*, that is, interleavings of read and write steps originating from several transactions. A step either reads or writes a *database entity*. The point is to design algorithms which monitor the concurrent execution of transactions, so that only *correct* schedules are allowed to execute.

Defining the right notion of correctness is quite nontrivial. It is now widely accepted in the field of database concurrency control that a schedule must be considered correct if it is *view-serializable*, that is, all transactions read the same views that they would in a serial execution, under all interpretations of the transactions (see, e.g., [10], [12] for basic terminology and facts for concurrency control). In practice, however, all concurrency control algorithms use a more relaxed correctness criterion called *conflict-serializability*, which requires that a certain *conflict graph* have no cycles; in fact, conflict-serializability was historically the first notion of correctness to be proposed in the literature [3]. There are at least two reasons for the popularity of conflict-serializability. First, view-serializability is an NP-complete property [9], and thus too involved to be the basis of practically useful algorithms; in contrast, conflict-serializability can be checked in polynomial time. Besides, it was shown in [12] that only subsets of the conflict-serializable schedules can be implemented by locking algorithms, by far the most popular kind.

Still, until recently there has been no major challenge to view-serializability as the standard sufficient condition for correctness in database concurrency control. Recent reseach in the *reliability* aspects of concurrency control [4], [6], [12], however, has shown that certain view-serializable schedules are, in a very real sense, incorrect. Consider, for example, the following schedule s with transactions A, B, C, and D:

$$s = \begin{array}{ll} A: & W(x) \\ B: & \quad R(x) \quad W(y) \\ C: & \quad\quad R(y) \quad W(x) \\ D: & \quad\quad\quad\quad W(x) \end{array}$$

* Received by the editors November 11, 1985; accepted for publication (in revised form) August 27, 1986.

† Department of Computer Science, Stanford University, Stanford, California 94305, and the National Technical University of Athens, Athens, Greece. The work of this author was supported by the National Science Foundation and the Hellenic Ministry for Research and Technology.

‡ AT&T Bell Laboratories, Murray Hill, New Jersey 07974. The research of this author was conducted while he was visiting the National Technical University of Athens, Athens, Greece, on leave from AT&T Bell Laboratories.

It is clearly view-serializable, because it is equivalent to *CABD*. There is, however, a serious problem with *s*. If, for some reason, transaction *D* is aborted, then *s* ceases being view-serializable. In some sense, the correctness of *s* depends critically on the completion of *D*. We can say that *s* is not *reliable*.

Let us try to arrive at the right definition of reliability. We assume that at all times any *active* transaction may fail. (A transaction is considered active at the closed interval defined by its first and last steps; single step transactions, such as *D* in our example above, are active only at the moment of the execution of their step.) When a transaction fails, its steps are removed from the schedule; in addition, we have the freedom of rearranging the steps after the moment of failure.

There are certain immediate necessary conditions for reliability. For example, if a transaction reads an entity from an active transaction, as in

$$\begin{array}{ll} A: & R(x)\,W(x) \qquad W(y) \\ B: & \qquad\quad R(x) \qquad W(z) \end{array}$$

then the schedule is definitely unreliable: If *A* aborts after the $R(x)$ step of *B*, then we cannot continue *B* either, since it has read a version of *x* which is no longer valid. (Of course, we could still salvage the correctness of the schedule by *rolling back B*, and redoing the read step, but this requires a drastic extension of our model; see [6] and the last section.) Also, if the initial value of *x* is not kept around after the $W(x)$ step of *A*, then the system cannot recover from a failure of *A*. Therefore, we shall make in this paper the following assumptions about schedules in the absence of which reliability is impossible [6]:

(a) The old version of an entity is kept after it is overwritten by a transaction, until this transaction terminates (we say that the new version is then *committed*).

(b) No read step reads an uncommitted version.

Note. Assumptions (a) and (b) above are automatically satisfied in the so-called *two-step model* of transactions [9], in which a transaction consists of an atomic read step of several entities, followed by an atomic write step of another set of entities. In fact, it seems clear in retrospect that this model was proposed because of implicit reliability considerations. In the negative results in this paper we restrict ourselves to the two-step model, although our positive results are applicable in general.

A schedule obeying assumptions (a) and (b) is to be called reliable if, intuitively, its correctness is immune to transaction failures. Thus, a first cut at the definition sought may be the following: Call a schedule *s reliable* if it is view-serializable and, for each step *a* of *s*, the following is true: If we abort a transaction active at *a*, then there is a view-serializable schedule of the remaining transactions, which agrees with the order of the steps in *s* up to *a*.

This definition has a serious problem. Consider the following schedule:

$$\begin{array}{lll} & A: W(x) & \\ & B: \quad R(x) & \qquad\qquad W(y) \\ s = & C: \qquad W(x)\,W(y) & \\ & D: \qquad\qquad W(x) & \\ & E: \qquad\qquad\qquad W(x) & \end{array}$$

This schedule is equivalent to *CDABE*, and is therefore view-serializable. It is also "reliable" in the sense of our candidate definition. If *D* is aborted (this is the only dangerous moment), then we can definitely extend the rest of *s* to the following

view-serializable continuation of s:

$$s' = \begin{array}{ll} A: & W(x) \\ B: & \quad R(x) \qquad\qquad\quad W(y) \\ C: & \quad\quad W(x)\,W(y) \\ E: & \quad\quad\quad\quad W(x) \end{array}.$$

Another possible continuation would be s'', which is s' with the last two steps interchanged. In fact, s' and s'' are the only such continuations. The problem is that s' *itself is not reliable* (neither is s''). *If now* we abort E, then no view-serializable continuation exists. We are therefore led to adopt the following recursive definition of reliability.

DEFINITION. Let s be a schedule, and let q be a suffix of s ($s = pq$). We say that s is *q-reliable* if, for every step a of q ($q = q_1 a q_2$) and each transaction A active at a, there is a schedule s' involving the transactions of s except A such that

(1) $s' = rq_2'$ where r contains the same steps as $pq_1 a$ in the same order (except of course for the steps of A which are necessarily missing from s');
(2) q_2' is a rearrangement of the steps in q_2 (again, except for the steps of A);
(3) s' is q_2'-reliable.

For the basis of the recursion, a schedule is λ-reliable, where λ is the empty suffix, if it is view-serializable. If a schedule s is s-reliable, then it is called just *reliable.*

This is our definition of reliable schedules. We can immediately show a helpful lemma stating that, when constructing s' according to the definition, we need not worry about transactions in s that have not yet started at point a.

LEMMA 1. *The definition above is not affected if we replace "involving the transactions of s except A" by "involving the transactions of s that have started at a, except A."*

Proof. Assuming that a subsequent transaction is necessary for the construction of s', we abort this transaction upon its appearance in s', and consider the resulting continuation which, by definition, exists. This continuation could have played the role of s' in the first place. □

Another helpful fact is that the definition is not affected if we allow simultaneous aborts of many transactions. (Proof: simultaneous aborts can be simulated by many individual aborts.)

2. Reliability and conflicts. How complex a concept is reliability? Our first result says that, in practice, it is not complex at all. As mentioned above, all known concurrency control algorithms implement a simpler notion of correctness called conflict-serializability. It is defined in terms of the *conflict graph* of the schedule s, $G(s)$, which has a node for each transaction, and an arc from A to B if a step of A conflicts with a step of B which comes later in s. (Two steps conflict if one writes an entity, and the other reads or writes it.) A schedule s is conflict-serializable if its conflict graph is acyclic. This correctness criterion, first proposed in [3], is known to imply view-serializability.

THEOREM 1. *If a schedule*[1] *is conflict-serializable, then it is reliable.*

Proof. The main theorem of [12] states that the class of conflict-serializable schedules is closed under projection, that is, omission of all the steps of some transaction from it. It follows by induction on the number of transactions that a conflict-serializable schedule has a reliable continuation, namely the same schedule with the aborted transaction omitted. □

[1] Recall that all schedules considered in this paper obey conditions (a) and (b) above.

In other words, no practical algorithm need worry about reliability, as long as conditions (a) and (b) are maintained. This is another ex post facto justification of the popularity of conflict-serializability as a correctness criterion in concurrency control: It simplifies the reliability problem considerably (compare Theorem 1 with Theorem 2 below).

3. Two conjectures. Is the converse of Theorem 1 true? If so, reliability would be remarkably simple, for all its complicated definition. The answer is "no." There are reliable schedules that are not conflict-serializable, for instance

$$\begin{array}{rlll} & A_1\colon W(x) & & \\ s = & A_2\colon \quad R(x) & \quad W(x). & \\ & A_3\colon \quad\quad W(x) & & \end{array}$$

Therefore, we cannot prove so easily that reliability is polynomial. Is it then NP-complete? Upon consideration, it may not even be in NP. The recursive definition is suspiciously reminiscent of a two-person game: A GOOD player tries to find continuations, from which an EVIL player aborts transactions, and so on. The natural complexity class for such problems is PSPACE [2].

If we could prove an interesting (and very intuitive) conjecture, we would at least show reliability to be in NP. The conjecture is motivated as follows: After the first transaction is aborted, GOOD has considerable freedom in choosing a continuation. Perhaps she is always able to choose a "super-reliable" continuation s'. Such a continuation would have the property that any subsequent abort by EVIL can be met by simply erasing the steps of the aborted transaction. No further rearrangement of steps is necessary.

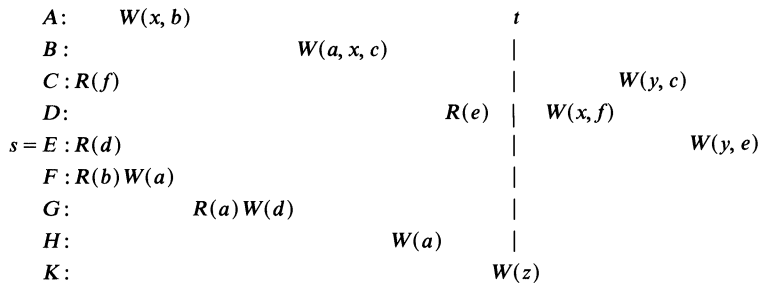
Unfortunately, super-reliable continuations do not always exist. The smallest counterexample that we know employs *nine* transactions (see Fig. 1(a)), plus the introduction of the notion of the *polygraph of a schedule at time t.* In general, a *polygraph* $P = (V, A, C)$ is a set V of nodes, together with a set A of directed arcs (pairs of nodes), and a set C of *choices* (ordered triples of nodes).

Let s be a schedule. We identify "time" with the steps of s (i.e., the first step takes place at time 1, etc.). The polygraph of s at time t, $P_t(s)$, has as its nodes the transactions that have started (and possibly finished) by time t (t included). Let s_t be the prefix of s up to and including time t. By *completion* of s_t we mean any schedule which agrees with s up to time t, and then executes in any order the remaining steps of the transactions that are active at t. An arc from A to B in the polygraph $P_t(s)$ indicates that A must come before B in any serial schedule equivalent to any completion of s_t. There are three types of arcs $A \to B$:

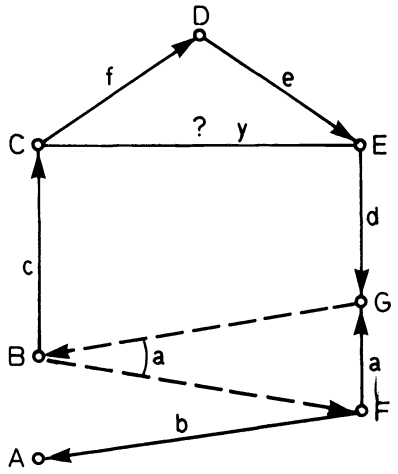
(1) B reads an entity x from A (example: arc $F \to G$ in Fig. 1).
(2) B writes the final value of entity x, and A writes or reads a previous version of x (example: arc $C \to D$ because of entity f).
(3) A reads the initial version of an entity which B subsequently writes (example: $F \to A$, $D \to E$).

Note that one of the steps that gives rise to an arc may not have occurred at time t, but rather it may be forced in any continuation. For example, the arc $C \to D$ is present because no other transaction besides D will write entity f after time t. On the other hand, there is no arc between C and E, because at time t we do not know which will write y last.

The choices of $P_t(s)$ are triples, such as (G, B, F), such that, in any equivalent schedule of any continuation of s after t, G must come after F, and B either before



(a)



(b)

FIG. 1. *Schedule s and* $P_t(s)$.

F or after G (since G reads a from F, and B writes a). In general, a choice (A, B, C) due to x is added whenever A reads x from C, and B writes a version of x, other than the final. Choices are shown as pairs of dotted arcs, as in Fig. 1(b).

The example of Fig. 1 is worth explaining, because it lies at the heart of the difficulty of reliability (for clarity, we have omitted from Fig. 1(b) some inconsequential parts of the polygraph, namely nodes H and K, and arcs $A \to D$ and $B \to D$ due to x). From the polygraph associated with this prefix of schedule, we can conclude the following:

(1) If D is aborted, then the arc (A, B) is added to the graph (D no longer overwrites x). This forces the choice (G, B) (that is, G must come before B), and thus the write step of C must come *after* that of E.

(2) If D is not aborted, it forces the choice (B, F), and the write step of C must come *before* that of E.

It follows that, if K aborts, *there is no super-reliable continuation*, despite the fact that the schedule is reliable. The right order of the write steps of C and E depends on whether D aborts in the future or not; there is no continuation good for both eventualities. Let us note for future reference that, until transaction D starts, the schedule is proceeding in a conflict-serializable way, in the sense that it has a conflict-serializable (and thus, by Theorem 1, super-reliable) completion. Interestingly, our

main negative result concerning reliability will depend on a variant of this counterexample.

Another tempting conjecture, which would also place reliability in the polynomial hierarchy, is the following: A schedule s is reliable if and only if the following property holds for each step a of s: If we abort any *set* of active transactions at a, then we can find a view-serializable completion of the remaining steps of the other active transactions. The difference with our first failed attempt at the definition of reliability is that many transactions are allowed to abort simultaneously. The *only if* part follows from the discussion after Lemma 1. Thus, we could discover that the schedule in the example of § 1 is not reliable, by noticing that if we abort both transactions D and E, then there is no reliable continuation. In fact, it may seem that an argument along the following lines would also prove the *if* part. By Lemma 1, we have only to worry about completing active transactions. Suppose that a schedule s is not reliable. This means that we can find at some point an active transaction to abort, such that in any continuation, we can find another transaction A_2 to abort, such that in any continuation we can find A_3 to abort, $\cdots$, such that any continuation is nonserializable. But then if we abort $A_1, A_2, A_3, \cdots$, there is no serializable completion.

There is a hole in this argument, and this hole is another root of the difficulty of reliability. In some sense, this second "weak reliability" conjecture is dual to our first "super-reliability" conjecture. In essence, the first conjecture said that if GOOD can win (the schedule is reliable) then after an abort she has a winning sequence of moves she can use no matter how EVIL plays in the future (a super-reliable continuation). The second conjecture says that, if EVIL can win (the schedule is not reliable) then he can choose a winning sequence of moves (set of transactions to abort) which he can use no matter how GOOD plays.

We can use the properties of the example of Fig. 1 to almost derive a counterexample to this second conjecture (see below why "almost"). In the schedule in Fig. 1, there is no conflict between the write steps of transaction D and the write steps of C and E. Thus, for serializability purposes, the write step of D can be switched with the write steps of C and E. However, for reliability purposes, it *must* come first. The reason is that we cannot decide an order between the write steps of C and E until we know whether D aborts. If the write step of C comes first, then EVIL knows that $C \rightarrow E$ and can abort D; if the write step of E comes first, then EVIL knows that $E \rightarrow C$ and can leave D to complete. Now suppose that we have two copies of the schedule of Fig. 1 running in parallel, and let us subscript transactions by the copy number. The two copies have disjoint sets of entities and transactions, except that $C_2 = D_1$, and $E_2 = C_1$. If D_2 aborts, then the write step of C_2 must come after the write step of E_2, i.e., the write step of D_1 after that of C_1. This destroys reliability in the first copy, since EVIL can now abort D_1 after the write step of C_1 takes place. However, if both $D_1 (= C_2)$ and D_2 abort simultaneously, then the second copy has no effect, and there is no problem. This construction would yield a counterexample to the second conjecture if only the graph of Fig. 1 did not have the arcs $C \rightarrow D$ and $D \rightarrow E$.

We are going to develop a schedule which has the properties of the one in Fig. 1, only without these two forced precedences. Besides providing a counterexample for the weak reliability conjecture, this schedule will be used as a crucial "gadget" in our PSPACE-completeness proof in the next section. In Fig. 2 we show a schedule s_1 of 15 transactions. The distinguished transactions are D, C, and E; their write steps come after the broken line and are not shown in Fig. 2. The transactions C and E will write a common entity, not accessed by any other transaction. Steps which are in the same column in Fig. 2 are assumed to take place in any order, as they do not conflict. In

C:		$R(l)$						
E:		$R(k)$						
D:							$R(f, g)$	
1:	$W(a, c, k)$							
2:			$W(a, l)$					
3:			$R(c, d)$					
4:				$R(a, b)$	$W(c, d)$			
$s_1 =$ 5:		$W(b, d, h)$						
6:			$W(b, e)$					
7:	$R(h)$							$W(g)$
8:					$W(e, f)$			
9:						$W(a)$		
10:						$W(b)$		
11:						$W(c)$		
12:						$W(d)$		

FIG. 2. *Schedule* s_1.

Fig. 3 we show the polygraph of the schedule at the broken line. We have labeled the arcs and the choices by the entities that caused them. We have omitted from Fig. 3 transactions 9 to 12. These transactions serve to overwrite the entities a, b, c and d which cause the choices; they are sinks (have no outgoing arcs), and therefore do not help create any paths or cycles on the graph. As the end of the schedule of Fig. 2 (the broken line) the only active transactions are C, D and E.

The schedule s_1 of Fig. 2 is view-serializable (assuming that C writes before E). In Fig. 4(a) we show an acyclic orientation of the polygraph in Fig. 3. Actually, this is the only acyclic orientation: The path $6 \to 8 \to D \to 7 \to 5$ forces us to delete arc $5 \to 6$

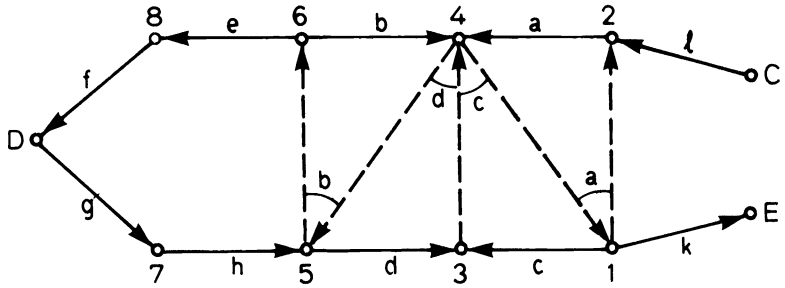


FIG. 3. $P_t(s_1)$.

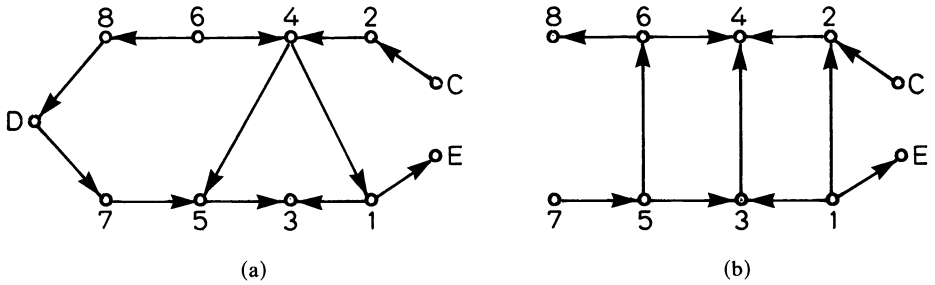


FIG. 4. *Two acyclic orientations of* $P_t(s_1)$.

and keep $4 \rightarrow 5$ from choice b; then we have to remove $3 \rightarrow 4$ from choices d and c, keep $4 \rightarrow 1$ and remove $1 \rightarrow 2$. Note that a path from C to E is forced. Thus, in any serial equivalent schedule, C must come before E. However, there is no directed path between D and either C or E.

If transaction D is removed, then the remaining polygraph has another acyclic orientation, shown in Fig. 4(b). In this orientation there is no directed path between C and E, and thus either order is possible.

The schedule s_1 of Fig. 2 is furthermore reliable. Until the point that D starts, conflict-serializability is observed (the acyclic conflict graph is the one shown in Fig. 4(b)). After D starts, we can choose the orientation in Fig. 4(a) or (b), depending on whether D aborts or not; there is no need for any rearrangement.

Our second gadget is a schedule s_2 composed of two copies s_1, s_1' of the schedule of Fig. 2. In s_1' we add primes to all entities and transactions, which are distinct from those of s_1 except that $C' = E$ and $E' = C$. The two copies run in parallel (note that we can realize s_2 using the two-step model of transactions, in which all read steps come together, followed by all write steps also coming together, because the order of the read steps of E and C is irrelevant). At the end of s_2, the only active transactions are D, D',C and E. We assume that none of the entities that have been accessed so far are written by any of these transactions, with the important exception that entity e' is in the write set of D. As a consequence, the polygraph does not have an arc $6' \rightarrow 8'$ labeled e' (the final value of e' is written by D', not $8'$), but rather has arcs $6' \rightarrow D$, $8' \rightarrow D$ labeled e'. The polygraph at the end of s_2 is shown in Fig. 5 (again omitting the transactions 9 to 12 and 9′ to 12′).

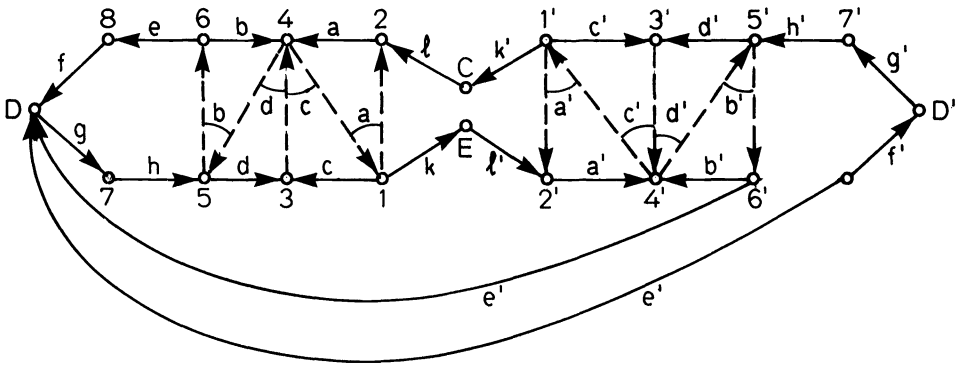


FIG. 5. $P_\infty(s_2)$.

Schedule s_2 is view-serializable. As we explained before, we have to choose for the unprimed (left-hand) copy the orientation of Fig. 4(a) for the choices. This causes the path $C \rightarrow E$ (otherwise we have a cycle in Fig. 4(a)), and thus forces the orientation of Fig. 4(b) for the primed (right-hand) copy. Note that this is possible because the right-hand copy does not have arc $6' \rightarrow 8'$. Thus, the overall polygraph has only one acyclic orientation as shown in Fig. 6(a). In this forced orientation there is a path from C to E. The only other path between the distinguished nodes D, D', C and E is from D' to D.

If transaction D aborts, the polygraph acquires an arc $6' \rightarrow 8'$, because $8'$ writes the final value of e'. Thus, the right-hand copy of s_1 is "turned on," while the left copy is "turned off." The right copy must choose the orientation of Fig. 4(a), and the left that in 4(b). In Fig. 6(b) we show the overall orientation that if forced if D is aborted.

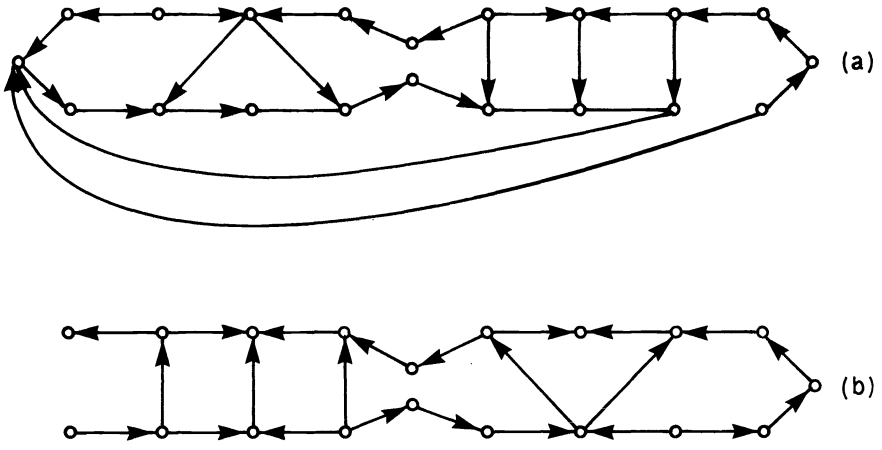


FIG. 6. *Transaction D is aborted.*

Notice that there is no path between D' and either E or C. If both D and D' abort, then both parts must choose the orientation in Fig. 4(b), and there is no path between C and E. Thus, s_2 is reliable.

To summarize, s_2 has the following properties:

(1) If D is not aborted, then $C \to E$;

(2) If D is aborted and D' is not, then $E \to C$;

(3) If both D and D' abort, then either order between C and E is possible.

Because of these properties, in order to ensure reliability, the write step of D must come before the write steps of C and E (assuming no transaction has been aborted). Recall that C and E write a common entity which no other transaction writes; thus the order of these write steps determines who writes the final value of the entity, and whether $C \to E$ or $E \to C$. If the write step of C comes before those of D and E, then EVIL can abort D after the write step of C (without aborting D') and win. If the write step of E comes before those of C and D, then EVIL does not have to do anything: the schedule is not view-serializable.

Schedule s_2 will be our workhorse in the PSPACE-completeness proof given in the next section. We call D and D' the *input terminals* of s_2; C and E are the *output terminals.*

4. The main result. We now show the following theorem.

THEOREM 2. *Telling whether a schedule is reliable is* PSPACE*-complete.*

Proof. First, it is quite clear that the problem can be solved in polynomial space: We can try all possible strategies for GOOD and EVIL, reusing space after we have checked each. To show completeness, we shall reduce to reliability the well-known PSPACE-complete problem *quantified satisfiability* (QSAT [5]). In this problem we are given a formula of the form

$$F = \forall x_1 \exists x_2 \forall x_3 \cdots \exists x_n [C_1 \wedge C_2 \wedge \cdots \wedge C_m],$$

where the C_j's are *clauses*, that is, disjunctions of literals. A literal is either one of the variables $x_1, \cdots, x_n$ or its negation. It is well known that we may only consider clauses with three literals. The problem is whether the formula F is true. QSAT can be viewed as a two-person game between a UNIVERSAL and EXISTENTIAL player.

UNIVERSAL chooses values for the odd variable trying to force a false clause, while EXISTENTIAL chooses in turn values for the even variables trying to satisfy all clauses.

From the formula F we shall construct a schedule $s(F)$ such that $s(F)$ is reliable iff F is true. For every variable x_i we have two transactions D_i and D_i'. A move of the EVIL player in the reliability game will correspond to a move of the UNIVERSAL player in the QSAT game as follows: Setting a universal variable x_i to **true** corresponds to abortion of D_i, while setting x_i to **false** corresponds to an abortion of D_i' (at a certain appropriate time). We shall have a device to ensure that EVIL aborts one of D_i, D_i' for all odd i (otherwise GOOD wins regardless of whether F is true or not).

A move by EXISTENTIAL in QSAT is simulated by GOOD in the reliability game as follows. For each even i (existential variable x_i), transactions D_i and D_i' write a common entity which no other transaction writes. Thus, the order of those write steps determines the final value of this entity. After EVIL aborts D_{i-1} or D_{i-1}', GOOD can rearrange the order in which D_i and D_i' write; D_i before D_i' corresponds to $x_i =$ **true**, and the opposite order corresponds to $x_i =$ **false**.

The schedule $s(F)$ has a large initial portion which "sets the stage" for the reliability game. During that portion, if EVIL makes a move (i.e., aborts a transaction) then GOOD has a guaranteed win. We shall first describe the schedule by its polygraph after the stage has been set. The schedule and its polygraph are composed of several parts:

(1) A part that forces EVIL to abort D_i or D_i' for each odd i.
(2) A part for each clause, such that a cycle will be formed if all literals are false.
(3) Fan-out gadgets for each variable x_i, which distribute the truth value from D_i, D_i' to the occurrences in the clauses.
(4) "Precedence" gadgets, which force the right order of play.

Let us now describe each part. For part (1), we have transactions $T_1, T_3, \cdots, T_{n-1}$ corresponding to the universal variables, and a transaction T_{n+1}. There is a *potential path* from T_1 to T_{n+1}, as in Fig. 7 (the reason it is "potential" is explained below). The arcs of the figure are write-write, that is, T_1 writes y_1 and y_1', then T_3 writes y_1, y_1', y_3, y_3', T_5 writes y_3, y_3', y_5, y_5', and so on. These arcs, however, are not present in the polygraph for the following reason: For each odd i, the final value of y_i is finally overwritten by D_i, and that of y_i' by D_i'. Thus, if for some odd i EVIL does not abort one of D_i, D_i', then the path from T_1 to T_{n+1} is broken; and EVIL needs this path in order to win (create a cycle).

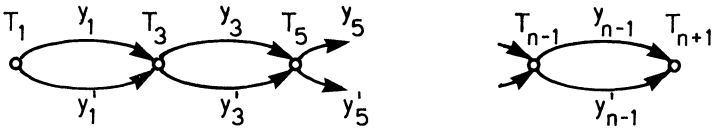


FIG. 7. *The potential path.*

In (2), for each clause we have a potential path back from T_{n+1} to T_1, as shown in Fig. 8. The path is composed of four arcs and three undirected edges; each undirected edge corresponds to a literal in the clause. For each arc and edge, the entity labeling it is unique to its endpoints, that is, no other transaction reads or writes it. Each arc is a read-write arc, that is, the tail of the arc reads the initial value of the entity, and then the head writes a new value of the entity. The undirected edges correspond to write-write conflicts that take place at the final part of the schedule, and are resolved in a way that reflects the truth values of the variables. Eventually, all edges will be

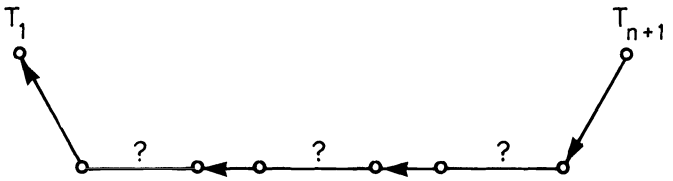


FIG. 8. *The clause path.*

directed one way or the other. If they all become directed in the direction $T_{n+1} \to T_1$, corresponding to all literals in the clause being false, then a path from T_{n+1} to T_1 will result. This path, together with a $T_1 \to T_{n+1}$ path from part (1), will form a cycle.

For part (3) we have different fan-out gadgets for universal and existential variables. For a universal variable x_i we use schedule s_2. That is, for every occurrence of x_i or $\bar{x}_i$ in a clause C_j, we have a distinct copy of s_2. Nodes D and D' of s_2 are identified with nodes D_i and D_i', and nodes C and E of s_2 are identified with the endpoints of the undirected edge that corresponds to the occurrence of x_i in C_j. For a positive occurrence, C is identified with the node that is closer to T_{n+1}, and for a negative occurrence it is identified with the endpoint that is closer to T_1. In either case, when EVIL chooses a truth value for x_i (aborts D_i or D_i') he forces GOOD to give a consistent direction to all edges if she is to keep the graph acyclic. If the literal is false, then the edge is directed towards T_1; if it is true, the edge is directed away from T_1, thus breaking this potential path from T_{n+1} to T_1.

For existential variables, we use a variant of s_1 as our fan-out gadget. In this variant, denote it $\bar{s}_1$, the read set of D is empty. The entities f and g are in the write set of D, the broken line is moved right after transactions 9 to 12 write, and the order in which D and 7 write their common entity is left unspecified (see Fig. 2). The polygraph of $\bar{s}_1$ at its broken line is almost identical to that of s_1 (Fig. 3). There is still an arc $8 \to D$ labeled f (now it is a write-write arc). The only difference is that we do not have arc $D \to 7$. Instead, there is an undirected edge, the orientation of which is to be determined by the order in which D and 7 write g. If $D \to 7$ then the gadget is "turned on" and we must have $C \to E$ (as in Fig. 4(a)). However, if $7 \to D$, then the gadget is "turned off"; we can choose the acyclic orientation in Fig. 4(b), which does not force an order between C and E. Notice that, in either case, there are no paths between D, 7 and C, E.

For each occurrence of an existential variable x_i (i even) or its negation in a clause, we have a distinct copy of $\bar{s}_1$ connecting nodes D_i and D_i' to the nodes of the edge that corresponds to the literal occurrence in the clause part. Node C of $\bar{s}_1$ is identified with that node of the edge which is closer to T_{n+1}, whereas node E is identified with the other one. For a positive occurrence of x_i node D of the copy of $\bar{s}_1$ is identified with D_i' and node 7 of $\bar{s}_1$ with D_i; for a negative occurrence, D is identified with D_i and 7 with D_i'.

Suppose that GOOD sets x_i to **true**, that is, orders the write step of D_i before D_i'. Then, for every occurrence of $\bar{x}_i$, the copy of $\bar{s}_1$ will be turned on and force $C \to E$, i.e., the corresponding edge in the clause part will be directed from T_{n+1} to T_1. The gadgets corresponding to the positive occurrence will be turned off; this means that GOOD may choose to direct the corresponding edges so that the potential path from T_{n+1} to T_1 is broken. The case in which GOOD sets x_i to **false** is symmetric.

Finally for (4), we have the precedence gadgets. For every pair of consecutive variables x_i, x_{i+1} we have gadgets which force the write steps of D_i and D_i' to come

before the write steps of D_{i+1} and D'_{i+1}, if the schedule is to be reliable. For example, to force the write step of D_i to come before that of D_{i+1}, we have a copy of s_2 (with a small alteration to be described in a moment) whose node D is identified with D_i and node C with D_{i+1}. The other two terminals of s_2 are ordinary transactions, but D' has empty write set, and E has in its write set one entity which is also written by D_{i+1} (terminal C), and no other transaction. We have similar copies of s_2 for the other three pairs (D_i, D'_{i+1}), (D'_i, D_{i+1}), (D'_i, D'_{i+1}). The purpose of these gadgets is to place restrictions on the ways that GOOD can rearrange after an abort, so that the rules of the QSAT game are not violated. To illustrate this point, suppose that our formula F is $\forall x_1 \exists x_2 \forall x_3 \exists x_4 [B]$, where B is a conjunction of clauses. After EVIL chooses a value for x_1, GOOD could rearrange the schedule to in effect transform the formula to $\forall x_1 \forall x_3 \exists x_2 \exists x_4 [B]$, that is, allow herself to look at EVIL's next move. These last gadgets prevent GOOD from cheating in this way; if she does, EVIL wins right away, even if F is true, since running C before D gives EVIL a winning game.

We will describe now how the various parts are put together in the schedule $s(F)$. First let us summarize what transactions there are. We have transactions $T_1, T_3, \cdots, T_{n+1}$ from part (1), and transactions belonging to various gadgets—copies of s_2 or $\bar{s}_1$. The various gadgets use distinct entities and transactions unless we have specified otherwise. Each gadget communicates with the rest of the world only through its terminals D, D', C, E for s_2, and D, 7, C, E for $\bar{s}_1$. Note that the terminals perform only read-steps in s_2 or $\bar{s}_1$.

The first portion of $s(F)$ executes all the steps of all the gadgets from parts (3) and (4), and also the read steps that are responsible for the arcs on the clause paths of part (2). If we allow multiple read steps for a transaction, then the steps from the different gadgets can be merged together in any arbitrary fashion. If we restrict ourselves to the 2-step model, some more care is required in synchronizing the read steps of the terminals. The only nodes that belong to more than one gadget are the transactions D_i, D'_i corresponding to the variables; note that each intermediate node in a clause path belongs to exactly one fan-out gadget. Nodes D_i and D'_i belong to several fan-out and precedence gadgets. Once we synchronize the read steps of the D_i's and D'_i's, then we can add easily (and independently of each other) the steps of the other transactions in the different gadgets. From Fig. 2 we see that in s_2, the read steps of C and E come before the read steps of D and D'. Therefore, because of the precedence gadgets, we must have read steps of D_n, D'_n before those of D_{n-1}, D'_{n-1} before $\cdots$ the read steps of D_1, D'_1. This is the only restriction: In s_2, either order between the read steps of D and D' is permitted; in $\bar{s}_1$, D does not read anything. Thus, the fan-out gadgets do not require any particular order between the read steps of D_i and D'_i.

Thus, in the first portion of $s(F)$, we execute in a consistent order all the copies of s_2 and $\bar{s}_1$, and then T_{n+1} executes its read step. At this point, parts (2)-(4) have been formed.

In the second portion of $s(F)$, $T_1, T_3, T_5, \cdots, T_{n+1}$ execute their write steps in this order. Now part (1) is also formed, and the stage is set for the game.

In the third portion of $s(F)$ we execute the write steps of the transaction D_i, D'_i in order of increasing i's, with D'_i immediately preceding D_i. After this, we execute the write steps of the dummy (output) terminals of the precedence gadgets.

Finally, in the fourth and final portion of $s(F)$, the transactions that correspond to literal occurrences (intermediate nodes on the clause paths) execute their write steps in an order consistent with a truth assignment that sets all variables to false. The construction of $s(F)$ is thus complete. It remains to show that $s(F)$ is reliable iff F is true.

First, we shall show that $s(F)$ is serializable. That is, the polygraph of the whole schedule has an acyclic orientation. Since all transactions D_i, D_i' are present, there is no $T_1 \to T_{n+1}$ path in part (1). Therefore, the clause paths cannot form a cycle. This means that the formula F plays no role, and the various gadgets are completely decoupled. (Note how crucial it is here that there are no directed paths between the input and the output terminals of the gadgets.)

In each gadget s_2, the fact that the input terminal D is present forces $C \to E$; this is the reason that we placed the dummy outputs of the precedence gadgets after the D_i's and D_i''s, and also that we use a default value of **false** (D_i present) for universal variables x_i. For existential variables x_i, we have $D_i' \to D_i$, corresponding $x_i =$ **false**, and this is correctly distributed to the clause parts.

Schedule $s(F)$ is reliable during the first two portions, in the sense that, if EVIL makes a move (aborts a transaction) before the end of the second portion, then GOOD can find a reliable continuation; and this, regardless of the value of F. The reason is that until the second portion, there no potential $T_1 \to T_{n+1}$ path. Thus, again, the paths of the clauses cannot close a cycle, and the different gadgets are completely decoupled. GOOD can win, because each individual gadget is reliable.

The third portion of the schedule is the crucial one. We shall show that $s(F)$ is reliable if and only if F is true.

IF: Suppose that F is true, i.e., the existential player has a winning strategy in QSAT. We shall show that GOOD has a winning strategy in the reliability game. Throughout the game, the current schedule resulting from GOOD's moves (rearrangements) has the following invariant:

(i) For each copy of s_2 and $\bar{s}_1$, the order of the write steps of the output terminals satisfies the serializability constraints;

(ii) For each copy of s_2, the reliability constraints are satisfied.

In (i), by "serializability constraint" we mean the ordering that may be forced between the output terminals. In the case of s_2, these are given by properties (1)–(3); for example, if D is present then $C \to E$. In the case of $\bar{s}_1$, the constraint is that $D \to 7$ (the write step of D before that of 7) implies $C \to E$. By "reliability constraints" in (ii), we mean that the write step of the input terminal D of s_2 comes before the write steps of the output terminals. Note that the invariant is true of our initial schedule $s(F)$.

After every move of EVIL, GOOD will do whatever rearrangements are necessary to restore the invariant. In particular, if EVIL aborts a transaction D_i, then GOOD will move the write steps of the dummy output terminals of the precedence gadgets for the pairs (D_i, D_{i+1}), (D_i, D_{i+1}') right before the write steps of D_{i+1} and D_{i+1}' respectively. In addition, if x_i is a universal variable, GOOD will reorder the write steps of the transactions corresponding to the occurrences of x_i and $\bar{x}_i$ in the clauses, to be consistent with $x_i =$ **true**. At all times, the write steps of all the nodes on the clause paths will be in the final (fourth) portion of the schedule. Similar rearrangements will be made by GOOD if a D_i' is aborted. Note that after the end of the second portion, all dummy input terminals of the precedence gadgets have terminated, and therefore, cannot be aborted anymore.

Given this invariant, the only cycle that EVIL can possibly force is one involving a path $T_1 \to T_{n+1}$ of part (1) and a path $T_{n+1} \to T_1$ from some clause of part (2). Therefore, if EVIL wants to have a chance at winning, he cannot pass up a chance to choose a truth value for a universal variable. That is, if for some odd i, both D_i and D_i' perform their write steps, without EVIL having aborted either of them, then GOOD has won: The path $T_1 \to T_{n+1}$ of part (1) is broken, and cannot be restored anymore.

We will describe now how GOOD responds to EVIL's moves before the write steps of D_1' and D_1 take place. As we mentioned before, there is no danger during the

first two portions. After that, the only active transactions are the ones corresponding to the variables (D_i, D_i'), the ones corresponding to literal occurrences and the dummy output terminals of the precedence gadgets. In fact, we can assume that EVIL does not abort a transaction except D_1, D_1'. Here is the reason: If EVIL aborts a transaction corresponding to a literal occurrence, then he has effectively forfeited the possibility of creating a cycle at the clause of the occurrence. Clearly this is a suboptimal move for EVIL, and we can assume it is not made. If EVIL aborts a dummy terminal, then he has destroyed the functionality of a precedence gadget. This implies that GOOD has the opportunity to rearrange future moves at will, a possibility that is also against EVIL's interests. Finally, if EVIL aborts some D_i or D_i' for $i > 1$, then GOOD restores the serializability and reliability constraints for the gadgets that are affected, as we described before, and does nothing else. This has the effect of EVIL revealing one of his subsequent moves ahead of time (i odd), or allowing GOOD to satisfy all clauses in which either x_i or $\bar{x}_i$ appear (i even). Thus, before D_1 and D_1' terminate, EVIL must abort one of them.

Suppose EVIL aborts D_1, corresponding to $x_1 = \textbf{true}$. GOOD sets the truth value of the existential variable x_2 according to the move of the existential player in response to the value of x_1. That is, GOOD possibly switches the write steps of D_2' and D_2 (if $x_k = \textbf{true}$); she also fans out the truth values to the clauses, and restores the serializability and reliability constraints of the affected gadgets. If EVIL aborts D_1', then GOOD responds in a symmetric way.

Subsequently, GOOD moves in exactly the same way: If x_i is the earliest universal variable whose value is still undetermined (both D_i and D_i' are present), GOOD waits until EVIL sets the value of x_i, making sure in the meantime only that the invariant is preserved. When EVIL sets the value of x_i, GOOD sets appropriately the value of x_{i+1} and continues similarly. Since F is true, all clauses will be eventually satisfied, and therefore no cycle will be formed.

ONLY IF: Suppose that F is false, i.e., the universal player has a winning strategy in QSAT. First, we observe that GOOD must move so that the invariant is continuously satisfied: If the serializability constraint of some gadget is violated, then the schedule is not serializable (the graph has a cycle) and EVIL has won; if a reliability constraint is violated, then EVIL can win in one more step (the schedule is not reliable, as we explained earlier).

EVIL follows the moves of the universal player. He starts by aborting D_1 or D_1' (according to whether $x_1 = \textbf{true}$ or **false**) right before the write step of D_1'. The precedence gadgets force GOOD to retain the order between transactions corresponding to different variables. GOOD has the freedom to switch any number of adjacent pairs of write steps of transactions D_i', D_i. Now, EVIL observes the value of x_2 chosen by GOOD, and correspondingly sets x_3 by aborting D_3 or D_3' right before they finish.

Note that now the values of x_1, x_2, x_3 have been set once and for all: GOOD cannot switch the write steps of D_2 and D_2' anymore. Because of the serializability constraints, GOOD must propagate the values to the clauses in a consistent way.

EVIL continues playing in this way until he sets properly all universal variables. Since F is false, some clause is not satisfied. The path $T_{n+1} \to T_1$ of this clause, together with the $T_1 \to T_{n+1}$ path of part (1), forms a cycle.

5. Other notions of reliability. In this section we study two other important notions of reliability. The first is related to *multiversion concurrency control* [7], [8]. In the multiversion context, a schedule s comes with a *version function* V, which assigns to each read step $a: R(x)$ a particular previous write step $V(a): W(x)$ on the same entity, such that a reads not necessarily the last value of x before a, but the value written by

$V(a)$. (Note that the image of V may be a write step of a fictitious *initial transaction*, which writes the initial values of the entities.) Also, V assigns to each write step $a: W(x)$ a previous write step on the same entity, the value of which is to be overwritten by a, or the symbols ν (for "new version") or λ (for "ignore this version"). We insist that no read or write step is mapped by V to a write step that is already overwritten. The pair (s, V) is called a *full schedule*. The version in which each step of s is mapped to the immediately previous write step on the same entity is called the *standard version function* of s. A full schedule is view-serializable if its steps read the same versions as some serial schedule with the standard version function.

It has been argued that multiple versions can improve the performance of concurrency control, and several schedulers have been proposed that support multiple versions [1], [7], [11]. Reliability is an important consideration in such schedulers as well. In fact, it can be argued [10] that multiple versions is the right context for discussing reliability issues, because recovery techniques are usually based on keeping multiple versions [6]. In view of this, it seems interesting to study the corresponding notion of reliability. The definition follows.

DEFINITION. Let (s, V) be a full schedule, and let q be a suffix of $s = pq$. We say that (s, V) is *q-reliable* if, for every step a of q $(q = q_1 a q_2)$ and each transaction A active at a, there is a full schedule (s', V') involving the transactions of s except a such that

(1) $s' = rq_2'$ where r contains the same steps as $pq_1 a$ in the same order (except of course for the steps of A which are necessarily missing from s'),
(2) q_2' is a rearrangement of the steps in q_2 (again, except for the steps of A),
(3) V' agrees with V on all steps of r, and
(4) (s', V') is q_2'-reliable.

For the basis of the recursion, a full schedule is λ-reliable, where λ is the empty suffix, if it is view-serializable. If a full schedule s is s-reliable, then it is called just *reliable*.

THEOREM 3. *Telling whether a full schedule is reliable is* PSPACE*-complete.*

Sketch of proof. The proof is very similar to that of Theorem 2, so we shall only sketch the necessary modifications. Up to the "dotted line," where the game starts, our full schedule will be exactly the one used in the proof of Theorem 2, *with the standard version function.* After the dotted line, the only difference is that transactions C and E corresponding to the gadgets s_2 and $\bar{s}_1$ do not just write a common entity (because this does not force an order in the case of multiple versions), but *both read and write* an entity, with C going first. As a result, the final order of these steps will determine whether $C \to E$, $E \to C$, or both (naturally, GOOD will avoid the latter possibility). Otherwise, the construction and the argument are identical. The full proof for this case can be found in [10].

Another variant of the notion of reliability is what we can call *rollback-reliability.* In our ordinary notion of reliability, we assume that the only means available to the scheduler after a failure is rearranging the steps that have not yet arrived. However, it is realistic to assume that the scheduler also has the option to *redo* certain steps of uncommitted transactions (we call such partial re-execution a *rollback*). As an extreme of this approach, the scheduler may *abort* an uncommitted transaction in reaction to some other abort (this is equivalent to redoing all steps of the transaction). This concept leads to the following definition.

DEFINITION. Let s be a schedule, and let q be a suffix of s $(s = pq)$. We say that s is *q-rollback reliable* if, for every step a of q $(q = q_1 a q_2)$ and each transaction A active at a, there is a schedule s' involving the transactions of s except A such that

(1) $s' = rq_2'$ where r contains the steps of pq_1a in the same order, except for the steps of A which are missing, *and possibly except for certain steps belonging to transactions uncommitted at a,*

(2) q_2' is a rearrangement of the steps in q_2, except for the steps of A, *plus the steps omitted from r,*

(3) s' is q_2'-rollback reliable.

For the basis of the recursion, a schedule is λ-rollback reliable, where λ is the empty suffix, if it is view-serializable. If a schedule s is s-rollback reliable, then it is called just *rollback reliable.*

Superficially, this definition is very much like the two previous ones, which lead to PSPACE-completeness results. However, rollback reliability turns out to be considerably simpler, and equivalent to the notion studied in [6], as established by the following theorem.

THEOREM 4. *Schedule s is rollback reliable if and only if for each prefix p of s, the schedule s_p is view-serializable, where s_p is s with all steps of transactions that were not committed at the end of p removed.*

Proof. For the **if** direction, suppose that s_p is view-serializable for all prefixes p of s, and consider any step a of s, $s = paq$, and any transaction A active at a. Then $s_p q'$ is an appropriate continuation from a, proving that s is rollback reliable, where q' is any serial execution of the transactions that were active at a except A. For the other direction, suppose that s is rollback reliable, and consider any prefix p of s. Once p is executed, we abort the immediately following step, and the first step of the continuation, and so on, until no more live transactions (or steps) remain. What we are left with is s_p, which must therefore be view-serializable. □

THEOREM 5. *Testing whether a schedule is rollback reliable is NP-complete.*

Sketch of proof. Just notice that the schedules produced in the standard proof of NP-completeness of view-serializability [9], [10] have the property that they are view-serializable if and only if all of their s_p's are view-serializable.

REFERENCES

[1] R. BAYER, H. HELLER AND A. REISER, *Parallelism and recovery in database systems,* ACM Trans. Database Systems, 5 (1980), pp. 139-156.

[2] A. CHANDRA, D. KOZEN AND L. J. STOCKMEYER, *Alternation,* J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[3] K. ESWARAN, J. GRAY, R. LORIE AND I. TRAIGER, *The notions of correctness and predicate locks in database systems,* Comm. ACM, 1976.

[4] M. GRAHAM, private communication.

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

[6] V. HADZILACOS, *A theory of reliability for database systems,* J. Assoc. Comput. Mach., to appear.

[7] T. HADZILACOS AND C. PAPADIMITRIOU, *Some algorithmic problems in multiversion concurrency control,* Proc. 1985 PODS Conference; J. Comput. System Sci., to appear.

[8] C. H. PAPADIMITRIOU AND P. KANELLAKIS, *On concurrency control by multiple versions,* ACM Trans. Database Systems, 9 (1984), pp. 89-99.

[9] C. H. PAPADIMITRIOU, *The serializability of concurrent database updates,* J. Assoc. Comput. Mach., 26 (1979), pp. 631-653.

[10] C. H. PAPADIMITRIOU, *The Theory of Database Concurrency Control,* Computer Science Press, Potomac, MD, 1986.

[11] R. E. STEARNS AND D. J. ROSENKRANTZ, *Distributed concurrency control using before-values,* Proc. ACM-SIGMOD, 1981, pp. 74-83.

[12] M. YANNAKAKIS, *Serializability by locking,* Proc. 13th STOC Conference, 1981.

SIAM J. COMPUT.
Vol. 16, No. 3, June 1987
© 1987 Society for Industrial and Applied Mathematics
008

TIGHTER BOUNDS FOR LPT SCHEDULING ON UNIFORM PROCESSORS*

DONALD K. FRIESEN†

Abstract. We examine the nonpreemptive assignment of independent tasks to a system of uniform processors with the objective of minimizing the makespan. The worst case bound for the largest processing time first algorithm for this problem is tightened to be in the interval (1.52, 1.67).

Key words. multiprocessor scheduling, worst-case performance, heuristic algorithms, uniform processors

AMS(MOS) subject classifications. 68Q, 90B

1. Introduction. A well-known deterministic scheduling problem concerns the nonpreemptive assignment of independent tasks to a set of processors in an effort to minimize the makespan (the total elapsed time from the start of execution until all tasks are completed). A multiprocessor system consists of a set of processors denoted by $P=\{P_1, P_2, \cdots, P_m\}$. We seek to best schedule a list $L=\{a_1, a_2, \ldots, a_n\}$ of tasks to P, each task, a, having a length or size $s(a)$. We restrict our attention to nonpreemptive scheduling, whereby a task, once assigned to a particular processor, may not be removed until it has finished execution. It is assumed that the elements of L are independent, i.e., do not require scheduling in accordance with any precedence constraints.

This problem has been shown to be NP-hard, and is therefore as intractable as those in a large class of notoriously difficult problems (see [GJ] and [Ul] for a detailed discussion). It is unlikely that there exists a polynomial time algorithm for producing a minimal makespan, so we consider heuristic algorithms in hope of providing near-optimal solutions.

When all elements of P are exactly the same, we say that we have an identical multiprocessor system. The LPT algorithm (the elements of L are first sorted into nonincreasing order of execution times) has been analyzed [Gr] and proved to have a "tight" worst-case performance bound of $4/3-1/(3m)$. Informally, this means that the length of an LPT schedule can be no more than about 33% longer than the optimum.

We consider here a more general model, that of a uniform multiprocessor system, in which the elements of P may differ in speed. We associate with P a set of relative speeds $R=\{r_1, r_2, \cdots, r_m\}$, where r_i denotes the ratio of the speed of P_i to that of P_1. This problem has drawn considerable attention [CS], [HS], [LL], [FL]. It was shown in [GIS] that the LPT algorithm can be implemented so that its worst case bound is between 1.5 and 2 times optimal. A recent paper [Do] showed that these bounds could be tightened to between 1.512 and 1.583. In this paper we show that the worst case bound for LPT is between 1.52 and 1.67 times optimal.

The same scheduling problem has been examined in [FL] where the MULTIFIT algorithm (introduced in [CGJ]) was modified to fit this problem and shown to have a worst case bound at most 1.4 times optimal.

2. Notation and a lower bound. We describe in this section the notation and the algorithm we use and then illustrate by an example that it may produce a schedule that is as much as 1.52 times the minimal schedule length.

* Received by the editors May 31, 1983; accepted for publication (in revised form) September 23, 1986.
† Department of Computer Science, Texas A&M University, College Station, Texas 77843.

In our version of the LPT algorithm, we assume that the tasks are arranged in nonincreasing order of size. The first (longest) task is assigned to the fastest processor. Then each task x is assigned by the following rule:

> Each processor P_i has a set (possibly empty) of tasks already assigned to it. If P_i executed task x after the other tasks already assigned to it, denote by t_i the time at which x would finish executing. Task x is then assigned to the processor P_j on which t_j is a minimum. Ties are broken arbitrarily.

We normalize the sizes of the tasks so that the smallest has size 1 and then measure speed so that a speed of 1 means execution of the smallest task would require 1 time unit on the processor. The speed of processor P_i will be denoted by α_i. This normalization effectively changes the "units" in which time is measured. Since both the LPT schedule length and the optimal length use the same units, the ratio is unchanged.

In order to visualize the problem more easily, it is convenient to model the problem a little differently. Rather than thinking of P_i as having speed α_i where the ratio α_i/α_1 relates the speed of P_i to that of the slowest processor P_1, we shall instead allow processor P_i to have α_i/α_1 times the amount of time allotted to P_1. In this way a task will have the same size no matter which processor executes it. Then when we say that the LPT algorithm generates a schedule with makespan at most M, we must know that for each processor P_i, the tasks assigned to it will be completed by time $M\alpha_i$, that is, the sum of the sizes, $s(P_i)$, must be less than or equal to $M\alpha_i$.

The following example should make this notation clearer, as well as show the worst case we have been able to find:

There is one processor of speed 4, three processors of speed 2.468 and five processors of each of the following speeds: $1.923,\ 1.923-\varepsilon,\ 1.923-2\varepsilon,\ \cdots,\ 1.234+\varepsilon$ where ε is some very small number dividing $1.923-1.234$ evenly, for example, .00001. The set of sizes of tasks are as follows: five tasks of each of the sizes $1.923,\ 1.923-\varepsilon,\ 1.923-2\varepsilon,\ \cdots,\ 1.234+\varepsilon$, six tasks of size 1.234 and four tasks of size 1.0. The optimal schedule is given in Fig. 2.1. Figure 2.2 shows the result of applying the LPT algorithm, with one task of size 1 that cannot be scheduled within the makespan of 1.52. In each figure, the numbers inside the boxes are the sizes of the tasks assigned, the first number

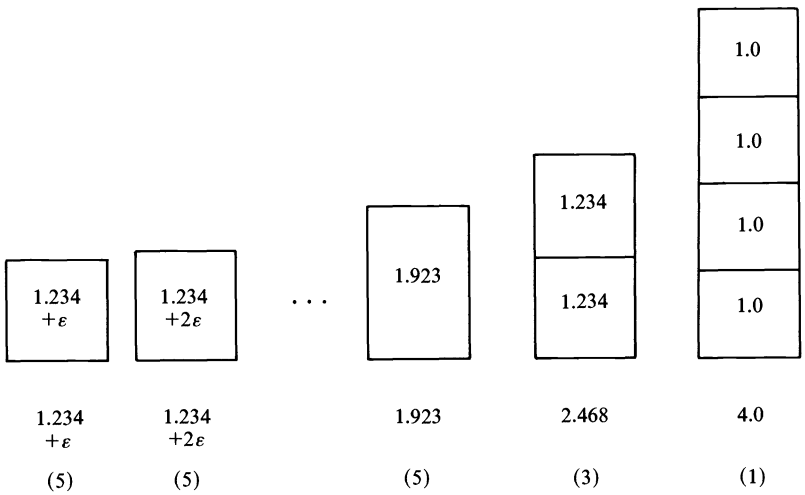


FIG. 2.1. *Optimal schedule.*

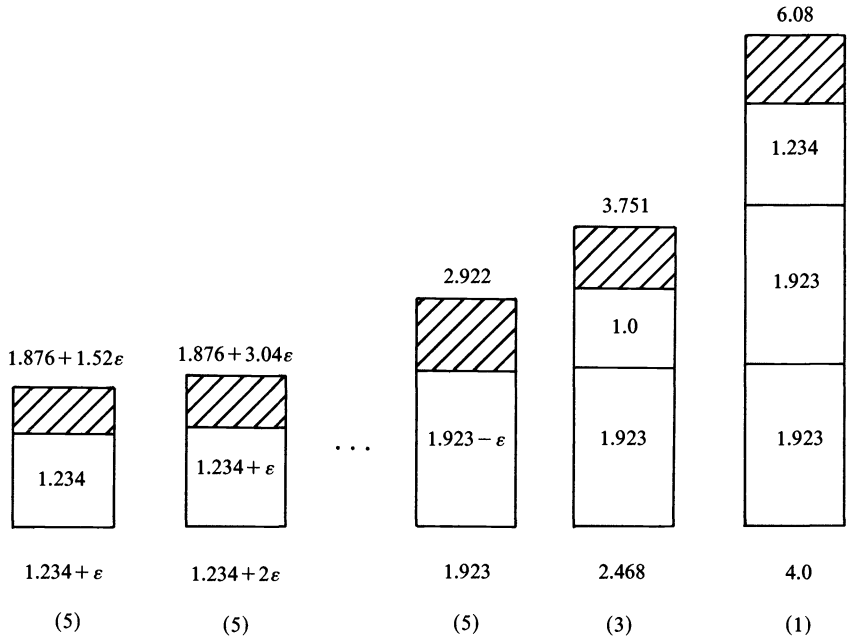


FIG. 2.2. LPT *schedule.*

below each box represents the speed of that processor and the number in parentheses represents the number of processors exactly like it.

3. Upper bound proof. In this section we show that the length of the schedule obtained by the LPT algorithm is at most 5/3 times the optimal length. The proof, like many others in the area of scheduling, will be based on assuming a minimal counterexample and deriving a contradiction via a weighting function argument.

By a minimal counterexample we mean a counterexample with the fewest possible processors and with the fewest possible tasks for that number of processors. Certainly if a counterexample exists, there is a minimal counterexample. For convenience of notation we assume that the optimal makespan for a minimal counterexample is 1.

Two observations we can make concerning minimal counterexamples are that we can assume that our LPT schedule was within 5/3 times optimal until the last task of size 1 was assigned, and that we can apply the principle of domination (analogous to the domination of [CGJ]). We state this formally below as a lemma. We shall select a particular optimal schedule and algorithm and compare our schedule and algorithm with these in the remainder of the paper.

DEFINITION. We say that processor P dominates processor P' if

(i) $\alpha \leqq \alpha'$ (P' is at least as fast as P), and

(ii) the LPT algorithm assigns P the tasks $a_1, \cdots, a_k$, the optimal algorithm assigns P' the tasks $b_1, \cdots, b_l$ and there is a function f:

$\{b_1, \cdots, b_l\} \dashrightarrow \{a_1, \cdots, a_k\}$ such that for each a_j

$\sum s(b_i) \leqq s(a_j), \qquad f(b_i) = a_j.$

LEMMA 1. *If the multiprocessor system M and list of tasks L is a minimal counterexample, no processor P dominates a processor P'.*

Proof. If P dominated P', then deleting processor P and all of the tasks assigned to it by LPT would result in a counterexample with fewer processors. The deletion would not affect the assignment of other tasks by the LPT algorithm.

The optimal algorithm could assign all tasks assigned to P to P'. Since P' is at least as fast as P, this can be done without increasing the makespan. All tasks optimally assigned to P' can be assigned to processors containing deleted tasks: a deleted task a_i can be replaced by all b_j such that $f(b_j) = a_i$. By the definition of domination, this too will not increase the optimal makespan. Thus the system $M - \{P\}$ and list $L - \{a_i \mid a_i \in P\}$ constitute a smaller counterexample.

We assume from now on that we are dealing with a minimal counterexample. From now on, it should be clear from the context what order the processors occur, and we shall omit the subscripts on the processors and their speeds.

LEMMA 2. *In the* LPT *scheduling, each processor P is busy until after time* $(5/3)\alpha - 1$.

Proof. If this were not the case, the last task (the only one that did not meet the 5/3 times optimal deadline) would execute on P within the required time.

LEMMA 3. *No processor P with $\alpha \geqq 3/2$ has exactly one task assigned to it by the* LPT *algorithm.*

Proof. Suppose that $\alpha \geqq 3/2$ and that the LPT algorithm assigns P the task x alone. Then, by Lemma 2, $s(x) \geqq (5/3)\alpha - 1 \geqq \alpha$. But then P dominates itself, contradicting Lemma 1.

For the remainder of the proof, we will use a weight function w which assigns to each task x a real number depending on the size of x.

DEFINITION. For any task x in L, $w(x)$ is defined as follows:

$$\begin{aligned} w(x) &= 1 && \text{if } s(x) \leqq 3/2, \\ &= (2/3)s(x) && \text{if } 3/2 < s(x) \leqq 11/6, \\ &= (8/11)s(x) && \text{if } 11/6 < s(x) \leqq 5/2, \\ &= (4/5)s(x) && \text{if } 5/2 < s(x). \end{aligned}$$

LEMMA 4. *If x is any task, then* $w(x) \geqq (2/3)s(x)$.

Proof. Trivial if $s(x) > 3/2$, and if $s(x) \leqq 3/2$ then $1/s(x) \geqq 2/3$.

We extend w and s to processors in the following way, where we have used the notation $x \in P$ and $x \in {}^*P$ to mean x is assigned to P by the LPT algorithm, and the optimal algorithm, respectively:

$$w(P) = \sum_{x \in P} w(x), \quad s(P) = \sum_{x \in P} s(x), \quad w^*(P) = \sum_{x \in {}^*P} w(x).$$

LEMMA 5. *If $w^*(P) > w(P)$, then the tasks assigned to P by the optimal algorithm all have weight* 1.

Proof. Let q be the number of tasks of weight 1 assigned to P by the optimal algorithm, and let r, l, and k be the sum of sizes of tasks x of weight $(2/3)s(x)$, $(8/11)s(x)$, and $(4/5)s(x)$, respectively, assigned by the optimal algorithm to P. Then

$$w^*(P) = q + (2/3)r + (8/11)l + (4/5)k$$

where $r = 0$ or $r > 3/2$, $l = 0$ or $l > 11/6$, $k = 0$ or $k > 5/2$, and $q + r + l + k \leqq \alpha$.

$$\begin{aligned} \text{Now } w(P) &\geqq (2/3)s(P) \\ &\geqq (2/3)((5/3)\alpha - 1) = (10/9)\alpha - 2/3 \\ &\geqq (10/9)(q + r + l + k) - 2/3. \end{aligned}$$

If $w(P^*) > w(P)$ then

$$q + (2/3)r + (8/11)l + (4/5)k > (10/9)q + (10/9)r + (10/9)l + (10/9)k - 2/3,$$

and simplifying this inequality produces

$$2/3 > (1/9)q + (4/9)r + (38/99)l + (14/45)k.$$

If k is not 0, then

$$2/3 > (14/45)(5/2) = 7/9.$$

If l is not 0, then

$$2/3 > (38/99)(11/6) = 19/27.$$

If r is not 0, then

$$2/3 > (4/9)(3/2) = 2/3.$$

Since all of these are impossible, we see that $r = l = k = 0$ and all tasks assigned have weight 1.

The bulk of the upper bound proof now is a case analysis looking at the possible values of α, showing in each case that $w(P) \geqq w^*(P)$. These will be proved in a sequence of lemmas. The final theorem will then be proved by a simple counting argument.

LEMMA 6. *If* $\alpha < 3$, *then* $w^*(P) \leqq w(P)$.

Proof. Suppose $\alpha < 3$ and $w^*(P) > w(P)$. There are at most 2 tasks assigned to P by the optimal algorithm. Hence by Lemma 5, $w^*(P) \leqq 2$. If $\alpha < 3/2$, then $w^*(P) \leqq 1 \leqq w(P)$ so we can assume $\alpha \geqq 3/2$. Now by Lemma 3, processor P has at least 2 tasks assigned to it by the LPT algorithm and $w(P) \geqq 2 \geqq w^*(P)$, contradicting our assumption.

LEMMA 7. If $3 \leqq \alpha < 4$, then $w^*(P) \leqq w(P)$.

Proof. The argument is by contradiction. Suppose $3 \leqq \alpha < 4$ and $w^*(P) > w(P)$. Using Lemma 5, $w^*(P) \leqq 3$. If P is assigned at least 3 tasks, $w(P) \geqq 3$. Thus if the lemma is false, P must be assigned exactly 2 tasks by the LPT algorithm, say x, y with $s(x) \geqq s(y)$. (P cannot have just 1 task by Lemma 3.) If P has 2 or fewer tasks assigned by the optimal algorithm, then $w^*(P) \leqq 2$ and since $w(P) \geqq 2$, we have a contradiction. Thus we assume that P has 3 tasks a, b, c with $s(a) \geqq s(b) \geqq s(c)$ assigned by the optimal algorithm.

Suppose $s(y)$, the size of the smallest task assigned to P by the LPT algorithm, were smaller than $s(a)$, the size of the largest task assigned by the optimal algorithm.

Then $s(y) + s(x) > (5/3)\alpha - 1$, and

$$\begin{aligned} s(x) &> (5/3)(s(a) + s(b) + s(c)) - 1 - s(y) \\ &> (5/3)(s(a) + s(b) + s(c)) - 1 - s(a). \end{aligned}$$

Simplifying we get

$$s(x) > (5/3)(s(b) + s(c)) + (2/3)s(a) - 1 \geqq 3,$$

since each task has size $\geqq 1$. Hence

$$w(x) = (4/5)s(x) > (4/5)3 = 12/5,$$

and

$$w(P) = (4/5)3 + 1 > 3 = w^*(P).$$

Thus we can conclude that each task assigned to P by the LPT algorithm is at least as large as each task assigned to P by the optimal algorithm.

If $s(x) \geqq s(b)+s(c)$, then P dominates itself contradicting Lemma 1. Hence $s(y) \leqq s(x) < s(b)+s(c)$. Then

$$(5/3)(s(a)+s(b)+s(c))-1 < s(x)+s(y) < 2(s(b)+s(c)).$$

Simplifying yields

$$(5/3)s(a) < 1+(1/3)s(b)+(1/3)s(c) \leqq (5/3)s(b),$$

contradicting $s(a) \geqq s(b)$.

LEMMA 8. *If* $4 \leqq \alpha < 5$, *then* $w^*(P) \leqq w(P)$.

Proof. $w^*(P)=4$ since P can be assigned at most four tasks by the optimal algorithm, and it is easy to see that the lemma holds if less than four are assigned. Certainly the lemma holds then if P is assigned four or more tasks by the LPT algorithm. Also, by Lemma 3, P must be assigned at least two tasks.

Suppose P is assigned two tasks. If both are at least $5/2$ in size, $w(P)=(4/5)s(P) > (4/5)5=4$. If neither is, $s(P) < 2(5/2) < (5/3)\alpha - 1$. Thus one task must be at least $5/2$ in size, and $w(P) \geqq (4/5)s(x)+(2/3)s(y)$, where $s(x) \geqq s(y)$ and P is assigned x and y. The function

$$(4/5)s(x)+(2/3)s(y)$$

is a minimum when $s(x)$ is as small as possible since $s(x)+s(y)$ is fixed. Hence

$$w(P) > (4/5)(5/2)+(2/3)(s(P)-5/2).$$

Since

$$s(P) > (5/3)\alpha - 1 \geqq (5/3)4-1 = (17/3),$$

$$w(P) > 2+(2/3)(17/3-5/2) > 4 = w^*(P).$$

Suppose now that P is assigned three tasks: x, y, and z with $s(x) \geqq s(y) \geqq s(z)$. If $s(y) > 11/6$, we would have

$$\begin{aligned} w(P) &> (8/11)s(x)+(8/11)s(y)+(2/3)s(z) \\ &\geqq (8/11)(11/6)+(8/11)(11/6)+(2/3)(17/3-11/3) \\ &\geqq 4. \end{aligned}$$

If $s(x) > 5/2$ then $w(P) > (4/5)(5/2)+2=4$. Hence

$$11/6 < s(x) \leqq 5/2 \quad \text{and} \quad s(z) \leqq s(y) \leqq 11/6.$$

Let P be assigned the four tasks a, b, c, d with $s(a) \geqq s(b) \geqq s(c) \geqq s(d)$ by the optimal algorithm. If $s(z) \leqq s(a)$, then

$$\begin{aligned} s(x)+s(y) &> (5/3)(s(a)+s(b)+s(c)+s(d))-1-s(a) \\ &\geqq 14/3 \quad \text{since } s(d) \geqq 1. \end{aligned}$$

Now $s(y) \leqq 11/6$ implies $s(x) > 14/3-11/6 > 5/2$. Hence each task assigned to P by LPT is larger than any task assigned optimally to P.

Thus $s(x) < s(c)+s(d)$ or else P dominates itself. But then

$$s(x)+s(y)+s(z) < s(c)+s(d)+s(y)+s(z)$$

and

$$(5/3)(s(a)+s(b)+s(c)+s(d))-1 < s(c)+s(d)+s(y)+s(z).$$

Since $s(a)+s(b) \geqq s(c)+s(d)$,

$$(10/3)(s(c)+s(d))-1 < s(c)+s(d)+s(y)+s(z)$$

and

$$11/3 \leqq (7/3)(s(c)+s(d))-1 < s(y)+s(z).$$

But then $s(y) > 11/6$, contradicting $w(y) = (2/3)s(y)$.

LEMMA 9. *If* $\alpha \in [5, 6)$, *then* $w^*(P) \leqq w(P)$.

Proof. As before we may assume that $w^*(P) = 5$ and that P is assigned at most four tasks by LPT.

$$s(P) > (5/3)\alpha - 1 \geqq 22/3.$$

If P is assigned two tasks at least one is $11/3$ in size and $w(P)$ would be greater than $(11/3)(4/5)+(2/3)(22/3-11/3) > 5$.

If P is assigned three tasks, at least one is $>22/9$ in size, and $w(P)$ is greater than $(8/11)(22/9)+(2/3)(22/3-22/9)$ which is greater than 5.

If P is assigned four tasks, at least one is $>11/6$ and $w(P)$ is at least $(8/11)(11/6)+(2/3)(22/3-11/6) = 5$.

LEMMA 10. *If* $\alpha \geqq 6$, *then* $w^*(P) \leqq w(P)$.

Proof. $s(P) \geqq (5/3)\alpha - 1$, and hence

$$w(P) \geqq (2/3)((5/3)\alpha - 1) = (10/9)\alpha - 2/3.$$

Since $(10/9)\alpha - 2/3 \geqq \alpha \geqq w^*(P)$ for $\alpha \geqq 6$, the lemma is proved.

THEOREM. *If there is a schedule achieving a makespan* ω, *then the* LPT *schedule achieves a makespan of at most* $(5/3)\omega$.

Proof. If not, then for the minimal counterexample we know that at the time no further tasks can be scheduled by the LPT algorithm,

$$w^*(P) \leqq w(P) \quad \text{for each processor } P.$$

Hence

$$\sum_{x \in L} w(x) = \sum_{p} w^*(P) \leqq \sum_{p} w(P) = \sum_{x \in L'} w(x)$$

where L' is the subset of tasks assigned by the LPT algorithm. Hence $L' = L$ and all tasks must be assigned.

REFERENCES

[CGJ] E. G. COFFMAN, JR., M. R. GAREY AND D. S. JOHNSON, *An application of bin-packing to multiprocessor scheduling*, this Journal, 7 (1978), pp. 1-17.

[CS] Y. CHO AND S. SAHNI, *Bounds for list schedules on uniform processors*, this Journal, 9 (1980), pp. 91-103.

[Do] G. DOBSON, *Scheduling independent tasks on uniform processors*, this Journal, 13 (1984), pp. 705-716.

[FL] D. K. FRIESEN AND M. A. LANGSTON, *Bounds for* MULTIFIT *scheduling on uniform processors*, this Journal, 12 (1983), pp. 60-70.

[GIS] T. GONZALES, O. H. IBARRA AND S. SAHNI, *Bounds for* LPT *schedules on uniform processors*, this Journal, 6 (1977), pp. 155-166.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, 1979.

[Gr] R. L. GRAHAM, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416-429.

[HS] E. HOROWITZ AND S. SAHNI, *Exact and approximate algorithms for scheduling non-identical processors*, J. Assoc. Comput. Mach., (1976), pp. 317-327.

[LL] J. W. S. LIU AND C. L. LIU, *Bounds on scheduling algorithms for heterogeneous computing systems*, Proc. International Federation of Information Processing Societies, 1974, pp. 349-353.

[Ul] J. D. ULLMAN, *Complexity of sequencing problems*, in Computer and Job-Shop Scheduling Theory, E. G. Coffman, ed., John Wiley, New York, 1976, Chap. 4.

SIAM J. COMPUT.
Vol. 16, No. 3 June 1987

© 1987 Society for Industrial and Applied Mathematics
009

ON SHORTEST PATHS AMIDST CONVEX POLYHEDRA*

MICHA SHARIR†

Abstract. Let K be a 3-D convex polyhedron having n vertices. A sequence ξ of edges of K is called a *shortest-path* sequence if there exist two points X, Y on the surface S of K such that ξ is the sequence of edges crossed by the shortest path from X to Y along S. We show that the number of shortest-path sequences for K is polynomial in n, and as a consequence prove that the shortest path between two points in 3-space which must avoid the interiors of a fixed number of disjoint convex polyhedral obstacles, can be calculated in time polynomial in the total number of vertices of these obstacles (but exponential in the number of obstacles).

Key words. Euclidean shortest paths, convex polyhedra, computational geometry

AMS(MOS) subject classifications. 51M20, 52A15, 52A25, C8Q20, 68Q25

1. Introduction. In this paper we study several problems related to the problem of calculating the Euclidean shortest path between two points in three-dimensional space, which must avoid the interiors of a collection of polyhedral obstacles having altogether n vertices. This general problem seems to be intractable, and the only known algorithms for it require exponential time [SS], [RS], although no nontrivial lower bounds are known as yet for this problem. However, if one restricts the problem to that of finding the shortest path between two points in 3-space which must avoid the interior of a single convex polyhedral obstacle, then the problem becomes solvable in time $O(n^2 \log n)$ [SS], [Mo].

Interpolating between these two extreme cases, one might consider the problem in which the polyhedral obstacles consist of a fixed number k of disjoint convex polyhedra (having altogether n vertices), and attempt to calibrate the complexity of this problem as a function of k and n.

Baltsan and Sharir [BS] have recently studied the case $k=2$, and have obtained an algorithm for this case running in polynomial time (almost proportional to $O(n^3 \log n)$). However, even for $k=3$ the problem becomes quite difficult. It is easy to see that, given three disjoint convex polyhedra A, B, C, the shortest path between two points X, Y lying respectively on the surfaces of the polyhedra A, C, which also travels along the surface of B, must consist of the following portions (see Fig. 1):

(i) a shortest path along the surface of A from X to a first "take-off" point t_1;
(ii) a straight segment from t_1 to a first "landing" point l_1 on B;
(iii) a shortest path along the surface of B from l_1 to a second take-off point t_2;
(iv) a straight segment from t_2 to a second landing point l_2 on C;
(v) a shortest path along the surface of C from l_2 to Y.

It is also easily seen that the take-off and landing points must lie on edges of the corresponding polyhedra (unless they coincide with X or Y themselves).

If one knew the sequences of edges on A, B, and C crossed by the three subpaths (i), (iii), (v), respectively, then, using the planar unfolding procedure described in [SS], one could unfold the faces of A (resp. B, C) crossed by the subpath (i) (resp. (iii), (v)) so that they all lie in a common plane, and so that the corresponding subpath

* Received by the editors November 4, 1985; accepted for publication October 2, 1986. This work was supported in part by Office of Naval Research grant N00014-82-K-0381, National Science Foundation grant NSF-DCR-83-20085, and by grants from the Digital Equipment Corporation, and the IBM Corporation.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

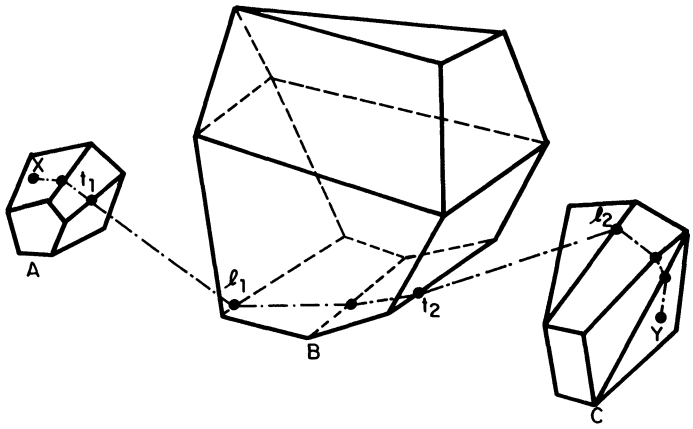


FIG. 1

becomes a straight segment. This reduces the problem to that of finding a shortest path between two points, which is constrained to pass through four specific edges (namely the edges containing the take-off and landing points). This latter problem can then be solved in constant time, using the technique described in [SS].

The subpaths (i) and (v) start or terminate at fixed known points, and the results of [SS], [Mo] show that, for a polyhedron with n vertices, there are at most $O(n^2)$ sequences of edges which can be crossed by shortest paths along that polyhedron, emerging from a fixed source point. However, the subpath (iii) starts and ends at unknown points, so that, a priori, there could be an exponential number of possible sequences of edges of B that could be crossed by the subpath (iii).

In this paper we show that there are only $O(n^7)$ possible sequences of edges of a convex polyhedron K with n vertices, which could be crossed by shortest paths along the surface of K between any two source and target points. This implies that the number of edge sequences that have to be explored in searching for our desired shortest path is only polynomial, so that shortest paths amidst three convex polyhedra can be calculated in time polynomial in the total number of their vertices. In fact, this observation can be easily generalized to yield polynomial-time algorithms for finding shortest paths amidst any fixed number of convex polyhedral obstacles.

The paper is organized as follows. In § 2 we prove our main result about a polynomial upper bound on the number of sequences of edges crossed by shortest paths along a convex polyhedron. In § 3 we extend the observations made above to obtain polynomial time shortest-path algorithms for any fixed number of convex polyhedral obstacles.

2. On the number of shortest-path edge sequences on a convex polyhedron. Let K be a 3-D convex polyhedron having n vertices. Let S be the surface of K. For each pair of points A, B in S we denote by $\pi(A, B)$ the shortest path along S from A to B, if this path is unique, and by $\xi(A, B)$ the sequence of edges of K crossed by $\pi(A, B)$ (if A or B lies on an edge, we include this edge in $\xi(A, B)$). If there exists more than one shortest path from A to B, $\pi(A, B)$ will denote any one of these paths. We call $\xi(A, B)$ a *shortest-path edge sequence* (induced by A and B). Figure 2 illustrates these definitions. It is easily seen that $\pi(A, B)$ cannot cross an edge of K more than once; thus $\xi(A, B)$ consists of distinct edges.

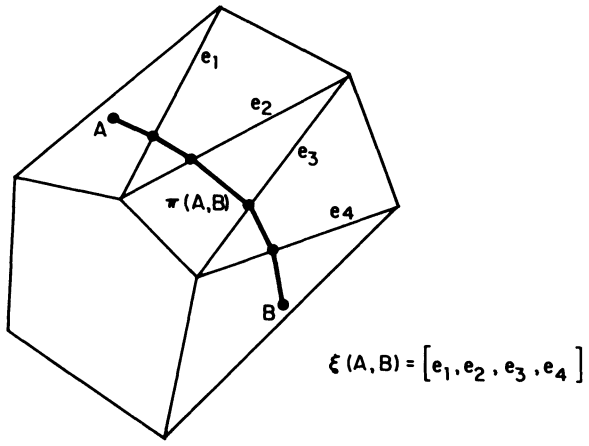


FIG. 2

We first recall a few basic properties of shortest paths along S (for more detail of which see [SS]). Let A, B be two points on S, then $\pi(A, B)$ does not pass through a vertex of K and is *geodesic*, in the sense that it is a polygonal path which bends only at edges of K, such that, with each edge that it crosses it subtends equal incoming and outgoing angles. B is called a *ridge* point relative to A if there exists more than one shortest path from A to B. A shortest path from A cannot path through a ridge point relative to A. The collection of ridge points relative to A consists of $O(n^2)$ straight segments linked together so as to form a tree whose leaves are vertices of K, and whose nodes of degree 2 are points lying on edges of K. The ridge points partition each edge of K into $O(n)$ intervals such that for each such interval I, the sequences $\xi(A, C)$, for $C \in I$, are all the same. When the surface of K is unfolded so as to make all edges in such a sequence co-planar, the source point A is mapped into a "source image" A_I, and the shortest paths $\pi(A, C)$, for $C \in I$, all become straight segments emanating from A_I. The collection of all source images, for a fixed edge e of K, forms a set of points whose (planar) Voronoi diagram intersects e precisely at the ridge points along e.

Unlike the case studied in [SS], [Mo], in which only shortest paths from a fixed source point are sought, here we consider the case in which the source point A varies along S, so that the associated ridges and their intersections with the edges of K all vary with A. To overcome this new technical difficulty and reduce the problem to purely discrete terms, we proceed as follows.

LEMMA 2.1. *Let A, B, C be three distinct points on S. Then either one of the paths $\pi(A, B)$, $\pi(A, C)$ is a subpath of the other, or these two paths meet only at A.*

Proof. This is well known (cf. [SS]), and is proved as follows. Suppose neither of the paths $\pi(A, B)$ and $\pi(A, C)$ is a subpath of the other, and that they meet at some point D. Then the lengths of their portions between A and D must be equal, so that D must be a ridge point relative to A, contradicting the property that shortest paths cannot pass through a ridge point. □

LEMMA 2.2. *Let A, B, C, D be four distinct points on S. Then the paths $\pi(A, B)$ and $\pi(C, D)$ intersect in at most one point, unless either one of these paths is a subpath of the other, or their intersection is a shortest path between one of the points A, B and one of the points C, D.*

Proof. Suppose that the special cases just mentioned do not occur, and that the paths $\pi(A, B)$ and $\pi(C, D)$ intersect in two points X, Y. Without loss of generality

suppose that Y lies on $\pi(A, B)$ between X and B, and that it lies on $\pi(C, D)$ between X and D. Then the shortest paths $\pi(X, B)$ and $\pi(X, D)$ are subpaths of the paths $\pi(A, B)$, $\pi(C, D)$ respectively, and by Lemma 2.1 they cannot intersect at Y, a contradiction which proves the claim. □

To estimate the number of shortest-path edge sequences along S, we first consider the restricted case in which the source point A of the shortest path lies on an edge e_1 of K, and the target point B lies on another edge e_2. Our approach is to consider the 2-D space $Z = e_1 \times e_2$ of all possible such pairs of source and target points, and partition it into regions, such that for each such region R there exists a small set $C(R)$ of edge sequences such that $\xi(A, B) \in C(R)$ for all $(A, B) \in R$.

To obtain these regions, consider first a fixed pair $(A, B) \in Z$ such that neither A nor B is a ridge point relative to any vertex of K. For each vertex X of K let $\gamma(X)$ denote the concatenation of (the unique paths) $\pi(A, X)$ and $\pi(X, B)$. Then we have

LEMMA 2.3. (a) *Let X and Y be distinct vertices of K. Then $\gamma(X)$ and $\gamma(Y)$ intersect in at most one point (in addition to their common endpoints A and B), unless both A and B lie on $\pi(X, Y)$.*

(b) *For each vertex X of K, the path $\pi(A, B)$ does not intersect $\gamma(X)$ (except at the points A and B), unless either A lies on $\pi(B, X)$ or B lies on $\pi(A, X)$.*

Proof. (a) Note that, by Lemma 2.1, the paths $\pi(A, X)$ and $\pi(A, Y)$ can meet one another only at A, because $\pi(A, X)$ cannot pass through the vertex Y, and vice versa. Similarly, the paths $\pi(X, B)$ and $\pi(Y, B)$ can meet only at B. By Lemma 2.2 the paths $\pi(A, X)$ and $\pi(Y, B)$ intersect in at most one point, unless they overlap in one of the manners stated there. Since shortest paths cannot pass through vertices, it is easy to check that the only way in which these paths can overlap is when both A and B lie on $\pi(X, Y)$. Similarly, unless this special case occurs, $\pi(X, B)$ and $\pi(A, Y)$ also intersect in at most one point, so that altogether $\gamma(X)$ and $\gamma(Y)$ intersect in at most two points.

Even though this result suffices for subsequent analysis, we can in fact show that $\gamma(X)$ and $\gamma(Y)$ cannot intersect in more than a single point. Indeed, suppose that these paths intersect in two points C and D. The preceding argument then implies that one of these points, say C, is an intersection of the subpaths $\pi(A, Y)$ and $\pi(B, X)$, and the other point D is an intersection of the subpaths $\pi(A, X)$ and $\pi(B, Y)$. Let $|\pi|$ denote the length of a path π. Then we have

$$|\gamma(X)|+|\gamma(Y)| = [|\pi(A, C)|+|\pi(C, X)|+|\pi(X, D)|+|\pi(D, B)|] + [|\pi(B, C)|+|\pi(C, Y)|+|\pi(Y, D)|+|\pi(D, A)|]$$

because the union of the paths in either side of this equality consists of the same segments. But since the path $\gamma(X)$ (resp. $\gamma(Y)$) is the shortest among all paths that connect A to B via X (resp. Y) it follows that

$$|\gamma(X)| = |\pi(A, C)|+|\pi(C, X)|+|\pi(X, D)|+|\pi(D, B)|,$$

and similar reasoning shows that

$$|\pi(A, X)| = |\pi(A, C)|+|\pi(C, X)|.$$

This, however, implies that A is a ridge point relative to the vertex X of K, contrary to assumption.

(b) Suppose that $\pi(A, B)$ intersects $\gamma(X)$. Then without loss of generality we can assume that $\pi(A, B)$ intersects $\pi(A, X)$, which, by Lemma 2.1, is possible only if B lies on $\pi(A, X)$ (X cannot lie on $\pi(A, B)$ because shortest paths cannot pass through vertices of K). □

Let us now draw all the paths $\gamma(X)$, for X a vertex of K. They collectively partition S into $O(n^2)$ (simply) connected regions, and Lemma 2.3(b) implies that $\pi(A, B)$ is wholly contained in just one of these regions. Let R be that region; the boundary of R then contains both A and B, and we denote by δ_1, δ_2 the two portions of this boundary connecting A to B. Note that R contains no vertex of K (except possibly on δ_1 and δ_2). Hence if e is an edge of K which intersects $\pi(A, B)$ (that is, $e \in \xi(A, B)$) then e must also intersect both δ_1 and δ_2, where each of these intersections can also be at an endpoint of e. Conversely, if e is an edge of K which intersects both δ_1 and δ_2, then it is clear that e must also intersect $\pi(A, B)$, i.e., $e \in \xi(A, B)$.

Furthermore, let e be an edge which intersects both δ_1 and δ_2. Let e' be a subsegment of e whose endpoints p_1, p_2 lie on δ_1, δ_2, respectively, and whose interior is wholly contained in R. We claim that if $\pi(A, B)$ is indeed contained in R then it must intersect e'. This is because e' splits R into two disjoint subregions, one of which contains A on its boundary and the other contains B on its boundary. Thus any continuous arc connecting A to B within R must cross e'.

This implies that if $\pi(A, B)$ is contained in R, then, once e is chosen, the subsegment e' defined above must be unique, for if e contained two such subsegments, $\pi(A, B)$ would have to intersect both of them, contradicting the fact that a shortest path cannot cross an edge of K more than once.

Finally, suppose that $\pi(A, B)$ is contained in R, and let d, e be two edges of K intersecting both δ_1, δ_2. Let $d' = p_1p_2$, $e' = q_1q_2$, be their unique subsegments connecting δ_1 to δ_2 as defined above, with $p_1, q_1 \in \delta_1$, and $p_2, q_2 \in \delta_2$. Then we claim that p_1 is nearer to A along δ_1 than q_1 if and only if p_2 is nearer to A along δ_2 than q_2. This property follows by general topological considerations from the fact that d' and e' do not intersect one another, and that each of them partitions R into two disjoint portions, so that the other subsegment is contained in just one of them. Moreover, similar considerations imply that p_1 is nearer to A along δ_1 than q_1 if and only if d precedes e in the sequence $\xi(A, B)$. See Fig. 3 for an illustration of these observations.

All these observations lead to the following procedure for calculating a small collection $C(A, B)$ of potential edge sequences, which $\xi(A, B)$ can assume.

PROCEDURE EDGE_SEQUENCES (A, B):

(I) Calculate all shortest paths $\pi(A, X)$, $\pi(X, B)$, for X a vertex of K, to obtain the curves $\gamma(X)$. Find all the intersection points between these curves, and obtain the decomposition of S into open connected regions by these curves.

(II) For each such region R whose boundary contains both A and B, find all intersections of edges of K with each of the half-boundaries δ_1, δ_2 of R connecting A to B.

(III) Find the collection Σ of all edges of K intersecting both δ_1 and δ_2. For each edge e in Σ check whether it contains a unique subsegment e' connecting a point p_1 on δ_1 to a point p_2 on δ_2, such that the interior of e' is fully contained in R. If not all edges in Σ have such unique subsegments, R is screened out. Otherwise, sort Σ by the order of the endpoints of the subsegments e' either along δ_1 or along δ_2, to obtain a sequence ξ of edges.

(IV) Output the collection $C(A, B)$ of all sequences ξ calculated by the preceding step.

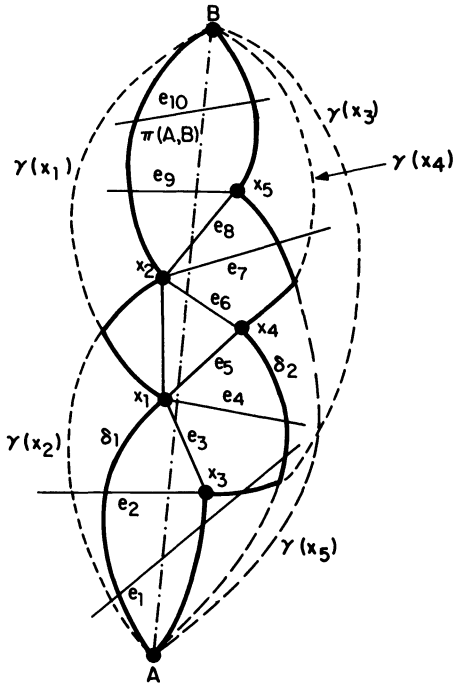


FIG. 3

It is easy to check that $C(A, B)$ contains at most n sequences, because at most n regions R can contain A or B on their boundaries (consider the fan of the n curves $\gamma(X)$ emanating, say, from A; each region R containing A on its boundary must contain one of the sectors between two curves $\gamma(X)$, $\gamma(Y)$, adjacent to each other in their circular order around A).

Also, the preceding considerations clearly imply that $\xi(A, B)$ belongs to $C(A, B)$ (this is easily seen to be true even in case B is a ridge point relative to A, if we take $\xi(A, B)$ to be any of the edge sequences crossed by shortest paths from A to B).

Our next goal is to show that the output of the procedure EDGE_SEQUENCES does not change as A and B vary slightly, unless one of the following critical conditions occurs:

DEFINITION. A *critical* point $(A, B) \in Z$ is defined as a point in which one of the following critical situations occurs:

(a) Either A or B is a vertex of K or is a ridge point relative to some vertex of K.

(b) Either A lies on $\pi(X, B)$ or B lies on $\pi(X, A)$ for some vertex X of K.

PROPOSITION 2.4. *Suppose $(A, B) \in Z$ is a noncritical point. Then the procedure* EDGE_SEQUENCES *will yield the same output $C(A', B') = C(A, B)$ when applied to any pair (A', B') in a sufficiently small neighborhood of (A, B).*

Proof. First note that the conditions of criticality define a closed subset of Z, so that the set of noncritical points is open. Consider the collection of curves $\gamma(X) = \gamma_{A,B}(X)$, for X a vertex of K. We first claim that as (A', B') varies in a sufficiently small neighborhood of (A, B) which consists only of noncritical points, the curves $\gamma_{A',B'}(X)$ remain simple arcs which vary continuously with (A', B'), their points of intersection vary continuously, and the combinatorial pattern of their intersections (that is, the curves intersecting each $\gamma(X)$ and the order of their intersections along $\gamma(X)$) remain constant.

To see this, note first that as long as neither A' nor B' becomes a ridge point relative to some vertex of K, the paths $\pi(A', X)$, $\pi(B', X)$ vary continuously with A', B', for each vertex X of K. Furthermore, their concatenation is a simple arc unless either A' lies on $\pi(B', X)$ or B' lies on $\pi(A', X)$, a case which does not occur as long as (A', B') remains noncritical. If two arcs $\gamma(X)$ and $\gamma(Y)$ intersect one another, say $\pi(A', X)$ intersects $\pi(B', Y)$, then since (A', B') is assumed to be noncritical it follows that these paths must intersect one another transversally so that their intersection point must also vary continuously with A', B'. Finally, note that, in general, if a collection of simple continuous arcs vary continuously in such a way that these arcs always intersect one another only transversally, then the combinatorial pattern of their intersections can change only when three of these arcs intersect at a common point. This, however, is impossible in our case. Indeed, suppose that $\gamma_{A',B'}(X)$, $\gamma_{A',B'}(Y)$, $\gamma_{A',B'}(Z)$ intersect at a common point, for some three distinct vertices X, Y, Z of K. Then two of these curves must intersect at their portions between their corresponding vertices and the same point A' or B', e.g., $\pi(A', X)$ intersects $\pi(A', Y)$. But this is impossible by Lemma 2.1 and because X and Y are both vertices.

This implies that the combinatorial structure of the connected regions into which S is partitioned by the arcs $\gamma_{A',B'}(X)$ remains constant as (A', B') varies within a sufficiently small neighborhood of (A, B), and that each of these regions varies continuously with (A', B'). Thus the first step of the procedure EDGE_SEQUENCES yields the same output for all (A', B') sufficiently near (A, B).

The second step of that procedure calculates the sequences of edges of K intersecting each half-boundary δ of regions R containing both A and B. The output of this step can in fact change at points A', B' for which two curves $\gamma_{A',B'}(X)$, $\gamma_{A',B'}(Y)$ intersect at a point lying on an edge of K, so that this intersection point can newly appear in, or disappear from such a half-boundary δ. Nevertheless we claim that these potential changes cannot affect the final output of our procedure. For this we first need the following:

Claim. Let $R = R_{A,B}$ be a region in the above partitioning of S, whose boundary δ contains both A and B, so that these points partition δ into two half boundaries $\delta_1 = \delta_1(A, B)$ and $\delta_2 = \delta_2(A, B)$. Then the following properties hold:

(a) An edge e intersects both $\delta_1(A, B)$ and $\delta_2(A, B)$ if and only if it intersects both $\delta_1(A', B')$ and $\delta_2(A', B')$ for all points (A', B') in a sufficiently small neighborhood of (A, B).

(b) For edges e as in (a), there exists a unique subsegment of e whose endpoints $p_1 = p_1(A, B)$, $p_2 = p_2(A, B)$ lie on $\delta_1(A, B)$, $\delta_2(A, B)$, respectively, and its interior is contained in R, if and only if this property holds for all (A', B') sufficiently near (A, B).

(c) For edges e as in (a) and (b), each of the endpoints $p_1(A', B')$ and $p_2(A', B')$ varies continuously with (A', B') near (A, B).

(Intuitively, these claims state that, as (A', B') varies near (A, B), the set of edges crossed by both $\delta_1(A', B')$ and $\delta_2(A', B')$ does not change, the subset of those edges whose intersection with R is connected also does not change, and these connected intersections vary continuously with (A', B').)

Indeed, suppose for the moment that (a), (b), (c) have been established. Then, for each such region R, as A', B' vary slightly near A, B, it is clear that the third step of procedure EDGE_SEQUENCES will produce the same sequence of edges intersecting both δ_1 and δ_2, that if this step does not screen out R at (A, B) it will not do so at nearby points (A', B') and vice versa, and finally that the points $p_1(A', B')$ and $p_2(A', B')$ calculated by this step will vary continuously along $\delta_1(A', B')$ and $\delta_2(A', B')$, respectively, so that their order along these half-boundaries will not change. Hence

the collection of edge sequences output by the procedure will remain constant as (A', B') varies sufficiently near (A, B).

To prove (a), let e be an edge of K which intersects both portions δ_1 and δ_2 of the boundary of R, and let e' be a subsegment of e whose endpoints $p_1(A, B)$, $p_2(A, B)$ are intersections of e with δ_1, δ_2, respectively, and whose interior is fully contained in R. If, say, at $p_1(A, B)$ the edge e meets just one of the curves $\gamma(X)$, then it is clear that as A', B' vary slightly near A, B, the point $p_1(A', B')$ will continue to lie on $\gamma(X)$ and will vary continuously (because the intersection between $\gamma(X)$ and e is transversal). Even if $p_1(A, B)$ lies at an intersection point of two curves $\gamma(X)$, $\gamma(Y)$, it is still easy to see, using the fact that, when properly unfolded, $\gamma(X)$, $\gamma(Y)$ (and e) are all locally straight segments crossing one another transversally near that intersection point, that the point $p_1(A', B')$ will continue to exist, and will vary continuously in this case too (possibly moving from $\gamma(X)$ into $\gamma(Y)$ or vice versa as A', B' vary near A, B). This implies in particular that e will continue to intersect both δ_1 and δ_2, for points A', B' near A, B, and this clearly implies (a).

To show (b), note that the condition stated there can change only when one of the half-boundaries δ_1, δ_2 touches e without crossing it. This, however, is impossible because the intersection of any $\gamma(X)$ with e is always transversal, unless it occurs at an endpoint of e, in which case it cannot cause changes in the condition stated in (b).

Finally, property (c) follows directly from the proof of (a) given above. The proof of the proposition is thus complete. □

Proposition 2.4 implies that if we partition the subset of all noncritical points in Z into connected components, then within each such component the set $C(A, B)$ of the $O(n)$ possible values that $\xi(A, B)$ can assume, as computed by procedure EDGE_SEQUENCES, remains constant. To estimate the number of connected noncritical regions in Z, we show next that the critical points in Z lie on the union of $O(n^2)$ *critical curves*, each of which is either a straight segment or part of an hyperbola. Hence the total number of intersections between these critical curves, and thus the maximal number of noncritical regions into which they decompose Z, is at most $O(n^4)$, so that the maximal number of shortest-path edge sequences induced by points $(A, B) \in e_1 \times e_2$ is at most $O(n^5)$.

To see this, we examine each condition of criticality separately.

Critical points of type (a). A critical point (A, B) of type (a) is one for which either A or B is an endpoint of one of the segments e_1, e_2, or is a ridge point with respect to a vertex of K. As mentioned above, each vertex of K has at most $O(n)$ ridge points on e_1 and on e_2. Hence critical points of type (a) lie on $O(n^2)$ straight segments of the form $A = \text{const}$ or $B = \text{const}$ in the parametric space Z.

Critical points of type (b). Fix some vertex X of K, and partition the edge e_1 into $O(n)$ disjoint subintervals by the ridge points relative to X lying on it. For each such interval I the sequences $\xi(X, A)$, for $A \in I$, are all equal. Suppose e_2 belongs to this common sequence, and unfold the fixed sequence of faces of K crossed by all paths $\pi(X, A)$, $A \in I$, such that all these faces lie in a common plane and such that after this unfolding the paths $\pi(X, A)$ become all straight segments. Then the locus of pairs $(A, B) \in Z$ for which $A \in I$ and B lies on $\pi(X, A)$ is equivalent to the locus of all pairs (A^*, B^*) of points lying respectively in the two coplanar unfolded images of e_1, e_2, such that the line connecting A^* to B^* passes through a fixed point $\mathbf{p}$ (the unfolded image of X). An easy exercise in analytical geometry shows that this locus is a hyperbola, which degenerates into a straight line if the unfolded images of e_1 and e_2 are parallel. (To see this, parametrize the point A^* (resp. B^*) as $\mathbf{a} + \mathbf{b}t$ (resp. $\mathbf{c} + \mathbf{d}s$), for appropriate vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$, and real parameters s, t. Then the condition that A^*, B^*, $\mathbf{p}$ are collinear

can be written as

$$(A^* - \mathbf{p}) \times (B^* - \mathbf{p}) = 0,$$

i.e.,

$$\begin{aligned} 0 &= (\mathbf{a} - \mathbf{p} + \mathbf{b}t) \times (\mathbf{c} - \mathbf{p} + \mathbf{d}s) \\ &= (\mathbf{a} - \mathbf{p}) \times (\mathbf{c} - \mathbf{p}) + [\mathbf{b} \times (\mathbf{c} - \mathbf{p})]t + [(\mathbf{a} - \mathbf{p}) \times \mathbf{d}]s + (\mathbf{b} \times \mathbf{d})ts, \end{aligned}$$

which is clearly an equation of a hyperbola in the $s - t$ space.)

Repeating this argument for each vertex X of K, and also reversing the roles of e_1 and e_2, we conclude that type (b) critical points lie on $O(n^2)$ hyperbolae and straight segments in Z.

This analysis establishes the claim made above.

Finally, note that to estimate the total number of shortest-path edge sequences along S it suffices (by trimming an initial and a final portion of shortest paths if necessary) to consider only the case in which the source and target points of the shortest paths lie on edges of K. Thus by repeating the above analysis for all $O(n^2)$ pairs of edges of K, we obtain the following main theorem of this section:

THEOREM 2.5. *The total number of shortest-path edge sequences along the surface of a* 3-D *convex polyhedron with n vertices is at most* $O(n^7)$.

Remark. We do not know whether this bound is tight. However, we are grateful to David Mount, who has provided us with the following example, showing that the total number of shortest-path edge sequences along S is $\Omega(n^4)$. This still leaves a gap between this lower bound and the above upper bound, and our intuition leads us to conjecture that our upper bound can still be substantially improved. In either case, the problem of obtaining a tight bound for the number of shortest-path edge sequences along the surface of a convex polyhedron with n vertices is still open.

In Mount's example, the surface S of the polyhedron K contains three groups G_1, G_2, G_3 of n edges each, such that the edges within each group are parallel to each other, and collectively bound $n - 1$ thin and long rectangular faces between them, such that the dihedral angle between any pair of adjacent faces is sufficiently small. The edges in G_1, G_3 are all parallel to the x-axis, and the edges in G_2 are all parallel to the y-axis. The faces bounded by the edges in G_1 (resp. G_3) are separated from the faces bounded by the edges in G_2 by a thin n-sided face f_{12} (resp. f_{32}). See Fig. 4 for an illustration of K.

It can be checked that, by appropriate choice of the parameters governing the shape of K, one has the following property: For each pair of edges $e \in G_1$, $e' \in G_3$ there exist $\Omega(n^2)$ pairs of edges of G_2 such that for each of these pairs e_i, e_j there

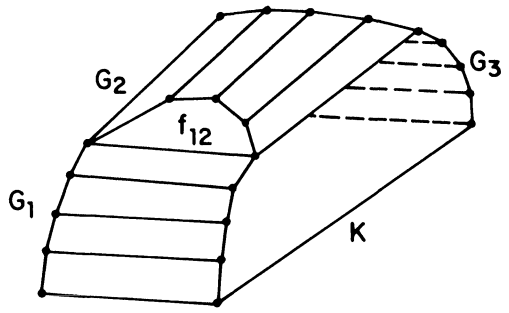


FIG. 4

exist points $x \in e$, $y \in e'$ such that the edges of G_2 crossed by $\pi(x, y)$ are precisely those lying between e_i and e_j. Thus there exist at least $\Omega(n^4)$ shortest-path edge sequences along S. (This example is weak in the sense that all the shortest-path edge sequences under consideration are contiguous subsequences of just $O(n^2)$ distinct edge sequences. Thus there still remains the problem whether there exist polyhedra having more than quadratically many "incomparable" shortest-path edge sequences, i.e., sequences none of which is a contiguous subsequence of the other.)

The above analysis also makes it easy to calculate all shortest-path edge sequences along S, in the following crude and straightforward manner:

For each pair e_1, e_2 of edges of K do the following:

(a) Draw all $O(n^2)$ critical lines and hyperbolae in $Z = e_1 \times e_2$ to obtain a decomposition of Z into $O(n^4)$ connected noncritical regions.

(b) For each such region R, choose an arbitrary pair $(A, B) \in R$, and apply the procedure EDGE_SEQUENCES to (A, B), obtaining a collection $C(R) = C(A, B)$ of $O(n)$ edge sequences.

Output the union of all collections obtained in step (b).

The running time of this procedure is clearly polynomial in n. Specifically, step (a) can be accomplished, for each pair of edges of K, by calculating all the intersection points between the critical curves, by sorting these points along each of these curves, and then by tracing the boundary of each noncritical region, in overall time $O(n^4 \log n)$.

In implementing step (b), we need to execute the procedure EDGE_SEQUENCES for (a representative pair (A, B) in) each of the resulting $O(n^6)$ noncritical regions. This execution can be carried out as follows. The curves $\gamma(X)$ can be calculated in time $O(n^2 \log n)$ using Mount's algorithm (for the two source points A, B). To obtain the decomposition of S into connected regions by these curves, we need to find their points of intersection. For this, we note that within each face f of K each of these $O(n)$ curves consists of just one segment (or at most two if f contains A or B); thus their points of intersection can be found by a standard sweeping technique in time $O((n + t_f) \log n)$, where t_f is the number of intersections of these curves along f. Since by Lemma 2.3 one has $\sum_f t_f = O(n^2)$, it follows that all the required intersections can be found in overall time $O(n^2 \log n)$. The required decomposition of S can then be found by sorting these intersection points along each of the curves $\gamma(X)$, and then by tracing the boundary of each region in this decomposition, again in total time $O(n^2 \log n)$. As this boundary tracing is performed, it is easy to collect on the fly all the intersections of region boundaries with edges of K, and process them so as to accomplish steps (II) and (III) of the procedure also in time $O(n^2 \log n)$.

Hence the above procedure for calculating all possible shortest-path edge sequences along S can be carried out in overall $O(n^8 \log n)$ time.

3. Shortest paths amidst a fixed number of convex polyhedral obstacles. As outlined in the introduction, Theorem 2.5 can be applied to obtain algorithms for the calculation of shortest paths amidst a fixed number k of convex polyhedral obstacles having altogether n vertices, which run in time polynomial in n (albeit exponential in k).

Let $A_1, \cdots, A_k$ be k disjoint convex polyhedra in 3-space, and let X and Y be two points in free space (i.e., in the complement of the union of these polyhedra). The problem is to calculate the shortest path π from X to Y which does not penetrate into the interiors of $A_1, \cdots, A_k$. As shown in [SS], π is a polygonal path whose corners lie on edges of the polyhedra A_i, such that with each edge it crosses it subtends equal incoming and outgoing angles. Furthermore, the fact that each A_i is convex implies that if W, Z are two points on π which lie on the surface of the same polyhedron A_i

then the entire subpath of π between W and Z must lie along that surface. Also, each connected portion of π in which it does not touch any polyhedron must be a straight segment connecting a "take-off" point on the surface of one polyhedron to a "landing" point on the surface of another (with X as an initial take-off point and Y as a final landing point). It is also easy to show that each of those take-off and landing points lie on a polyhedron edge (unless it coincides with one of the points X, Y). See Fig. 5 for an illustration of this structure.

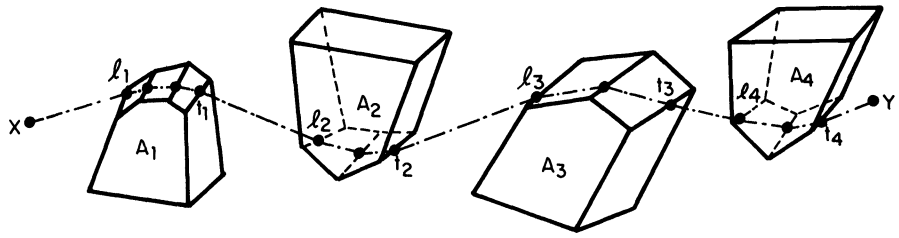


FIG. 5

Summing up these observations, there exist a subsequence $A_{i_1}, A_{i_2}, \cdots, A_{i_p}$ of the given polyhedra, with each polyhedron A_q appearing at most once in this sequence, and a corresponding sequence of take-off and landing points $l_1, t_1, l_2, t_2, \cdots, l_p, t_p$ with l_q, t_q lying on the surface of A_{i_q} for $q = 1, \cdots, p$, such that

$$\pi = Xl_1 \| \pi_1 \| t_1 l_2 \| \pi_2 \| t_2 l_3 \| \cdots \| t_{p-1} l_p \| \pi_p \| t_p Y$$

where each π_q is a shortest path along the surface of A_{i_q} between l_q and t_q, and where the interior of each of the straight segments $t_q l_{q+1}$ does not intersect any of the polyhedra.

Hence, to find the desired path π, one needs to solve two subproblems:

(I) Determine the sequence $A_{i_1}, A_{i_2}, \cdots, A_{i_p}$ of polyhedra through which π passes.

(II) Given that sequence, calculate the take-off and landing points of π on those polyhedra.

As for the first subproblem, we do not propose any efficient procedure, and make do with enumeration of all $O(k^k)$ possible subsequences of polyhedra, choosing the one yielding the shortest of all the paths calculated for each of these sequences. (We note that this subproblem is at least as difficult as the general problem of calculating shortest paths amidst arbitrary polyhedral obstacles, for which only exponential-time algorithms are known as yet [SS], [RS].)

Suppose then that we are given a specific sequence of polyhedra, which, without loss of generality, we will take to be the sequence $A_1, A_2, \cdots, A_k$ itself, and consider the second subproblem of determining the take-off and landing points on the shortest path from X to Y which is constrained to pass through the surfaces of $A_1, \cdots, A_k$ in that order. Write π as above as $Xl_1 \| \pi_1 \| t_1 l_2 \| \pi_2 \| \cdots \| t_{k-1} l_k \| \pi_k \| t_k Y$, where each π_i is a shortest path along the surface of A_i from l_i to t_i. Note that by Theorem 2.5, for each $i = 1, \cdots, k$, the total number of shortest-path edge sequences through which π_i can pass is at most $O(n^7)$, where n is the maximal number of vertices on each of the polyhedra A_i. Hence altogether there are at most $O(n^{7k})$ possible sequences of edges through which π can pass.

Let ξ be one of these candidate sequences of edges. We first unfold, for each $i = 1, \cdots, k$, all faces of A_i bounded by the edges in ξ which lie on A_i, so as to make

all these faces co-planar, and to make π_i a straight segment. These unfoldings then reduce the specific subproblem at hand to that of calculating the shortest path between two given points which is constrained to pass through $2k$ lines (namely the lines containing the edges in ξ on which the take-off and landing points lie). Note however that these unfoldings may introduce irrational terms into the coefficients representing these edges. Thus we assume a model of computation in which each infinite-precision real arithmetic operation (such as rational operations, square root extractions, and comparisons) requires constant time. Then we can use the technique of [RS] to solve subproblem II in time singly exponential in k.

Finally, note that for each candiate path π we also need to check that (i) the contact points of π with the lines containing the polyhedra edges through which π is constrained to pass, actually fall inside the corresponding edges, and (ii) π does not intersect any polyhedron other than those through which it is assumed to pass a priori. Only paths π which meet this criterion are further considered. These additional tests, however, are easily accomplished in polynomial time.

We thus obtain the following main theorem of this section:

THEOREM 3.1. *The shortest path between two points X, Y which does not penetrate into the interiors of k disjoint convex polyhedra having altogether n vertices, can be calculated in time $n^{O(k)}$.*

Remarks. (1) In [RS] the authors claim that in case of k disjoint convex obstacles, their algorithm will run in time $O(n^{k^{O(1)}})$. This is true in the infinite-precision real arithmetic model assumed above, but is false in the purely rational arithmetic model, because it relies on a polynomial time solution for the case of a single convex polyhedral obstacle. However the known polynomial-time algorithms for this latter problem all use also irrational operations, such as square root extractions (to perform planar unfoldings). Thus in the purely rational model the technique presented here and the technique of [RS] will both run in time singly exponential in n. However, in the infinite-precision real model, both procedures will run in time polynomial in n, but the exponent for our algorithm is considerably smaller than that obtained in [RS].

(2) The result stated in Theorem 3.1 is related to the solution of the 2-D "asteroid avoidance problem" studied in [RSh]. Theorem 3.1 is stronger, however, because it makes use of the rather heavy Theorem 2.5. In contrast, the analysis in [RSh] deals with only trivial sets of edge sequences, whose calculation is also straightforward.

REFERENCES

[BS] A. BALTSAN AND M. SHARIR, *On shortest paths between two convex polyhedra*, Tech. Rep. 180, Computer Science Dept., Courant Institute, New York, September 1985.

[Mo] D. M. MOUNT, *On finding shortest paths on convex polyhedra*, Tech. Rep., Computer Science Dept., Univ. of Maryland, College Park, MD, October 1984.

[RSh] J. H. REIF AND M. SHARIR, *Motion planning in the presence of moving obstacles*, Proc. 26th IEEE Symposium on the Foundations of Computer Science, 1985, pp. 144–154.

[RS] J. H. REIF AND J. A. STORER, *Shortest paths in Euclidean space with polyhedral obstacles*, Tech. Rep. CS-85-121, Computer Science Dept., Brandeis University, Waltham, MA, April 1985.

[SS] M. SHARIR AND A. SCHORR, *On shortest paths in polyhedral spaces*, this Journal, 15 (1986), pp. 193–215.

SIAM J. COMPUT.
Vol. 16, No. 3 June 1987

© 1987 Society for Industrial and Applied Mathematics
010

BINARY SEARCH ON A TAPE*

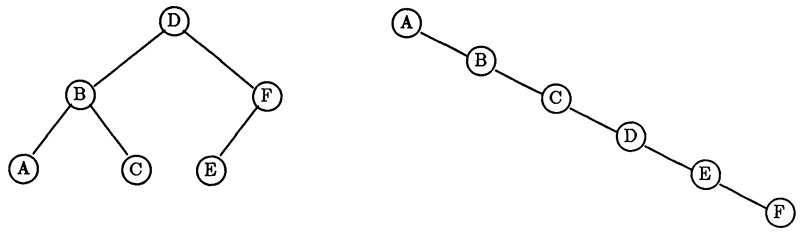
T. C. HU† AND MICHELLE L. WACHS‡

Abstract. Given n records stored alphabetically on a tape, any comparison search procedure can be characterized by a binary tree. The complete binary tree (binary search) uses the minimum number of comparisons but not the minimum number of movements. The linear binary tree (sequential search) uses the minimum number of movements but not the minimum number of comparisons. A tape-optimal tree is a tree which minimizes the total cost of comparisons and movements. The tape-optimal tree is a "hybrid" of the linear tree and the complete binary tree, and is characterized for arbitrary n.

Key words. optimal file search procedure, sequential access file, binary search, sequential search, binary tree

AMS(MOS) subject classifications. 05C05, 68P05, 68P10, 68Q20, 68Q25

1. Introduction. Consider a sorted file of n records. It is well known that any comparison search procedure can be represented by a binary tree of n nodes (binary search tree). The ith node in the symmetric order (inorder) of the nodes of the tree corresponds to the ith record of the file. The root of the tree corresponds to the first record to be examined in the search strategy. The left and right subtrees of the root correspond to a splitting of the file into two subfiles, one of which is to be searched if the record we are trying to locate does not correspond to the root. To locate a particular record X, X is compared with the root of the tree. If X is less than the root then the search continues with the left subtree; if X is the root then the search terminates successfully; otherwise the search continues with the right subtree. The traditional binary search is represented by the almost complete binary tree (every level except possibly the last has the maximum number of nodes) and sequential search is represented by the linear binary tree. For example, if A, B, C, D, E, F is the sorted file of records then Fig. 1.1(a) represents the traditional binary search and Fig. 1.1(b) represents the sequential search.



(a) (b)

FIG 1.1

* Received by the editors August 7, 1984; accepted for publication (in revised form) July 14, 1986.

† Department of Electrical Engineering and Computer Sciences, University of California, San Diego, La Jolla, California 92093. The research of this author was supported in part by the National Science Foundation under grant MCS80-03362.

‡ Department of Mathematics and Computer Science, University of Miami, Coral Gables, Florida 33124. This research was done while the author was visiting the Department of Electrical Engineering and Computer Sciences at the University of California, San Diego. The research of this author was supported in part by National Science Foundation grant MCS 81-03474.

If the search is successful then the number of comparisons needed to locate a given record is equal to the level of the corresponding node in the tree (we use the convention that the root of the tree is at level one). Thus we need 3 comparisons to locate record A in the search procedure of Fig. 1.1(a) and 1 comparison in the search procedure of Fig. 1.1(b).

Now suppose that a sorted file is stored on a tape, or to discretize the situation, the file is stored in a doubly linked linear list. In addition to the number of comparisons needed to locate a given record, we shall be concerned with the number of movements needed. By a movement we mean a forward or reverse pass from one record to an adjacent record. For example, suppose we are using the search procedure of Fig. 1.1(a) to locate record C. It takes 4 forward movements to reach record D where the first comparison is made. Then 2 reverse movements bring us to record B, where the second comparison is made. Finally 1 forward movement brings us to record C, where the third comparison is made. Thus the total number of movements needed to locate record C is $4+2+1=7$. However, in the search procedure of Fig. 1.1(b) only 3 forward movements are needed to locate record C.

It is well known that an important component of a sequential access file is an index for the key being searched. Efficient indexing techniques have been extensively studied in the literature (see e.g. [3] or [5]). Although in practice one would use an index rather than perform a binary search directly on a tape, we consider here the situation in which an index is not available. A natural question for this situation is to find the binary tree (the search procedure) which minimizes the average number of comparisons and movements needed to search for a record. This question was raised by A. J. Hu [1] who considers a restricted form of this problem [2]. The binary tree which minimizes the average number of comparisons and movements needed to successfully search for a record in a sequential access file will be called a tape-optimal tree. It is easy to see that the almost complete binary tree is optimal when only comparisons are counted and that the linear tree is optimal when only movements are counted. Hence the tape-optimal tree is in a sense a "hybrid" of the linear tree and the almost complete binary tree.

For $n=46$, a tape-optimal tree is shown in Fig. 1.2. This indicates that in an optimal search strategy for a file with 46 records, the first comparison is made at the 6th record of the file and the file is split into subfiles of sizes 5 and 40. If we happen to be searching for the 13th record of the file then the tape head moves, in succession,

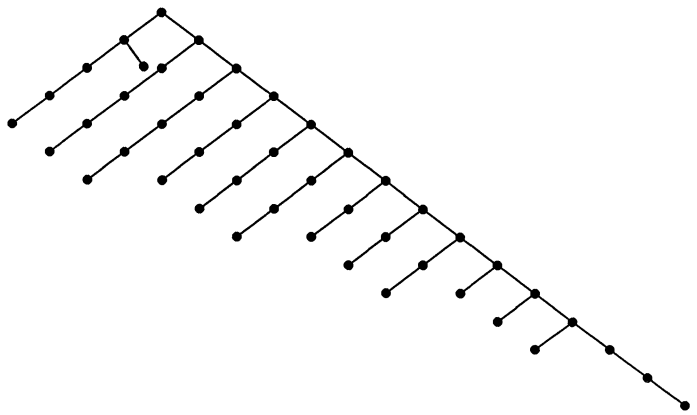


FIG. 1.2

to the 6th, 11th, 16th, 15th, 14th and 13th records, where the comparisons are made. This requires 16+3 movements and 6 comparisons.

In this paper, we characterize the tape-optimal trees by giving explicit expressions for the sizes of the subtrees that hang from the right-most path of the tree. That is, for each n, the number of nodes in the tree, we will determine the m_i in Fig. 1.3. In §§ 2-5 we establish our characterization of tape-optimal trees. To do this, we study the following interesting recurrence relation which is derived in § 2:

$$T(n)=\begin{cases}\min\limits_{0\leqq m<n}\left((m+2)n+T(m)+T(n-m-1)\right) & \text{if } n>0,\\ 0 & \text{if } n=0.\end{cases}$$

In § 6 we consider the more general situation in which comparisons and movements have different "costs." The optimal trees obtained reduce to the almost complete binary trees when the "cost" of movements is 0. Hence § 6 gives a generalization of traditional binary search.

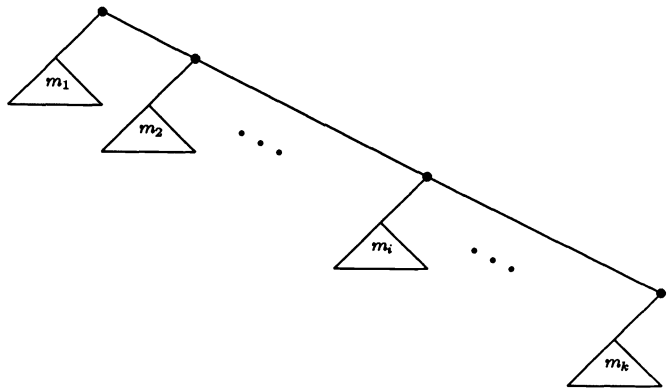


FIG. 1.3

In the remainder of this section, we shall describe our characterization of the tape-optimal trees. The following notation will be used. Let A and B be binary trees.

(1) $A \wedge B$ denotes the tree whose left subtree is A and whose right subtree is B.

(2) $|A|$ denotes the size of A, i.e., the number of nodes of A.

(3) $\bar{A}$ denotes the binary tree obtained from A by interchanging the left and right subtree of each node of A, i.e., $\bar{A}$ is the mirror image of A. Note that $\overline{A \wedge B}=\bar{B} \wedge \bar{A}$.

(4) Δ denotes the empty tree.

Let $A=A_1 \wedge (A_2 \wedge (A_3 \wedge \cdots (A_k \wedge \Delta)\cdots))$ and $|A_i|=m_i$ (as in Fig. 1.3). It is easy to show that if A is tape-optimal then

(1) $m_1 \geqq m_2 \geqq \cdots \geqq m_k$ (Lemma 2.1) and

(2) $\bar{A}_i$ is tape-optimal for all $i=1, 2, \cdots, k$.

We shall call binary trees which satisfy (1) and (2), *trimmed trees* of type $(a_m, a_{m-1}, \cdots, a_0)$, where a_j is the number of times j appears in the sequence $m_1, m_2, \cdots, m_k$. For example, the trimmed tree in Fig. 1.4 has type (1, 0, 2, 0, 1, 3).

A fundamental result which follows from results in § 2 is that tape-optimal trees can be "grown" in a greedy fashion.

THEOREM 1.1. *Let A be a tape-optimal tree with n nodes. Then a tape-optimal tree with $n+1$ nodes can be obtained by attaching a leaf to A. Furthermore, if $n>0$ then a tape-optimal tree with $n-1$ nodes can be obtained by removing a leaf from A.*

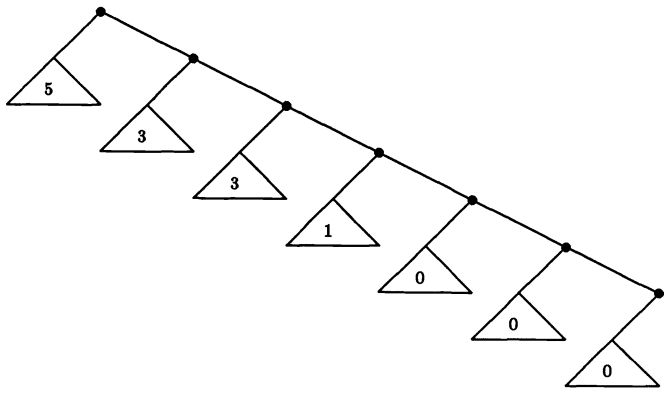


FIG. 1.4

The tape-optimal trees will be obtained by growing leaves on the trees which belong to a special class of tape-optimal trees. The situation is analogous to the case of traditional optimal binary search trees (i.e., only comparisons count). The following trivial statements about binary search trees have tape-analogues.

1) There is a unique optimal binary search tree with n nodes if and only if $n = 2^k - 1$ for some $k \geqq 0$.

2) The unique optimal binary search trees are the complete binary trees.

3) Any other optimal binary search tree (almost complete binary tree) can be obtained from a complete binary tree by "growing" leaves on the next level.

4) If all the leaves on the next level are grown then we get the next complete binary tree.

To present the tape analogue of these statements, we need to define a sequence of numbers. Let $w = w_0 w_1 w_2 \cdots \in \{2, 3\}^\infty$ be the infinite word defined recursively by

$$w = 3 \prod_{i=0}^{\infty} (2^i 3)^{w_i},$$

where the relevant operation is concatenation (and $w_i \in \{2, 3\}$). Note that $w_0 = 3$. An initial segment of w (with blanks inserted to identify the factors $(2^i 3)^{w_i}$) is

3 333 232323 223223223 222322232223 2222322223 222223222223222223.

The analogue of the complete binary tree can be characterized as follows: A trimmed tree of type $(a_m, a_{m-1}, \cdots, a_0)$ is said to be tape-complete if $a_i = w_i$, $i = 0, 1, \cdots, m-1$ and $0 \leqq a_m < w_m$. An example of a tape-complete tree for $m = 5$ is given in Fig. 1.2.

Tape analogue of 1. There is a "unique" tape-optimal tree with n nodes if and only if

$$n = \sum_{i=0}^{m-1} (i+1) w_i + (m+1) d$$

for some m and d such that $0 \leqq d < w_m$.

Tape analogue of 2. The "unique" tape-optimal trees are the tape-complete trees.

Note. By "unique" we mean that the type $(a_m, a_{m-1}, \cdots, a_0)$ is unique. The subtrees along the rightmost path will not necessarily be unique.

We will use a leaf growing operator which is applied to tape-complete trees. This operator allows certain subtrees that hang from the rightmost path of a tape-complete

tree of type $(d, w_{m-1}, w_{m-2}, \cdots, w_0)$ to grow *one* leaf. The subtrees that are allowed to grow a leaf are the highest (leftmost) subtrees of each size, $m-1, m-2, \cdots, 0, -1$ (by subtree of size -1 we mean the null right subtree of the rightmost node of the tree, as indicated in the lowest circle of Fig. 1.5(a)). Note that the tree will remain trimmed if any of the allowable trees grow a leaf. For example, the tree in Fig. 1.5(a) is tape-complete. The circled subtrees are the trees that are allowed to grow one new leaf. The starred subtrees form an arbitrary subset of the circled subtrees. To obtain the tape-optimal tree in Fig. 1.5(b), we add one new leaf to each of the starred subtrees. The leaf growing operator will be precisely defined in § 3.

Tape analogue of 3. Any tape-optimal tree can be obtained by "growing" one leaf on each subtree of a subset of the allowable subtrees that hang from the rightmost path of a tape-complete tree.

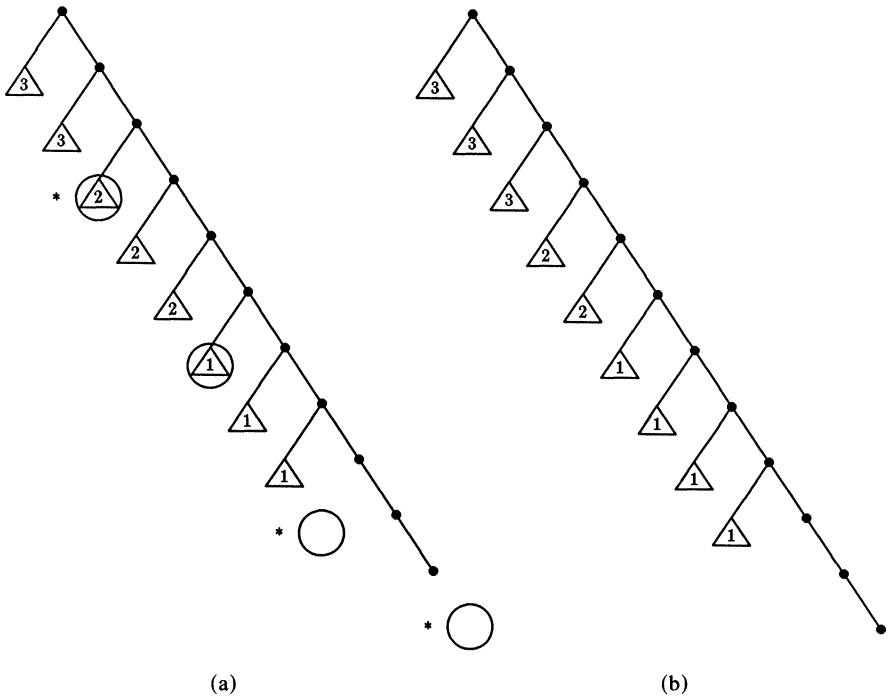
Tape analogue of 4. If all the allowable subtrees grow a leaf then we get the next tape-complete tree.

The tape analogues follow from the results presented in the next four sections.

2. Preliminaries. For a binary tree A, let $M(A)$ denote the total number of movements needed to search for every node of A, let $C(A)$ denote the total number of comparisons, and let $S(A)$ denote the sum of these, i.e.,

$$S(A) = M(A) + C(A). \tag{2.0}$$

Suppose A has $n > 0$ nodes and $A = A_1 \wedge A_2$. Then in each search of a node of A, there are initially $|A_1| + 1$ forward movements to reach the root of A where the first comparison is made. This contributes $(|A_1| + 1) \cdot n$ movements to the total number of



(a) (b)

FIG. 1.5

movements needed to search for every node of A. Suppose we are searching for a node of A_1. After the root of A is reached, the search continues from the right end, rather than the left end, of the subfile corresponding to A_1. Hence we can think of this as searching the reverse of the subfile. Since $\bar{A}_1$ gives the search strategy for the reverse of the subfile, the number of additional movements needed to search for all the nodes of A_1 is $M(\bar{A}_1)$. We also have that the number of additional movements needed to search for all the nodes of A_2 is $M(A_2)$. It follows that

$$M(A) = (|A_1|+1)\cdot n + M(\bar{A}_1) + M(A_2).$$

We can also see that

$$C(A) = 1\cdot n + C(\bar{A}_1) + C(A_2).$$

Combining these recurrence relations for M and C we get the following recurrence relation for S:

$$S(A) = \begin{cases} (|A_1|+2)\cdot n + S(\bar{A}_1) + S(A_2) & \text{if } |A| = n > 0, \\ 0 & \text{if } |A| = 0. \end{cases} \tag{2.1}$$

To simplify our presentation we modify the cost function S by removing the bar above A_1 in (2.1). This gives us a new cost function T defined by the recurrence relation:

$$T(A) = \begin{cases} (|A_1|+2)\cdot n + T(A_1) + T(A_2) & \text{if } |A| = n > 0, \\ 0 & \text{if } |A| = 0. \end{cases} \tag{2.2}$$

Note that the advantage of the new cost function T is that subtrees of optimal trees inherit optimality; while under the original cost function S, the right subtree of an optimal tree is optimal but the left subtree must be "flipped" to get an optimal subtree. There is a bijection between the optimal trees for S and the optimal trees for T, which preserves the sizes of the left and right subtrees as well as the cost of an optimal tree. Let $\mathcal{B}_S$ and $\mathcal{B}_T$ denote the sets of optimal binary trees for the cost functions S and T, respectively. Then the function $\varphi\colon \mathcal{B}_S \to \mathcal{B}_T$ defined recursively by

$$\varphi(A_1 \wedge A_2) = \varphi(\bar{A}_1) \wedge \varphi(A_2), \qquad \varphi(\Delta) = \Delta,$$

is the bijection. It is easy to verify inductively that φ is indeed a bijection which preserves the sizes of the left and right subtrees and that $S(A) = T(\varphi(A))$ for all $A \in \mathcal{B}_S$. Since the right subtree of an optimal subtree for either cost function is optimal, it also follows inductively that φ will preserve the sizes of the subtrees that hang from the rightmost path of an optimal tree. Since the optimal trees will be characterized in terms of these sizes, this characterization will be valid for both cost functions.

The terms defined in § 1 will now be redefined for the new cost function T. We say that a binary tree A is *tape-optimal* if $T(A) \leqq T(A')$ for all A' such that $|A| = |A'|$. The *rightmost tape-optimal* tree with n nodes is a tape-optimal tree A such that if $A = A_1 \wedge A_2$ then

(1) A_1 and A_2 are rightmost tape-optimal and

(2) if $B_1 \wedge B_2$ is tape-optimal with n nodes then $|A_2| \geqq |B_2|$.

We will use the symbol $[n]$ to denote the rightmost tape-optimal tree with n nodes. The tree $[m_1] \wedge ([m_2] \wedge ([m_3] \wedge \cdots ([m_k] \wedge [0]) \cdots))$ will be denoted by $\langle m_1, m_2, \cdots, m_k \rangle$.

Let $T(n)$ be the cost of an optimal tree with n nodes, i.e. $T(n) = T([n])$. Then by (2.2),

$$T(n) = \min_{0 \leqq m < n} ((m+2)n + T(m) + T(n-m-1)) \quad \text{for } n > 0,$$

$$T(0) = 0.$$

The function D defined as follows will play a fundamental role in our proofs.

$$D(n) = T(n) - T(n-1) \quad \text{for } n \geqq 1,$$

$$D(0) = 0.$$

We begin by showing that if $\langle m_1, m_2, \cdots, m_k \rangle$ is tape-optimal then the m_i are decreasing.

LEMMA 2.1. *$T(\langle m_1, m_2, \cdots, m_k \rangle) < T(\langle m_1, m_2, \cdots, m_{i+1}, m_i, \cdots, m_k \rangle)$ if and only if $m_i > m_{i+1}$.*

Proof. A simple computation following from (2.2) results in

$$T(\langle m_1, m_2, \cdots, m_k \rangle) - T(\langle m_1, m_2, \cdots, m_{i+1}, m_i, \cdots, m_k \rangle) = m_{i+1} - m_i. \qquad \square$$

A tree, $\langle m_1, m_2, \cdots, m_k \rangle$ for which $m_1 \geqq m_2 \geqq \cdots \geqq m_k$ will be called a *trimmed tree.* We can use a shorter notation to denote trimmed trees. For $m \geqq 0$ and $a_i \geqq 0$, $i = 0, 1, \cdots, m$, let $(a_m, a_{m-1}, \cdots, a_0)$ denote the trimmed tree, $\langle m, m, \cdots, m, m-1, \cdots, m-1, \cdots, 0, 0, \cdots, 0 \rangle$ where each i appears a_i times. If $a_i > 0$ for all $i = 0, 1, \cdots, m$ then we say that $(a_m, a_{m-1}, \cdots, a_0)$ is a *proper trimmed tree.*

We have the following equivalence relation on trees. Let $A = A_1 \wedge (A_2 \wedge (\cdots \wedge (A_k \wedge \Delta) \cdots))$ and $B = B_1 \wedge (B_2 \wedge (\cdots \wedge (B_k \wedge \Delta) \cdots))$ (see Fig. 2.1). Then A is said to be *equivalent* to B, denoted $A \sim B$, if $|A_i| = |B_i|$ and $T(A_i) = T(B_i)$ for $i = 1, 2, \cdots, k$. Note that if A and B are equivalent then $T(A) = T(B)$.

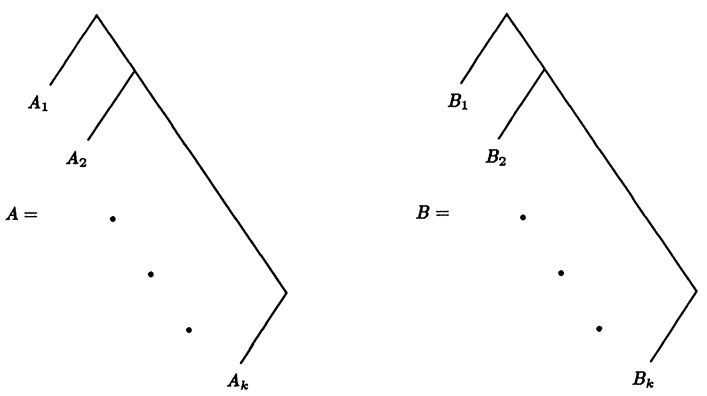


FIG. 2.1

From Lemma 2.1 we have the following.

PROPOSITION 2.2. *Every tape-optimal tree is equivalent to a unique trimmed tree, $(a_m, a_{m-1}, \cdots, a_0)$.* $\square$

It will turn out that tape-optimal trees are actually equivalent to *proper* trimmed trees.

The next two lemmas are fundamental.

LEMMA 2.3. *Let $[m] \wedge [n-m-1]$ be a tape-optimal tree with n nodes. Then*

(a) *Either $[m+1] \wedge [n-m-1]$ or $[m] \wedge [n-m]$ is tape-optimal with $n+1$ nodes and*

(b) *Either* $[m-1]\wedge[n-m-1]$ *or* $[m]\wedge[n-m-2]$ *is tape-optimal with* $n-1$ *nodes, if* $n>1$.

LEMMA 2.4. *For all* $n\geqq 0$, $D(n+1)-D(n)\geqq 1$.

These two lemmas are essentially proved simultaneously by induction. The following sublemma serves to separate the two induction proofs. First the sublemma is used in the proof of Lemma 2.4. Then Lemma 2.4 and the sublemma are combined to give Lemma 2.3.

SUBLEMMA. *Let* n *be such that* $D(k+1)-D(k)\geqq 1$ *for all* $k<n$. *Let* $[m]\wedge[n-m-1]$ *be a tape-optimal tree with* n *nodes. Then* (a) *and* (b) *of Lemma* 2.3 *hold.*

Proof. We prove (a). (The proof of (b) is similar and is omitted.) Suppose r is such that $m<r\leqq n-1$. The trees, $[m]\wedge[n-m-1]$ and $[r]\wedge[n-r-1]$ both have n nodes, and the trees $[m+1]\wedge[n-m-1]$ and $[r+1]\wedge[n-r-1]$ both have $n+1$ nodes. Since $[m]\wedge[n-m-1]$ is tape-optimal we have

$$T(m, n-m-1)\leqq T(r, n-r-1) \tag{2.3}$$

where $T(i,j)$ denotes $T([i]\wedge[j])$. We shall show that

$$T(m+1, n-m-1)\leqq T(r+1, n-r-1). \tag{2.4}$$

A straightforward computation using (2.2) gives,

$$\begin{aligned} &T(r+1, n-r-1)-T(m+1, n-m-1)-(T(r, n-r-1)-T(m, n-m-1)) \\ &\quad = D(r+1)-D(m+1)+r-m. \end{aligned} \tag{2.5}$$

We clearly have,

$$\begin{aligned} D(r+1)-D(m+1) &= \sum_{k=m+1}^{r}(D(k+1)-D(k)) \\ &\geqq \sum_{k=m+1}^{r} 1 \\ &= r-m. \end{aligned}$$

Substituting this into (2.5), we have

$$\begin{aligned} &T(r+1, n-r-1)-T(m+1, n-m-1)-(T(r, n-r-1)-T(m, n-m-1)) \\ &\quad \geqq 2(r-m) \\ &\quad \geqq 0. \end{aligned}$$

Combining this with (2.3) we get (2.4).

Now suppose r is such that $0\leqq r<m$. Inequality (2.3) remains true. We now consider the trees $|m|\wedge[n-m]$ and $[r]\wedge[n-r]$ each having $n+1$ nodes. We shall show that

$$T(m, n-m)\leqq T(r, n-r). \tag{2.6}$$

A straightforward computation gives

$$\begin{aligned} &T(r, n-r)-T(m, n-m)-(T(r, n-r-1)-T(m, n-m-1)) \\ &\quad = D(n-r)-D(n-m)+r-m. \end{aligned}$$

Since $n-m<n-r\leqq n$, we can replace $D(n-r)-D(n-m)$ by a telescoping sum to get

$$\begin{aligned} &T(r, n-r)-T(m, n-m)-(T(r, n-r-1)-T(m, n-m-1)) \\ &\quad\geqq n-r-(n-m)+r-m \\ &\quad=0. \end{aligned}$$

Combining this with (2.3) we get (2.6).

It follows from (2.4) and (2.6) that no tree with $n+1$ nodes has cost less than both $[m+1]\wedge[n-m-1]$ and $[m]\wedge[n-m]$. Hence, either $[m+1]\wedge[n-m-1]$ or $[m]\wedge[n-m]$ is tape-optimal. □

Proof of Lemma 2.4. We use induction on n. For $n=0, 1$, we have $D(n+1)-D(n)=2\geqq 1$. Assume $n>1$ and $D(k+1)-D(k)\geqq 1$ for all $k<n$. Suppose $[m]\wedge[n-m-2]$ is tape-optimal with $n-1$ nodes. By the Sublemma, there is a tape-optimal tree with n nodes and a tape-optimal tree with $n+1$ nodes among the trees given in Fig. 2.2. It follows that there are three cases.

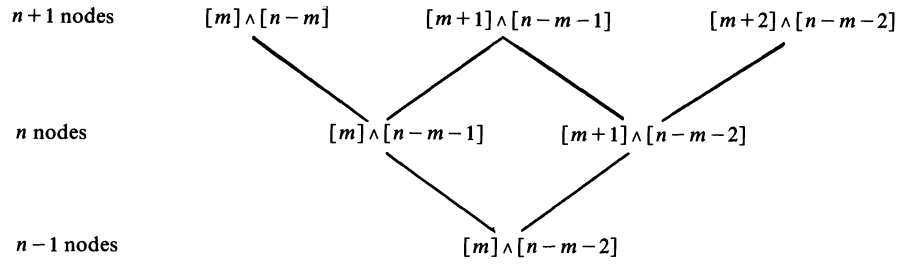


FIG. 2.2

CASE 1. Suppose $T(n+1)=T(m, n-m)$. Then since $T(n)\leqq T(m, n-m-1)$, we have

$$\begin{aligned} D(n+1)-D(n) &\geqq T(m, n-m)+T(m, n-m-2)-2T(m, n-m-1) \\ &=D(n-m)-D(n-m-1). \end{aligned}$$

Since $n-m\leqq n$, the result follows by induction.

CASE 2. Suppose $T(n+1)=T(m+2, n-m-2)$. Then since $T(n)\leqq T(m+1, n-m-2)$, we have

$$\begin{aligned} D(n+1)-D(n) &\geqq T(m+2, n-m-2)+T(m, n-m-2)-2T(m+1, n-m-2) \\ &=D(m+2)-D(m+1)+2. \end{aligned}$$

Since $m+2\leqq n$, the result follows by induction.

CASE 3. Suppose $T(n+1)=T(m+1, n-m-1)$. Then since $2T(n)\leqq T(m, n-m-1)+T(m+1, n-m-2)$, we have

$$\begin{aligned} D(n+1)-D(n) &\geqq T(m+1, n-m-1)+T(m, n-m-2) \\ &\quad -T(m, n-m-1)-T(m+1, n-m-2) \\ &=1. \end{aligned}$$

□

3. Tape-complete trees. In this section we begin our description of the a's of Proposition 2.2. First we need the tape analogue of complete binary trees and an operator for "growing" leaves on trees.

DEFINITION 3.1. A trimmed tree, $(a_m, a_{m-1}, \cdots, a_0)$ is said to be *m-tape-complete* if $a_i=D(i+1)-D(i)+1$ for $i=0, 1, \cdots, m-1$ and $0\leqq a_m<D(m+1)-D(m)+1$.

Remark. It will be shown in Theorem 5.1 that the definition of m-tape-complete given here coincides with the definition given in the introduction.

DEFINITION 3.2. Let $A=(a_m, a_{m-1}, \cdots, a_0)$ be a sequence of integers. Define the growing operator G_i to be

$$G_i(A)=\begin{cases}(a_m, a_{m-1}, \cdots, a_i+1, a_{i-1}-1, \cdots, a_0) & \text{for } i=1, 2, \cdots, m,\\ (a_m, a_{m-1}, \cdots, a_0+1) & \text{for } i=0,\\ (1, a_m-1, a_{m-1}, \cdots, a_0) & \text{for } i=m+1.\end{cases}$$

Since G_i and G_j commute for all $i, j \leqq m+1$, we can generalize G_i as follows.

DEFINITION 3.3. Let $A=(a_m, a_{m-1}, \cdots, a_0)$ be a sequence of integers and let $S=\{i_1, i_2, \cdots, i_j\} \subseteq \{0, 1, \cdots, m+1\}$. Define the growing operator G_S to be

$$G_S(A)=G_{i_1}G_{i_2}\cdots G_{i_j}(A).$$

Remark. We are really interested here in applying G_S to trimmed trees. Since trimmed trees are represented by sequences of integers, we can easily do this. An example is given in Fig. 3.1. For a proper trimmed tree A, observe that $G_S(A)$ will be a sequence of nonnegative integers. Hence, $G_S(A)$ will represent a trimmed tree.

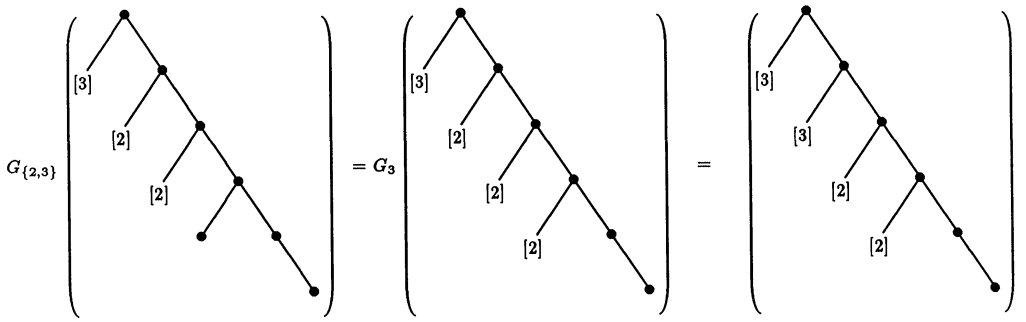


FIG. 3.1

We are now ready to characterize the tape-optimal trees in terms of D.

THEOREM 3.4. *A tree is tape-optimal if and only if it is equivalent to $G_S(A)$ for some m-tape-complete tree A and some $S \subseteq \{0, 1, \cdots, m\}$.*

We need the following lemmas to prove this theorem.

LEMMA 3.5. *Let $A=(a_m, a_{m-1}, \cdots, a_0)$ be a trimmed tree and let $j \in \{0, 1, \cdots, m-1\}$.*

(a) *If $j>0$ and $B=(a_m, a_{m-1}, \cdots, a_{j+1}-1, a_j+2, a_{j-1}-1, \cdots, a_0)$ is a trimmed tree then*

$$T(A)-T(B)=D(j+1)-D(j)-a_j. \tag{3.0}$$

(b) *If $j=0$ and $B=(a_m, a_{m-1}, \cdots, a_1-1, a_0+2)$ is a trimmed tree then* (3.0) *holds.*

Proof. We will prove (a). (The proof of (b) is similar and will be omitted.) Since B can be obtained from A by adding a node to a subtree of A with $j-1$ nodes and removing a node from a subtree with $j+1$ nodes, A and B have the same number of nodes. We will use the notation $X \backslash Y$ to denote the tree formed by attaching the tree Y as a right subtree of the rightmost node of the tree X. For example see Fig. 3.2.

It is easy to see that

$$T(A)-T(B)=T(A')-T(B'), \tag{3.1}$$

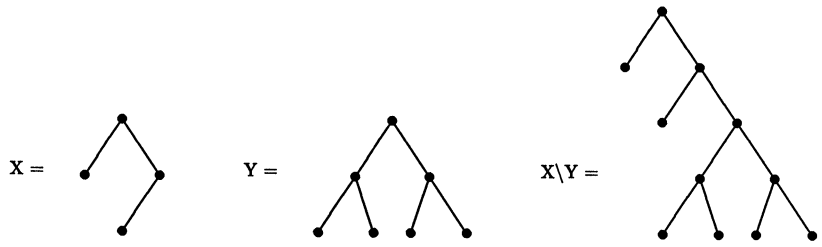


FIG. 3.2

where

$$A' = \langle j+1, \underbrace{j, \cdots, j}_{a_j}, j-1\rangle \backslash C,\ B' = \langle \underbrace{j, j, \cdots, j, j}_{a_j+2}\rangle \backslash C \quad \text{and} \quad C = (a_{j-1}-1, a_{j-2}, \cdots, a_0).$$

A straightforward computation results in

$$T(A') - T(B') = D(j+1) + |A'| - (j+2)a_j + T([j-1] \wedge C) - T([j] \wedge C). \tag{3.2}$$

Another computation gives

$$T([j-1] \wedge C) - T([j] \wedge C) = -D(j) - |C| - 2(j+1).$$

Combining this with (3.1) and (3.2) we get

$$T(A) - T(B) = D(j+1) - D(j) + |A'| - (j+2)a_j - |C| - 2(j+1).$$

Since $|A'| = (a_j+2)(j+1) + |C|$, we have

$$T(A) - T(B) = D(j+1) - D(j) - a_j. \qquad \square$$

The next lemma shows that it costs more to grow a leaf on certain subtrees along the rightmost path of $G_S(A)$ than on other subtrees.

LEMMA 3.6. *Let $A = (a_m, a_{m-1}, \cdots, a_0)$ be m-tape-complete and $S \subseteq \{0, 1, \cdots, m\}$. Then there is a positive integer γ such that*

(a) $$T(G_i G_S(A)) = \begin{cases} \gamma & \text{if } i \in \{0, 1, \cdots, m\} - S, \\ \gamma + 1 & \text{if } i \in S, \end{cases}$$

(b) *if $a_m > 0$ then $T(G_{m+1} G_S(A)) > \gamma$.*

Proof. Let $G_S(A) = (a'_m, a'_{m-1}, \cdots, a'_0)$. Observe that for $i \in \{0, 1, \cdots, m\}$ we have,

$$\begin{aligned} &i \in S,\ i+1 \notin S \ \Rightarrow\ a'_i = a_i + 1, \\ &i \notin S,\ i+1 \in S \ \Rightarrow\ a'_i = a_i - 1, \\ &i \in S,\ i+1 \in S \ \Rightarrow\ a'_i = a_i, \\ &i \notin S,\ i+1 \notin S \ \Rightarrow\ a'_i = a_i. \end{aligned} \tag{3.3}$$

Hence $a'_i \geqq a_i - 1$ for all $i = 0, \cdots, m-1$ and $a'_m \geqq a_m$. Since for $i = 0, \cdots, m-1$, $a_i = D(i+1) - D(i) + 1$, by Lemma 2.4 $a_i \geqq 2$. Hence $G_S(A)$ is a proper trimmed tree. It follows that $G_i G_S(A)$ is a trimmed tree for all $i = 0, 1, \cdots, m$. If $a_m > 0$ then $a'_m > 0$. So in this case, $G_{m+1} G_S(A)$ is also a trimmed tree. Since we are dealing with trimmed trees, applying T to $G_i G_S(A)$ is legal for all $i = 0, 1, \cdots, m$, and for $i = m+1$ when $a_m > 0$.

Let γ be defined by

$$\gamma = \begin{cases} T(G_0 G_S(A)) & \text{if } 0 \notin S, \\ T(G_0 G_S(A)) - 1 & \text{if } 0 \in S. \end{cases}$$

We prove (a) by induction on i. For $i = 0$, (a) follows from the definition of γ. Suppose $0 \leqq i \leqq m-1$ and (a) holds for i. We have

$$G_i G_S(A) = (a'_m, a'_{m-1}, \cdots, a'_{i+1}, a'_i + 1, a'_{i-1} - 1, \cdots, a'_0),$$

$$G_{i+1} G_S(A) = (a'_m, a'_{m-1}, \cdots, a'_{i+1} + 1, a'_i - 1, a'_{i-1}, \cdots, a'_0),$$

where for $i = 0$ this should be appropriately modified. By Lemma 3.5,

$$\begin{aligned} T(G_{i+1} G_S(A)) - T(G_i G_S(A)) &= D(i+1) - D(i) - (a'_i - 1) \\ &= a_i - a'_i. \end{aligned}$$

It follows from this and (3.3) that

$$T(G_{i+1} G_S(A)) - T(G_i G_S(A)) = \begin{cases} -1 & \text{if } i \in S, \quad i+1 \notin S, \\ 1 & \text{if } i \notin S, \quad i+1 \in S, \\ 0 & \text{if } i, i+1 \in S \text{ or } i, i+1 \notin S. \end{cases}$$

We now invoke the induction hypothesis to get

$$T(G_{i+1} G_S(A)) = \begin{cases} \gamma + 1 - 1 & \text{if } i \in S, i+1 \notin S, \\ \gamma + 1 & \text{if } i \notin S, i+1 \in S, \\ \gamma + 1 + 0 & \text{if } i, i+1 \in S, \\ \gamma + 0 & \text{if } i, i+1 \notin S, \end{cases}$$

$$= \begin{cases} \gamma & \text{if } i+1 \notin S, \\ \gamma + 1 & \text{if } i+1 \in S. \end{cases}$$

Hence (a) holds for all $i = 0, 1, \cdots, m$.

To prove (b) we apply Lemma 3.5, just as for (a) to get

$$T(G_{m+1} G_S(A)) - T(G_m G_S(A)) = D(m+1) - D(m) - (a'_m - 1) > a_m - a'_m.$$

From this and (3.3) we get

$$T(G_{m+1} G_S(A)) - T(G_m G_S(A)) > \begin{cases} 0 & \text{if } m \notin S, \\ -1 & \text{if } m \in S. \end{cases}$$

Since by (a)

$$T(G_m G_S(A)) = \begin{cases} \gamma & \text{if } m \notin S, \\ \gamma + 1 & \text{if } m \in S, \end{cases}$$

we have

$$T(G_{m+1} G_S(A)) > \begin{cases} \gamma & \text{if } m \notin S, \\ \gamma + 1 - 1 & \text{if } m \in S. \end{cases}$$

Hence $T(G_{m+1} G_S(A)) > \gamma$. □

LEMMA 3.7. *A nonempty tree A is tape-complete if and only if $A = G_S(A')$ for some m-tape-complete tree A' and $S = \{0, 1, \cdots, m\}$.*

Proof. Let $A=(a_m, a_{m-1}, \cdots, a_0)$ with $a_m>0$. We have $A=G_{\{0,1,\cdots,m\}}(A')$, where $A'=(a_m-1, a_{m-1}, \cdots, a_0)$. If $a_m=D(m+1)-D(m)+1$, then clearly A is $(m+1)$-tape-complete if and only if A' is m-tape-complete. If $a_m<D(m+1)-D(m)+1$ then A is m-tape-complete if and only if A' is m-tape-complete. □

Proof of Theorem 3.4. ($\Rightarrow$) Let B be tape-optimal. By Proposition 2.2 we can assume that B is trimmed. We will show by induction on $|B|$, that $B=G_S(A)$ for some m-tape-complete tree A and some $S\subseteq\{0, 1, \cdots, m\}$. For $|B|=0$, the result is trivial.

Assume $|B|\geqq 1$. By Lemma 2.3(b), $B=G_i(B')$ for some tape-optimal trimmed tree B', and for some $i\geqq 0$. By induction $B'=G_S(A)$ for some m-tape-complete tree A and some $S\subseteq\{0, 1, \cdots, m\}$. By Lemma 3.7 we can assume that $S\neq\{0, 1, \cdots, m\}$. Since B is tape-optimal, Lemma 3.6 implies that $i\in\{0, 1, \cdots, m\}-S$. Let $S'=S\cup\{i\}$. Since $S'\subseteq\{0, 1, \cdots, m\}$ and $B=G_{S'}(A)$ we are done.

($\Leftarrow$) Now suppose B is equivalent to $G_S(A)$ for some m-tape-complete tree A and some $S\subseteq\{0, 1, \cdots, m\}$. We can assume that $B=G_S(A)$. We show that B is tape-optimal by induction on $|B|$. For $|B|=0$ the result is trivial.

Suppose $|B|\geqq 1$. By Lemma 3.7 we can assume that $S\neq\varnothing$. Let $i\in S$ and let $B'=G_{S-\{i\}}(A)$. By induction B' is tape-optimal. It now follows from Lemmas 2.3(a) and 3.6 that $B=G_i(B')$ is top-optimal. □

4. Evaluation of $D(n+1)-D(n)$. In § 3 we described the tape-complete trees in terms of the $D(n+1)-D(n)$. We now describe the $D(n+1)-D(n)$ in terms of the tape-complete trees. First we need a lemma.

LEMMA 4.1. *Let $A=(a_m, a_{m-1}, \cdots, a_0)$ be a trimmed tree. Then for any $i=1, \cdots, m$ where $a_{i-1}\geqq 2$ and for $i=0$,*

$$T(G_i^2(A))+T(A)-2T(G_i(A))=2.$$

Proof. The following equation can easily be established:

$$T([m]\wedge X)+T([m]\wedge Z)-2T([m]\wedge Y)=T(X)+T(Z)-2T(Y)$$

whenever $m\geqq 0$ and $|X|+2=|Y|+1=|Z|$. It follows from this that for $i>0$,

$$T(G_i^2(A))+T(A)-2T(G_i(A))=T(\langle i, i\rangle\backslash B)+T(\langle i-1, i-1\rangle\backslash B)-2T(\langle i, i-1\rangle\backslash B),$$

where B is some trimmed tree. A straightforward computation shows that the right-hand side of this equation equals 2. A similar argument works for $i=0$. □

THEOREM 4.2. *Let A be a trimmed tape-optimal tree with n nodes. Then*

$$D(n+1)-D(n)=\begin{cases}1 & \text{if } A \text{ is not tape-complete,}\\ 2 & \text{if } A \text{ is tape-complete.}\end{cases}$$

Proof. For $n=0$, the result holds trivially. Suppose A is tape-complete and $n\geqq 1$. We apply Lemma 3.7 to get $A=G_S(A')$ where A' is m-tape-complete and $S=\{0, 1, \cdots, m\}$. So $G_S(A')$ is tape-optimal with n nodes. Let $i\in S$. By Theorem 3.4, $G_{S-\{i\}}(A')$ is tape-optimal with $n-1$ nodes and $G_i(A)$ is tape-optimal with $n+1$ nodes. It follows that

$$\begin{aligned}D(n+1)-D(n)&=T(n+1)+T(n-1)-2T(n)\\&=T(G_i(A))+T(G_{S-\{i\}}(A'))-2T(G_S(A'))\\&=T(G_i^2G_{S-\{i\}}(A'))+T(G_{S-\{i\}}(A'))-2T(G_iG_{S-\{i\}}(A'))\\&=2,\end{aligned}$$

with the last step following from Lemma 4.1 (since $G_iG_{S-\{i\}}(A')=A$ is proper, the hypothesis of the lemma is satisfied).

Now suppose A is not tape-complete. By Theorem 3.4, $A = G_S(A')$ where A' is m-tape-complete and $S \subseteq \{0, 1, \cdots, m\}$. By Lemma 3.7, $S \neq \{0, 1, \cdots, m\}$. We also have that $S \neq \varnothing$. Let $i \in S$ and let $j \in \{0, 1, \cdots, m\} - S$. By Theorem 3.4, $G_{S-\{i\}}(A')$ is tape-optimal with $n-1$ nodes and $G_jG_S(A')$ is tape-optimal with $n+1$ nodes. By Lemma 3.6,

$$T(G_jG_S(A')) = T(G_iG_S(A')) - 1.$$

We now have

$$\begin{aligned} D(n+1) - D(n) &= T(G_jG_S(A')) + T(G_{S-\{i\}}(A')) - 2T(G_S(A')) \\ &= T(G_iG_S(A')) - 1 + T(G_{S-\{i\}}(A')) - 2T(G_S(A')) \\ &= T(G_i^2G_{S-\{i\}}(A')) - T(G_{S-\{i\}}(A')) - 2T(G_iG_{S-\{i\}}(A')) - 1 \\ &= 2 - 1 \quad \text{(by Lemma 4.1)} \\ &= 1. \end{aligned}$$

□

5. The solution. Let $w = w_0w_1w_2 \cdots \in \{2, 3\}^\infty$ be the infinite word defined recursively by

$$w = 3 \prod_{i=0}^{\infty} (2^i3)^{w_i},$$

where the relevant operation is concatenation. For each $m \geqq 0$, let

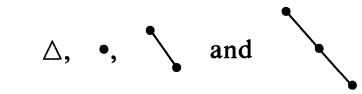
$$N_m = \sum_{i=0}^{m} (i+1)w_i.$$

THEOREM 5.1. *For each* $n \geqq 0$, $D(n+1) - D(n) + 1 = w_n$.

Proof. We prove the following by induction on m:

$$\prod_{j=0}^{N_m} (D(j+1) - D(j) + 1) = 3 \prod_{i=0}^{m} (2^i3)^{w_i}.$$

For $m = 0$, $N_m = 3$. Since $D(1) - D(0) + 1 = 3$, the tree (j) is 0-tape-complete for $0 \leqq j < 3$, and the tree $(0, 3)$ is 1-tape-complete. This means that the trees,



are tape-complete. Hence by Theorem 4.2,

$$\begin{aligned} \prod_{j=0}^{3} (D(j+1) - D(j) + 1) &= 3333 \\ &= 3(3)^{w_0}. \end{aligned}$$

Suppose $m > 0$. Assume by induction that

$$\prod_{j=0}^{N_{m-1}} (D(j+1) - D(j) + 1) = 3 \prod_{i=0}^{m-1} (2^i3)^{w_i}. \tag{5.1}$$

Consider the tree $A=(d, w_{m-1}, w_{m-2}, \cdots, w_0)$ where $0 \leqq d < w_m$. Since $N_{m-1} \geqq m$, it follows from (5.1) that $w_i = D(i+1) - D(i) + 1$ for all $i = 0, 1, \cdots, m$. Hence A is m-tape-complete. By Theorem 4.2 and Lemma 3.7,

$$D(|G_S(A)|+1) - D(|G_S(A)|) + 1 = \begin{cases} 2 & \text{if } \varnothing \neq S \subsetneq \{0, 1, \cdots, m\}, \\ 3 & \text{if } S = \{0, 1, \cdots, m\}. \end{cases}$$

Since $|G_S(A)| = N_{m-1} + d(m+1) + |S|$ we have

$$\prod_{r=1}^{m+1} (D(N_{m-1} + d(m+1) + r + 1) - D(N_{m-1} + d(m+1) + r) + 1) = 2^m 3.$$

Consequently,

$$\prod_{d=0}^{w_m - 1} \prod_{r=1}^{m+1} (D(N_{m-1} + d(m+1) + r + 1) - D(N_{m-1} + d(m+1) + r) + 1) = (2^m 3)^{w_m}.$$

This is the same as

$$\prod_{j=N_{m-1}+1}^{K} (D(j+1) - D(j) + 1) = (2^m 3)^{w_m},$$

where $K = N_{m-1} + (w_m - 1)(m+1) + m + 1 = N_{m-1} + w_m(m+1) = N_m$. Combining this with (5.1) we complete the induction. □

We are now able to explicitly describe the tape-optimal trees.

THEOREM 5.2. *Let $N_{m-1} \leqq n < N_m$ and let d and r be determined by $n - N_{m-1} = d(m+1) + r$, where $d \geqq 0$, $0 \leqq r < m+1$. Then a tree with n nodes is tape-optimal if and only if it is equivalent to $G_S(d, w_{m-1}, w_{m-2}, \cdots, w_0)$ for some $S \subsetneq \{0, 1, \cdots, m\}$ with $|S| = r$.*

Proof. Since $n < N_m$, we have that $n - N_{m-1} < w_m(m+1)$. Hence $d < w_m$. It follows from Theorem 5.1 that $(d, w_{m-1}, \cdots, w_0)$ is m-tape-complete. The result now follows from Theorem 3.4. □

The rightmost tape-optimal tree has a particularly simple description.

COROLLARY 5.3. *Let $n, m, d,$ and r be as in Theorem 5.2. Then the rightmost tape-optimal tree with n nodes is $(d, w_{m-1}, \cdots, w_{r-1}+1, \cdots, w_0)$ if $r > 0$ and is $(d, w_{m-1}, \cdots, w_0)$ if $r = 0$.* □

We are also able to express the cost of a tape-optimal tree in terms of the w_i.

COROLLARY 5.4. *For all $n \geqq 0$,*

$$\begin{aligned} T(n) &= \sum_{j=0}^{n-1} (w_j - 1)(n-j) \\ &= (n+1) \sum_{j=0}^{n-1} w_j - N_{n-1} - \frac{n(n+1)}{2}. \end{aligned}$$

Proof. Since $D(i) = T(i) - T(i-1)$, we have

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} D(i) \\ &= \sum_{i=1}^{n} \sum_{j=0}^{i-1} (D(j+1) - D(j)). \end{aligned}$$

By Theorem 5.1 we can replace $D(j+1) - D(j)$ with $w_j - 1$, to get

$$T(n) = \sum_{i=1}^{n} \sum_{j=0}^{i-1} (w_j - 1).$$

By reversing the order of summation we obtain the first expression. We leave it to the reader to show that the second expression is equal to the first. □

It follows immediately from Corollary 5.4 that the cost of a tape-optimal tree lies between $n(n+1)/2$ and $n(n+1)$. This can also be proved directly by induction using the recurrence relation for $T(n)$. Since $n(n+1)$ is the cost of sequential search on a tape, we see that the optimal tape searching strategy is at best twice as fast as sequential search. Since most of the w_i are equal to 2, it seems likely that the cost of the optimal tape searching strategy will approach $n(n+1)/2$ as n gets large. Elementary considerations, using Theorem 5.2, show that the size of the left subtree of a tape-optimal tree falls between $(2n/3)^{1/2}-2$ and $n^{1/2}-1$. It also appears likely that the actual size approaches the upper bound as n gets large.

6. Generalizations and related problems. We now consider the general tape searching problem in which the cost of each comparison is not necessarily the same as the cost of each movement. If a cost of α is assigned to each movement and a cost of β is assigned to each comparison then the total cost of searching for every node of tree A is given by

$$S(A)=\alpha M(A)+\beta C(A).$$

It turns out that if $2\alpha/\beta$ is an integer then all of our results still hold with certain modifications.

Let $p=2\alpha/\beta$ and $q=p+1$. We define $w\in\{p, q\}^{\infty}$ as follows:

$$w=q\prod_{i=0}^{\infty}(p^iq)^{w_i}.$$

Let $N_m=\sum_{i=0}^{m}(i+1)w_i$. The general tape-complete trees are now the trees $(d, w_{m-1}, w_{m-2}, \cdots, w_0)$, where $0\leqq d<w_m$ and Theorem 5.2 holds in the general setting. Before discussing the modifications that must be made in proving this we consider some examples.

For the usual binary search ($\alpha=0$), we get that $p=0$ and $q=1$. Hence w is the infinite word defined recursively by

$$w=1\prod_{i=0}^{\infty}(0^i1)^{w_i}.$$

In this case w_i has a very simple explicit formulation. It can be shown that

$$w_i=\begin{cases}1 & \text{if } i=2^k-1 \quad \text{for some } k=0, 1, \cdots,\\ 0 & \text{otherwise.}\end{cases}$$

Hence the tape-complete trees for $\alpha=0$ are the usual complete binary trees which is exactly what one would expect. Note that the almost complete binary trees can be obtained by applying G_S to the complete binary trees. Hence the tape-optimal trees for $\alpha=0$ are the almost complete binary trees and the complete binary trees.

The tape-optimal trees for sequential search ($\beta=0$) are not a special case of our results since p is not defined for $\beta=0$. However we can consider the situation as β approaches 0. In this case q will approach ∞ and any tape-optimal tree with less than $w_0=q$ nodes will be the linear tree. Hence the tape-optimal trees approach the linear tree as β approaches 0.

Tape-optimal trees with 46 nodes are given in Fig. 6.1(a) for $\alpha=1, \beta=2$ and in Fig. 6.1(b) for $\alpha=2, \beta=1$. Recall that a tape-optimal tree with 46 nodes for $\alpha=\beta=1$ is given in Fig. 1.2. Observe that the trees become less balanced as α/β gets larger.

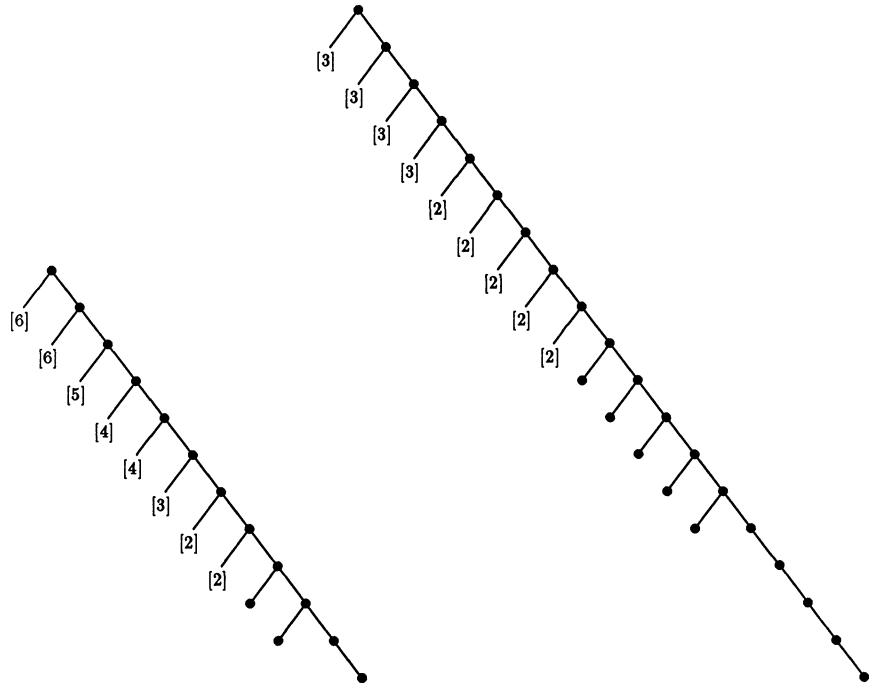


FIG. 6.1

We now list the results of §§ 2–5 which must be modified to prove Theorem 5.2 in the general setting.

LEMMA 2.4′. *For all* $n \geqq 0$, $D(n+1) - D(n) \geqq \alpha$.

DEFINITION 3.1′. A trimmed tree $(a_m, a_{m-1}, \cdots, a_0)$ is said to be m-tape-complete if $a_i = (D(i+1) - D(i) + \alpha)/\beta$ for $i = 0, 1, \cdots, m-1$ and $0 \leqq a_m < (D(m+1) - D(m) + \alpha)/\beta$.

Remark. For this definition to make sense, $(D(i+1) - D(i) + \alpha)/\beta$ must be an integer. We assume inductively that this holds for $i < n$. Theorem 4.2′ and the induction hypothesis will establish that $(D(n+1) - D(n) + \alpha)/\beta$ is an integer for all n.

LEMMA 3.5′. *Let* $A = (a_m, a_{m-1}, \cdots, a_0)$ *be a trimmed tree and let* $j \in \{0, 1, \cdots, m-1\}$.

(a) *If* $j > 0$ *and* $B = (a_m, a_{m-1}, \cdots, a_{j+1} - 1, a_j + 2, a_{j-1} - 1, \cdots, a_0)$ *is a trimmed tree then*

$$T(A) - T(B) = D(j+1) - D(j) + \alpha - \beta - \beta a_j. \tag{6.0}$$

(b) *If* $j = 0$ *and* $B = (a_m, a_{m-1}, \cdots, a_1 - 1, a_0 + 2)$ *is a trimmed tree then* (6.0) *holds.*

LEMMA 3.6′. *Let* $A = (a_m, a_{m-1}, \cdots, a_0)$ *be* m*-tape-complete and* $S \subseteq \{0, 1, \cdots, m\}$. *Then there is a positive integer* γ *such that*

(a) $T(G_i G_S(A)) = \begin{cases} \gamma & \text{if } i \in \{0, 1, \cdots, m\} - S, \\ \gamma + \beta & \text{if } i \in S. \end{cases}$

(b) *If* $a_m > 0$ *then* $T(G_{m+1} G_S(A)) > \gamma$.

LEMMA 4.1′. *Let* $A=(a_m, a_{m-1}, \cdots, a_0)$ *be a trimmed tree. Then for any* $i=1, \cdots, m$ *where* $a_{i-1}\geqq 2$ *and for* $i=0$,

$$T(G_i^2(A))+T(A)-2T(G_i(A))=\alpha+\beta.$$

THEOREM 4.2′. *Let A be a trimmed tape-optimal tree with n nodes. Then*

$$D(n+1)-D(n)=\begin{cases}\alpha & \text{if } A \text{ is not tape-complete,}\\ \alpha+\beta & \text{if } A \text{ is tape-complete.}\end{cases}$$

THEOREM 5.1′. *For each* $n\geqq 0$, $(D(n+1)-D(n)+\alpha)/\beta=w_n$.

Many of these results hold even when $2\alpha/\beta$ is not an integer. In particular Lemma 2.3 holds. It follows that tape-optimal trees can be grown in a greedy fashion (Theorem 1.1). Hence we can grow tape-optimal trees of size n by making only n comparisons. This gives an $O(n)$ algorithm for finding tape-optimal trees when $2\alpha/\beta$ is not necessarily an integer. These tape-optimal trees appear to have similar characteristics to the ones for integer $2\alpha/\beta$ and is a subject for future investigation.

A still more general tape searching problem is that in which the records have varying lengths and varying probabilities of being searched. By extending the results of Knuth [3, p. 436] and Yao [7], Wachs [6] has obtained an $O(n^2)$ dynamic programming algorithm for finding the optimal trees for this problem.

Another tape searching problem in which the records have varying lengths and probabilities of being searched is discussed in [3, pp. 400–401]. Instead of considering optimal search strategies, an algorithm of Smith [4] is used to find an optimal arrangement of records for a sequential search of the tape.

REFERENCES

[1] A. J. HU, *Problem* 84-3, J. Algorithms, 5 (1984), p. 579.
[2] ———, *Selection of the optimum uniform partition search*, Computing (WIEN), 37 (1986), pp. 261–264.
[3] D. E. KNUTH, *Sorting and searching*, in The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, MA, 1973.
[4] W. E. SMITH, *Various optimizers for single-state production*, Naval Res. Logist. Quart., 3 (1956), pp. 59–66.
[5] J. D. ULLMAN, *Principles of Database Systems*, Computer Science Press, Rockville, MD, 1982.
[6] M. L. WACHS, *An extension of Yao's dynamic programming quadrangle inequalities*, to appear.
[7] F. F. YAO, *Efficient dynamic programming using quadrangle inequalities*, Proceedings of the 12th ACM Symposium on Theory of Computing, April 1980, pp. 429–435.

SIAM J. COMPUT.
Vol. 16, No. 3 June 1987
© 1987 Society for Industrial and Applied Mathematics
011

FACTORING MULTIVARIATE POLYNOMIALS OVER ALGEBRAIC NUMBER FIELDS*

ARJEN K. LENSTRA†

Abstract. We present an algorithm to factor multivariate polynomials over algebraic number fields that is polynomial-time in the degrees of the polynomial to be factored. The algorithm is an immediate generalization of the polynomial-time algorithm to factor univariate polynomials with rational coefficients.

Key words. polynomial factorization

AMS(MOS) subject classifications. 12A20, 68C25

1. Introduction. We show that the algorithm from [9] to factor univariate polynomials with rational coefficients can be generalized to multivariate polynomials with coefficients in an algebraic number field. As a result we get an algorithm that is polynomial-time in the degrees and the coefficient-size of the polynomial to be factored.

An outline of the algorithm is as follows. First the polynomial $f \in \mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$ is evaluated in a suitably chosen integer point $(X_2 = s_2, X_3 = s_3, \cdots, X_t = s_t)$. Next, for some prime number p, a p-adic irreducible factor $\tilde{h}$ of the resulting polynomial $\tilde{f} \in \mathbf{Q}(\alpha)[X_1]$ is determined up to a certain precision. We then show that the irreducible factor h_0 of f for which $\tilde{h}$ is a p-adic factor of $\tilde{h}_0$, belongs to a certain integral lattice, and that h_0 is relatively short in this lattice. This enables us to compute this factor h_0 by means of the so-called *basis reduction algorithm* (cf. [9, § 1]).

As [9] is easily available, we do not consider it to be necessary to recall the basis reduction algorithm here; we will assume the reader to be familiar with this algorithm and its properties.

Although the algorithm presented in this paper is polynomial-time, we do not think it is a useful method for practical purposes. Like the other generalizations of the algorithm from [9], which can be found in [10]–[13], the algorithm will be slow, because the basis reduction algorithm has to be applied to huge dimensional lattices with large entries. In practice, a combination of the methods from [8], [17] and [18] can be recommended (cf. [8]).

In [3] and [6] polynomial-time algorithms are given for similar multivariate polynomial factoring problems. These papers make use of a completely different approach. In [6] the multivariate factoring is reduced in polynomial time to univariate factoring and the latter problem is solved by the algorithm from [9]. A polynomial-time reduction to univariate factoring and multivariate factoring over finite fields is presented in [3].

All papers referred to above, and also the present paper, assumed the polynomials to be *densely* encoded. For algorithms which apply the more realistic *sparse* encoding we refer to [16].

2. Preliminaries. In this section we introduce some notation, and we derive an upper bound for the coefficients of factors of multivariate polynomials over algebraic number fields.

* Received by the editors February 4, 1985; accepted for publication (in revised form) September 9, 1986.
† Department of Computer Science, The University of Chicago, Chicago, Illinois 60637.

Let the algebraic number field $\mathbf{Q}(\alpha)$ be given as the field of rational numbers $\mathbf{Q}$ extended by a root α of a prescribed *minimal polynomial* $F \in \mathbf{Z}[T]$ with leading coefficient equal to one, i.e., $\mathbf{Q}(\alpha) \simeq \mathbf{Q}[T]/(F)$. Similarly, we define $\mathbf{Z}[\alpha] = \mathbf{Z}[T]/(F)$ as a ring of polynomials in α over $\mathbf{Z}$ of degree less than I, where I denotes the degree δF of F.

Let $f \in \mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$ be the polynomial to be factored, with the number of variables $t \geqq 2$. By $\delta_i f = n_i$ we denote the degree of f in X_i, for $1 \leqq i \leqq t$. We often use n instead of n_1. We put $N_i = \prod_{k=i}^{t} (n_k + 1)$, and $N = N_1$. Let $lc_0(f) = f$. For $1 \leqq i \leqq t$ we define $lc_i(f) \in \mathbf{Q}(\alpha)[X_{i+1}, X_{i+2}, \cdots, X_t]$ as the leading coefficient with respect to X_i of $lc_{i-1}(f)$, and we put $lc(f) = lc_t(f)$. Finally, we define the *content* $\operatorname{cont}(f) \in \mathbf{Q}(\alpha)[X_2, X_3, \cdots, X_t]$ of f as the greatest common divisor of the coefficients of f with respect to X_1. Without loss of generality we may assume that $2 \leqq n_i \leqq n_{i+1}$ for $1 \leqq i < t$, that f is *monic* (i.e. $lc(f) = 1$), and that $\delta_i \operatorname{cont}(f) = 0$ for $2 \leqq i \leqq t$.

Let $d \in \mathbf{Z}_{>0}$ be such that $f \in (1/d)\mathbf{Z}[\alpha][X_1, X_2, \cdots, X_t]$, and let $\operatorname{discr}(F)$ denote the discriminant of F. It is well known (cf. [18]) that if we take $D = d|\operatorname{discr}(F)|$, then all monic factors of f are in $(1/D)\mathbf{Z}[\alpha][X_1, X_2, \cdots, X_t]$ (in fact it is sufficient to take $D = d \cdot s$, where s is the largest integer such that s^2 divides $\operatorname{discr}(F)$, but this integer s might be too difficult to compute).

We now introduce some notation. Suppose that we are given a prime number p such that

(2.1) $\qquad p$ does not divide D.

For $G = \sum_i a_i T^i \in \mathbf{Z}[T]$ we denote by G_l or $G \bmod p^l$ the polynomial $\sum_i (a_i \bmod p^l) T^i \in (\mathbf{Z}/p^l\mathbf{Z})[T]$, for any positive integer l. Suppose furthermore that we are given some positive integer k, and that p is chosen in such a way that a polynomial $H \in \mathbf{Z}[T]$ exists such that

(2.2) $\qquad H$ has leading coefficient equal to one,

(2.3) $\qquad H_k$ divides F_k in $(\mathbf{Z}/p^k\mathbf{Z})[T]$,

(2.4) $\qquad H_1$ is irreducible in $(\mathbf{Z}/p\mathbf{Z})[T]$,

(2.5) $\qquad (H_1)^2$ does not divide F_1 in $(\mathbf{Z}/p\mathbf{Z})[T]$.

Clearly H_1 divides F_1 in $(\mathbf{Z}/p\mathbf{Z})[T]$, and $0 < \delta H \leqq I$. In the sequel we will assume that conditions (2.1), (2.2), (2.3), (2.4) and (2.5) are satisfied.

By $\mathbf{F}_q$ we denote the finite field containing $q = p^{\delta H}$ elements. From (2.4) we have $\mathbf{F}_q \simeq (\mathbf{Z}/p\mathbf{Z})[T]/(H_1) \simeq \{\sum_{i=0}^{\delta H - 1} a_i \alpha_1^i : a_i \in \mathbf{Z}/p\mathbf{Z}\}$, where $\alpha_1 = T \bmod (H_1)$ is a zero of H_1. Furthermore we put $W_k(\mathbf{F}_q) = (\mathbf{Z}/p^k\mathbf{Z})[T]/(H_k) = \{\sum_{i=0}^{\delta H - 1} a_i \alpha_k^i : a_i \in \mathbf{Z}/p^k\mathbf{Z}\}$, where $\alpha_k = T \bmod (H_k)$ is a zero of H_k. Notice that $W_k(\mathbf{F}_q)$ is a ring containing q^k elements, and that $W_1(\mathbf{F}_q) \simeq \mathbf{F}_q$. For $a \in \mathbf{Z}[\alpha]$ we denote by $a \bmod (p^l, H_l) \in W_l(\mathbf{F}_q)$ the result of the canonical mapping from $\mathbf{Z}[\alpha] = \mathbf{Z}[T]/(F)$ to $W_l(\mathbf{F}_q) = (\mathbf{Z}/p^l\mathbf{Z})[T]/(H_l)$ applied to a, for $l = 1, k$. For $\tilde{g} = \sum_i (a_i/D) X_1^i \in (1/D)\mathbf{Z}[\alpha][X_1]$ we denote by $\tilde{g} \bmod (p^l, H_l)$ the polynomial $\sum_i (((D^{-1} \bmod p^l) a_i) \bmod (p^l, H_l)) X_1^i \in W_l(\mathbf{F}_q)[X_1]$ (notice that $D^{-1} \bmod p^l$ exists due to (2.1)).

We derive an upper bound for the height of a monic factor g of f. As usual, for $g = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_t} \sum_j a_{i_1 i_2 \cdots i_t j} \alpha^j X_1^{i_1} X_2^{i_2} \cdots X_t^{i_t} \in \mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$, the *height* $g_{\max}$ is defined as $\max |a_{i_1 i_2 \cdots i_t j}|$, and the *length* $|g|$ as $(\sum a_{i_1 i_2 \cdots i_t j}^2)^{1/2}$. Similarly, for a polynomial h with complex coefficients, we define its height h_{cmax} as the maximum of the absolute values of its complex coefficients.

For any choice of $\alpha \in \{\alpha_1, \alpha_2, \cdots, \alpha_I\}$, where $\alpha_1, \alpha_2, \cdots, \alpha_I$ are the conjugates of α, we can regard g as a polynomial g_α with complex coefficients. We define $\|g\|$ as $\max_{1 \leqq i \leqq I} (g_{\alpha_i})_{c\max}$. From [4] we have

$$\|g\| \leqq e^{\sum_{i=1}^{t} n_i} \|f\|.$$

In [10, § 4] we have shown that this leads to

$$g_{\max} \leqq e^{\sum_{i=1}^{t} n_i} \|f\| I(I-1)^{(I-1)/2} |F|^{I-1} |\operatorname{discr}(F)|^{-1/2}. \tag{2.6}$$

From [15] we know that the length $|F|$ of F is an upper bound for the absolute value of the conjugates of α, so that

$$\|f\| \leqq f_{\max} \sum_{i=0}^{I-1} |F|^i,$$

which yields, combined with (2.6),

$$g_{\max} \leqq e^{\sum_{i=1}^{t} n_i} f_{\max} I(I-1)^{(I-1)/2} |F|^{I-1} |\operatorname{discr}(F)|^{-1/2} \sum_{i=0}^{I-1} |F|^i. \tag{2.7}$$

The upper bound for the height of monic factors of f, as given by the right-hand side of (2.7), will be denoted by B_f. Because $|\operatorname{discr}(F)| \geqq 1$, we find

$$\log B_f = O\left(\sum_{i=1}^{t} n_i + \log f_{\max} + I \log (I|F|) \right). \tag{2.8}$$

3. Factoring multivariate polynomials over algebraic number fields. We describe an algorithm to compute the irreducible factorization of f in $\mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$.

Let $s_2, s_3, \cdots, s_t \in \mathbf{Z}_{>0}$ be a $(t-1)$-tuple of integers. For $g \in \mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$ we denote by $\tilde{g}_j$ the polynomial $g \bmod ((X_2 - s_2), (X_3 - s_3), \cdots, (X_j - s_j)) \in \mathbf{Q}(\alpha)[X_1, X_{j+1}, X_{j+2}, \cdots, X_t]$; i.e., $\tilde{g}_j$ is g with s_i substituted for X_i, for $2 \leqq i \leqq j$. Notice that $\tilde{g}_1 = g$ and that $\tilde{g}_j = \tilde{g}_{j-1} \bmod (X_j - s_j)$. We put $\tilde{g} = \tilde{g}_t$.

Suppose that a polynomial $\tilde{h} \in \mathbf{Z}[\alpha][X_1]$ is given such that

(3.1) $\tilde{h}$ is monic,

(3.2) $\tilde{h} \bmod (p^k, H_k)$ divides $\tilde{f} \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$,

(3.3) $\tilde{h} \bmod (p, H_1)$ is irreducible in $\mathbf{F}_q[X_1]$,

(3.4) $(\tilde{h} \bmod (p, H_1))^2$ does not divide $\tilde{f} \bmod (p, H_1)$ in $\mathbf{F}_q[X_1]$.

We put $l = \delta_1 \tilde{h}$, so $0 < l \leqq n$. By $h_0 \in (1/D)\mathbf{Z}[\alpha][X_1, X_2, \cdots, X_t]$ we denote the unique, monic, irreducible factor of f such that $\tilde{h} \bmod (p^k, H_k)$ divides $\tilde{h}_0 \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$ (cf. (3.2), (3.3), (3.4)).

(3.5) Let $m = m_1, m_2, m_3, \cdots, m_t$ be a t-tuple of integers satisfying $l \leqq m < n$ and $0 \leqq m_i \leqq \delta_i lc_{i-1}(f)$ for $2 \leqq i \leqq t$, and let $M = 1 + I \sum_{i=1}^{t} m_i N_{i+1}$ (where of course $N_{t+1} = 1$). We define $L \subset (\mathbf{Z}/D)^M$ as the lattice of rank M, consisting of the polynomials $g \in (1/D)\mathbf{Z}[\alpha][X_1, X_2, \cdots, X_t]$ for which

(i) $\delta_1 g \leqq m$ and $\delta_i g \leqq n_i$ for $2 \leqq i \leqq t$;

(ii) If $\delta_j lc_{j-1}(g) = m_j$ for $1 \leqq j \leqq i$, then $\delta_{i+1} lc_i(g) \leqq m_{i+1}$ for $1 \leqq i < t$;

(iii) If $\delta_i lc_{i-1}(g) = m_i$ for $1 \leqq i \leqq t$, then $lc(g) \in \mathbf{Z}$;

(iv) $\tilde{h} \bmod (p^k, H_k)$ divides $\tilde{g} \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$.

Here M-dimensional vectors and polynomials satisfying conditions (i), (ii) and (iii) are identified in the usual way (cf. [10, (2.6)] and [12, (3.5)]). For notational convenience

we only give a basis for L in the case that $m_i = n_i$ for $2 \leqq i \leqq t$; the general case can easily be derived from this:

$$\left\{\frac{1}{D} p^k \alpha^j X_1^i : 0 \leqq j < \delta H, 0 \leqq i < l\right\}$$

$$\cup \left\{\frac{1}{D} \alpha^{j-\delta H} H(\alpha) X_1^i : \delta H \leqq j < I, 0 \leqq i < l\right\}$$

$$\cup \left\{\frac{1}{D} \alpha^j \tilde{h} X_1^{i-l} : 0 \leqq j < I, l \leqq i \leqq m\right\}$$

$$\cup \left\{\frac{1}{D} \alpha^j X_1^{i_1} \prod_{r=2}^{t} (X_r - s_r)^{i_r} : 0 \leqq j < I, 0 \leqq i_1 \leqq m, 0 \leqq i_r \leqq n_r \text{ for } 2 \leqq r \leqq t,\right.$$

$$(i_2, i_3, \cdots, i_t) \neq (0, 0, \cdots, 0),$$

$$\left.\text{and } (i_1, i_2, i_3, \cdots, i_t) \neq (m, n_2, n_3, \cdots, n_t)\right\}$$

$$\cup \left\{X_1^m \prod_{r=2}^{t} (X_r - s_r)^{n_r}\right\}$$

(cf. [10, (2.6)], [12, (3.5)], (2.2), and (3.1)).

PROPOSITION 3.6. *Let b be a nonzero element of L and let*

$$\tilde{B}_j = f_{\max}^m b_{\max}^n (n+m)! \left(DN_2 (1+F_{\max})^{I-1} \prod_{i=2}^{j} s_i^{n_i}\right)^{n+m}, \tag{3.7}$$

for $1 \leqq j \leqq t$, where $f_{\max}^m$ denotes $(f_{\max})^m$.

Suppose that

$$s_j \geqq ((n+m)n_j + 1)^{1/2} \tilde{B}_{j-1} \tag{3.8}$$

for $2 \leqq j \leqq t$, and

$$p^{k\delta H} \geqq |F|^{I-1} (I^{1/2} \tilde{B}_t)^I. \tag{3.9}$$

Then $\gcd(f, b) \neq 1$ in $\mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$.

Proof. Denote by $R = R(Df, Db) \in \mathbf{Z}[\alpha][X_2, X_3, \cdots, X_t]$ the resultant of Df and Db (with respect to the variable X_1). An outline of the proof is as follows. First we prove that an upper bound for $(\tilde{R}_j)_{\max}$ is given by $\tilde{B}_j$. Combining this with (3.8), we then see that $X_j = s_j$ cannot be a zero of $\tilde{R}_{j-1}$ if $\tilde{R}_{j-1} \neq 0$, for $2 \leqq j \leqq t$. This implies that the assumption that $R \neq 0$ (i.e., $\gcd(f, b) = 1$) leads to $\tilde{R} \neq 0$. We then apply a result from [8], and we find with (3.9) that $\tilde{R} \bmod (p^k, H_k) \neq 0$. But this is a contradiction, because $\tilde{h} \bmod (p^k, H_k)$ divides both $\tilde{f} \bmod (p^k, H_k)$ and $\tilde{b} \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$. We conclude that $R = 0$, so that $\gcd(f, b) \neq 1$ in $\mathbf{Q}(\alpha)[X_1, X_2, \cdots, X_t]$.

If a and b are two polynomials in any number of variables over $\mathbf{Q}(\alpha)$, having l_a and l_b terms, respectively, then

$$(a \cdot b)_{\max} \leqq a_{\max} b_{\max} \min(l_a, l_b)(1+F_{\max})^{I-1}. \tag{3.10}$$

From (3.10) we easily derive an upper bound for $(\tilde{R}_j)_{\max}$, because $\tilde{R}_j \in \mathbf{Z}[\alpha][X_{j+1}, X_{j+2}, \cdots, X_t]$ is the resultant of $D\tilde{f}_j$ and $D\tilde{b}_j$:

$$(\tilde{R}_j)_{\max} \leqq (D\tilde{f}_j)_{\max}^m (D\tilde{b}_j)_{\max}^n (n+m)! N_{j+1}^{n+m-1} (1+F_{\max})^{(I-1)(n+m-1)}. \tag{3.11}$$

It follows from $\tilde{f}_j = \tilde{f}_{j-1} \bmod (X_j - s_j)$, that $(\tilde{f}_j)_{\max} \leqq (\tilde{f}_{j-1})_{\max}(n_j+1)s_j^{n_j}$, so that

$$(\tilde{f}_j)_{\max} \leqq f_{\max} \prod_{i=2}^{j} (n_i+1)s_i^{n_i}. \tag{3.12}$$

Combining (3.11), (3.12) and a similar bound for $(\tilde{b}_j)_{\max}$, we obtain

$$(\tilde{R}_j)_{\max} < f_{\max}^m b_{\max}^n (n+m)! \left(DN_2 \prod_{i=2}^{j} s_i^{n_i} \right)^{n+m} (1+F_{\max})^{(I-1)(n+m-1)}, \tag{3.13}$$

for $1 \leqq j < t$. (Remark that (3.13) with "$<$" replaced by "$\leqq$" holds for $j = t$.)

Now assume, for some j with $2 \leqq j \leqq t$, that $\tilde{R}_{j-1}$ is unequal to zero. We prove that $\tilde{R}_j \neq 0$. Because $\tilde{R}_j = \tilde{R}_{j-1} \bmod (X_j - s_j)$, the condition $\tilde{R}_j = 0$ would imply that all polynomials in $\mathbf{Z}[X_j]$ that result from $\tilde{R}_{j-1}$ by grouping together all terms with identical exponents in α and X_{j+1} up to X_t, have $(X_j - s_j)$ as a factor. These polynomials have degree (in X_j) at most $(n+m)n_j$, so that we get, with the result from [14], that

$$|s_j| \leqq ((n+m)n_j+1)^{1/2}(\tilde{R}_{j-1})_{\max}.$$

Combined with (3.13) and (3.7) this is a contradiction with (3.8). We conclude that $\tilde{R}_j \neq 0$ if $\tilde{R}_{j-1} \neq 0$ for any j with $2 \leqq j \leqq t$, so that the assumption $\gcd(f, b) = 1$ (i.e., $R \neq 0$) leads to $\tilde{R} \neq 0$.

Assume that $H_k(T)$ divides $\tilde{R}(T) \in \mathbf{Z}[T]$ in $(\mathbf{Z}/p^k\mathbf{Z})[T]$, i.e., $\tilde{R} \bmod (p^k, H_k) = 0$. We show that this leads to a contradiction (cf. [8, proof of Thm. 2]). Consider the polynomials $T^i p^k$ for $0 \leqq i < \delta H$ and $T^{j-\delta H} H_k$ for $\delta H \leqq j < I + \delta\tilde{R}$. These polynomials, when regarded as $(I+\delta\tilde{R})$-dimensional integral vectors, form a basis for an integral lattice L_H in $\mathbf{Z}^{I+\delta\tilde{R}}$. From (2.2) it follows that the determinant $\det(L_H)$ of L_H equals $p^{k\delta H}$, and that L_H can be considered as the set of polynomials in $\mathbf{Z}[T]$ of degree less than $I + \delta\tilde{R}$ which have $H_k(T)$ as a divisor when taken modulo p^k.

Because F is irreducible and $\delta\tilde{R} < \delta F = I$, we have that $\gcd(F(T), \tilde{R}(T)) = 1$. This implies that the polynomials $T^i F$ for $0 \leqq i < \delta\tilde{R}$ and $T^j\tilde{R}$ for $0 \leqq j < I$, when regarded as $(I+\delta\tilde{R})$-dimensional integral vectors, form a basis for an integral lattice L_F in $\mathbf{Z}^{I+\delta\tilde{R}}$. Applying Hadamard's inequality to this basis for L_F, we see that $\det(L_F) \leqq |F|^{\delta\tilde{R}}((\delta\tilde{R}+1)^{1/2}\tilde{R}_{\max})^I$.

Our assumption $\tilde{R} \bmod (p^k, H_k) = 0$ combined with (2.3) leads to the conclusion that all elements of L_F, when considered as polynomials in $\mathbf{Z}[T]$, have $H_k(T)$ as a divisor when taken modulo p^k, so that L_F is a sublattice of L_H, and in particular

$$p^{k\delta H} = \det(L_H) \leqq \det(L_F) \leqq |F|^{I-1}(I^{1/2}\tilde{R}_{\max})^I.$$

With the remark after (3.13) and (3.7) this is a contradiction with (3.9), so that $\tilde{R} \bmod (p^k, H_k) \neq 0$. As stated in the first paragraph of this proof, this is a contradiction because $\tilde{h} \bmod (p^k, H_k)$ divides both $\tilde{f} \bmod (p^k, H_k)$ and $\tilde{b} \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$, so that our initial assumption $\gcd(f, b) = 1$ must be wrong. This concludes the proof of (3.6). □

PROPOSITION 3.14. *Let $b_1, b_2, \cdots, b_M$ be a reduced basis for L (cf. [9, § 1]), where L and M are as in* (3.5), *and let*

$$B_j = (n+m)!(M2^{M-1})^{n/2} \left(B_f DN_2 (1+F_{\max})^{I-1} \prod_{i=2}^{j} s_i^{n_i} \right)^{n+m}, \tag{3.15}$$

for $2 \leqq j \leqq t$, where B_f is as in § 2. *Suppose that*

$$s_j \geqq ((n+m)n_j+1)^{1/2} B_{j-1} \tag{3.16}$$

for $2 \leqq j \leqq t$, that

$$p^{k\delta H} \geqq |F|^{I-1}(I^{1/2}B_t)^I, \tag{3.17}$$

and that f does not contain multiple factors. Then

$$(b_1)_{\max} \leqq (M2^{M-1})^{1/2} B_f \tag{3.18}$$

and h_0 divides b_1, if and only if $h_0 \in L$.

Proof. If h_0 divides b_1, then $h_0 \in L$, because $b_1 \in L$ and $\delta_1 h_0 \leqq \delta_1 b_1$; this proves the "if" part.

To prove the "only if" part, suppose that $h_0 \in L$. Because h_0 is a monic factor of f, we have from (2.7) that $(h_0)_{\max} \leqq B_f$. With [9, (1.11)] and $h_0 \in L$ this gives $|b_1| \leqq (M2^{M-1})^{1/2} B_f$, so that (3.18) holds, because $(b_1)_{\max} \leqq |b_1|$. Because of (3.18), (3.16), (3.17), (3.15), and the definition of B_f, we can apply (3.6), which yields $\gcd(f, b_1) \neq 1$.

Now suppose that h_0 does not divide b_1. This implies that h_0 also does not divide $r = \gcd(f, b_1)$, where r can be assumed to be monic. But then $\tilde{h} \bmod (p^k, H_k)$ divides $(\tilde{f}/\tilde{r}) \bmod (p^k, H_k)$, so that Proposition 3.6 can be applied with f replaced by f/r. Conditions (3.8) and (3.9) are satisfied because $(f/r)_{\max} \leqq B_f$ (cf. (2.7)) and because of (3.16), (3.17) and (3.15). It follows that $\gcd(f/r, b_1) \neq 1$, which contradicts $r = \gcd(f, b_1)$ because f does not contain multiple factors. □

(3.19) We describe how to compute the irreducible factor h_0 of f. Suppose that f does not contain multiple factors, and that the polynomial $\tilde{h}$, the $(t-1)$-tuple $s_2, s_3, \cdots, s_t$, and the prime power p^k are chosen such that (3.1), (3.2), (3.3), (3.4), (3.16) and (3.17) are satisfied with, for (3.16) and (3.17), m replaced by $n-1$. Remember that we also have to take care that conditions (2.1), (2.2), (2.3), (2.4) and (2.5) on p and H are satisfied.

We apply the basis reduction algorithm (cf. [9, § 1]) to a sequence of M_j-dimensional lattices as in (3.5), where the $M_j = 1 + I \sum_{i=1}^{t} m_i N_{i+1}$ run through the range of admissible values for $m_1, m_2, \cdots, m_t$ (cf. (3.5)), in such a way that $M_j < M_{j+1}$. (So, for $m = l, l+1, \cdots, n-1$, and $m_i = 0, 1, \cdots, \delta_i l c_{i-1}(f)$ for $i = t, t-1, \cdots, 2$ in succession.) According to (3.14), the first vector b_1 that we find that satisfies (3.18) equals $\pm h_0$ (remember that b_1 belongs to a basis for the lattice), so that we can stop if such a vector is found. If for none of the lattices a vector satisfying (3.18) is found, then h_0 is not contained in any of these lattices according to (3.14), so that $h_0 = f$.

PROPOSITION 3.20. *Assume that the conditions in* (3.19) *are satisfied. The polynomial h_0 can be computed in $O((\delta_1 h_0 I N_2)^4 k \log p)$ arithmetic operations on integers having binary length $O(INk \log p)$.*

Proof. Observing that $\log(INp^{2k}) = O(k \log p)$ (cf. (3.17), (3.15) and (2.8)), the proof immediately follows from (3.19), (3.5) and [9, (1.26), (1.37)]. □

(3.21) We now show how $s_2, s_3, \cdots, s_t$ and p can be chosen in such a way that the conditions in (3.19) can be satisfied. The algorithm to factor f then easily follows by repeated application of (3.19).

We assume that f does not contain multiple factors, so that the resultant $R = R(df, df')$ of df and its derivative df' with respect to X_1 is unequal to zero. First we choose $s_2, s_3, \cdots, s_t \in \mathbf{Z}_{>0}$ minimal such that (3.16) is satisfied with m replaced by $n-1$. It follows from (3.16), (3.15), (2.8) and $\log D = O(\log d + I \log(I|F|))$ (because $D = d|\operatorname{discr}(F)|$), that

$$\begin{aligned} \log s_j &= O(\log((n+m)n_j) + \log B_{j-1}) \\ &= O\left(InN + n\left(\log B_f + \log D + I \log(1 + F_{\max}) + \sum_{i=1}^{j-1} n_i \log s_i\right)\right) \end{aligned}$$

$$= O\left(n\left(IN + \log(df_{\max}) + I\log(I|F|) + \sum_{i=1}^{j-1} n_i \log s_i\right)\right)$$

for $2 \leqq j \leqq t$, so that

$$\log s_j = O\left(n(IN + \log(df_{\max}) + I\log(I|F|)) \prod_{i=2}^{j-1}(1+nn_i)\right)$$

and

$$\sum_{i=2}^{t} n_i \log s_i = O(n^{t-2}N(IN + \log(df_{\max}) + I\log(I|F|))). \tag{3.22}$$

From the proof of (3.6) it follows that, for this choice of $s_2, s_3, \cdots, s_t$ the resultant $R \in \mathbf{Z}[\alpha]$ of $d\tilde{f}$ and $d\tilde{f}'$ is unequal to zero.

Next we choose p minimal such that p does not divide D or discr (F), and such that $\tilde{R} \not\equiv 0 \bmod p$. Clearly

$$\prod_{q \text{ Prime}, q<p} q \leqq d \operatorname{discr}(F)\tilde{R}_{\max}$$

which yields, together with

$$\prod_{q \text{ prime}, q<p} q > e^{Ap}$$

for all $p > 2$ and some constant $A > 0$ (cf. [5, § 22.2]), that

$$p = O(\log d + I\log(I|F|) + \log \tilde{R}_{\max}). \tag{3.23}$$

Similar to (3.13) we obtain

$$\tilde{R}_{\max} \leqq f_{\max}^{2n-1} n^n (2n-1)! \left(dN_2 \prod_{i=2}^{t} s_i^{n_i}\right)^{2n-1} (1+F_{\max})^{(I-1)(2n-2)},$$

so that we get, using (3.22)

$$\log \tilde{R}_{\max} = O(n^{t-1}N(IN + \log(df_{\max}) + I\log(I|F|))).$$

Combining this with (3.23) we conclude that

$$p = O(n^{t-1}N(IN + \log(df_{\max}) + I\log(I|F|))). \tag{3.24}$$

Notice that (2.1) is now satisfied. In order to compute a polynomial $H \in \mathbf{Z}[T]$ satisfying (2.2), (2.4), (2.5) and (2.3) with k replaced by 1, we factor $F \bmod p$ by means of Berlekamp's algorithm [7, § 4.6.2] and we choose H as an irreducible factor of $f \bmod p$ for which $\tilde{R} \bmod (p, H_1) \neq 0$; such a polynomial H exists because $\tilde{R} \bmod p \neq 0$. Conditions (2.4) and (2.3) with k replaced by 1 are clear from the construction of H, and because we may assume that H has leading coefficient equal to one, (2.2) also holds. The condition that discr $(F) \bmod p \neq 0$, finally, guarantees that $F \bmod p$ does not contain multiple factors, so that (2.5) is satisfied.

We choose k minimal such that (3.17) holds, so that

$$k \log p = O\left(I\left(InN + n\log(df_{\max}) + In\log(I|F|) + n\sum_{i=2}^{t} n_i \log s_i\right) + \log p\right)$$

(cf. (3.15) and (2.8)), which gives, with (3.22) and (3.24),

$$k \log p = O(In^{t-1}N(IN + \log(df_{\max}) + I\log(I|F|))). \tag{3.25}$$

Now we apply Hensel's lemma [7, Exer. 4.6.22] to modify H in such a way that (2.3) holds for this value of k (this is possible because (2.3) already holds for $k = 1$), and finally we apply Berlekamp's algorithm as described in [1, § 5] and Hensel's lemma as in [17] to compute the irreducible factorization of $\tilde{f} \bmod (p^k, H_k)$ in $W_k(\mathbf{F}_q)[X_1]$.

Condition (3.4) is satisfied for each irreducible factor $\tilde{h} \bmod (p^k, H_k)$ of $\tilde{f} \bmod (p^k, H_k)$ because $\tilde{R} \bmod (p, H_1) \neq 0$, and (3.1), (3.2) and (3.3) are clear from the construction of $\tilde{h}$.

We have shown how to choose $s_2, s_3, \cdots, s_t$ and p, and how to satisfy the conditions in (3.19). We are now ready for our theorem.

THEOREM 3.26. *Let f be a monic polynomial in $(1/d)\mathbf{Z}[\alpha][X_1, X_2, \cdots, X_t]$ with $t \geqq 2$, of degree n_i in X_i and $2 \leqq n = n_1 \leqq n_2 \leqq \cdots \leqq n_t$. The irreducible factorization of f can be found in $O(n^{t-1}(IN)^5(IN + \log(d f_{\max}) + I \log(I|F|)))$ arithmetic operations on integers having binary length $O(n^{t-1}(IN)^2(IN + \log(d f_{\max}) + I \log(I|F|)))$, where $N = \prod_{i=1}^{t}(n_i + 1)$.*

Proof. If f does not contain multiple factors, then f can be factored by repeated application of (3.19). In that case (3.26) follows from (3.21), (3.20), (3.25) and the well-known estimates for the application of Berlekamp's algorithm and Hensel's lemma (cf. [1], [7] and [19]).

If f contains multiple factors, then we first have to compute the monic gcd g of f and its derivative with respect to X_1, and the factoring algorithm is then applied to f/g. The cost of factoring f/g satisfies the same estimates as above, because $(f/g)_{\max} \leqq B_f$ (cf. (2.7)), and this dominates the costs of the computation of g, which can be done by means of the subresultant algorithm (cf. [2]). □

REFERENCES

[1] E. R. BERLEKAMP, *Factoring polynomials over large finite fields*, Math. Comp., 24 (1970), pp. 713–735.

[2] W. S. BROWN, *The subresultant* PRS *algorithm*, ACM Trans. Math. Software, 4 (1978), pp. 237–249.

[3] A. L. CHISTOV AND D. YU. GRIGORYEV, *Polynomial-time factoring of the multivariable polynomials over a global field*, LOMI preprint E-5-82, Leningrad, 1982.

[4] A. O. GEL'FOND, *Transcendental and Algebraic Numbers*, Dover, New York, 1960.

[5] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Oxford University Press, London, 1979.

[6] E. KALTOFEN, *Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization*, this Journal, 14 (1985), pp. 469–489.

[7] D. E. KNUTH, *The Art of Computer Programming, Vol.* 2, *Seminumerical Algorithms*, sec. ed., Addison-Wesley, Reading, MA, 1981.

[8] A. K. LENSTRA, *Lattices and factorization of polynomials over algebraic number fields*, Proc. Eurocam 82, LNCS, 144 (1982), pp. 32–39.

[9] A. K. LENSTRA, H. W. LENSTRA, JR AND L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[10] A. K. LENSTRA, *Factoring polynomials over algebraic number fields*, Report IW 213/82, Mathematisch Centrum, Amsterdam, 1982.

[11] ———, *Factoring multivariate polynomials over finite fields*, J. Comput. System Sci., 30 (1985), pp. 235–248; Proc. 15th Annual ACM Symp. on the Theory of Computing, pp. 189–192.

[12] ———, *Factoring multivariate integral polynomials*, Theoretical Computer Science, 34 (1984), pp. 207–213; Proc. 10th ICALP, LNCS 154, pp. 458–465.

[13] ———, *Factoring multivariate integral polynomials*, II, Report IW 230/83, Mathematisch Centrum, Amsterdam, 1983.

[14] M. MIGNOTTE, *An inequality about factors of polynomials*, Math. Comp., 28 (1974), pp. 1153–1157.

[15] J. STOER, *Einführung in die numerische Mathematik* I, Springer, Berlin, 1972.

[16] J. VON ZUR GATHEN, *Factoring sparse multivariate polynomials*, Proc. 24th Annual Symposium on Foundations of Computer Science (1983), pp. 172–179.

[17] P. S. WANG, *Factoring multivariate polynomials over algebraic number fields*, Math. Comp., 30 (1976), pp. 324–336.

[18] P. J. WEINBERGER AND L. P. ROTHSCHILD, *Factoring polynomials over algebraic number fields*, ACM Trans. Math. Software, 2 (1976), pp. 335–350.

[19] D. Y. Y. YUN, *The Hensel Lemma in Algebraic Manipulation*, MIT, Cambridge, MA, 1974; reprint: Garland, New York, 1980.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics

OPTIMAL MIXED GRAPH AUGMENTATION*

DAN GUSFIELD†

Abstract. In this paper we consider an augmentation problem on mixed graphs that generalizes and unifies two augmentation problems considered by Eswaran and Tarjan [this Journal, 5 (1976), pp. 653-665]. The mixed augmentation problem has applications in the design of communication networks, and forms of the mixed augmentation problem are central in statistical data security problems discussed in [Yale Univ. Tech. Report, August 1984]. We solve the augmentation problem on mixed graphs in time linear in the number of edges of the graph, the same time bound obtained in [ET] for each of the two special cases.

Key words. graph augmentation, strong connectivity, graph orientation

AMS(MOS) subject classifications. 05C40, 05C20, 68Q25, 68R10

1. Introduction. In this paper we consider an augmentation problem on mixed graphs that generalizes and unifies two augmentation problems considered by Eswaran and Tarjan [ET]. The mixed augmentation problem has applications in the design of communication networks, and forms of the mixed augmentation problem are central in statistical data security problems discussed in [G].

Let G be a connected undirected graph and G' a connected directed graph. The following two problems are solved with linear time algorithms in [ET]: add the fewest number of *undirected* edges to G so that the resulting graph contains no bridges, and add the fewest number of *directed* edges to G' so that the resulting graph is strongly connected. The first problem is equivalent to the problem of adding the fewest number of undirected edges to G so that every edge is in an undirected simple cycle, and the second problem is equivalent to the problem of adding the fewest number of directed edges so that every edge is in a directed simple cycle. In a *mixed* graph (one with both undirected and directed edges) we say that a cycle is *traversable* if and only if it is possible to walk around the cycle without walking opposite to the direction of any directed edges on the cycle. Clearly an undirected or a directed cycle is traversable. It will follow from Theorem 1 below that the first problem is also equivalent to that of adding the fewest number of *directed* edges to the undirected graph G so that in the resulting *mixed* graph every edge is in a traversable simple cycle. The second problem can obviously also be similarly described, i.e., as the problem of adding the fewest directed edges to the directed graph G' so that every edge is in a traversable simple cycle. Hence a common generalization of these two augmentation problems is that of adding the fewest number of *directed* edges to a *mixed* graph so that every edge in the resulting graph is in a traversable simple cycle. We call this the *mixed graph augmentation* problem; we will give a linear time algorithm to solve this problem. We also note that the mixed graph augmentation problem is equivalent to the problem of adding the minimum number of undirected edges to a mixed graph to make the resulting graph strongly connected and bridgeless.

The mixed graph augmentation problem is of interest not only because it generalizes the strictly undirected and the strictly directed augmentation problems, but because its solution unifies the separate solutions presented for these two problems in [ET]. The mixed problem is also central in the study of data security [G] where a data

* Received by the editors April 15, 1985; accepted for publication (in revised form) August 8, 1986. This work was partially funded by the National Science Foundation under grant MCS-81/05894.

† Computer Science Division, University of California, Davis, California 95616.

protection problem reduces to a special case of the mixed augmentation problem. In fact, the work reported here results from an effort to solve the most general form of the protection problem. In § 3 we will briefly discuss this problem and show how the results of this paper efficiently solve a special case of it.

As a final application for the mixed augmentation problem consider the task of completing or upgrading a partially constructed communication system: one-way communication lines are to be laid out so that every point in the system can send data to any other point, i.e., the graph representing the system must be made strongly connected. Input for the problem consists of a connected[1] mixed graph G and a directed graph $\bar{G}$. The directed edges in G represent communication lines that already exist, the undirected edges in G represent existent conduits or right of way along which one-way lines can be (cheaply) added, and the edges in $\bar{G}$ represent one-way lines requiring new (expensive) conduits or right of way. In the most general problem there is a distinct cost to route a line in each conduit and a distinct cost to build each new conduit, and the objective is to choose the cheapest way to use the existing conduits and to add more, to create a directed, strongly connected communication system. This problem is unsolved, and apparently hard. However, if the cost of building each new conduit is roughly the same, and much greater than that of routing line in an existing conduit, then the following simpler problem is of interest: minimize the number of new conduits needed so that one-directional lines can be laid to make the graph strongly connected. We call this the *communication completion* problem and we will see that this is exactly the mixed augmentation problem.

1.1. Definitions and problem specification.

DEFINITION 1.1. Let $G=(V, E)$ be a connected *mixed* graph with node set V and edge set E, where E can contain both directed and undirected edges. Let $\bar{G}=(V, \bar{E})$ be a *directed* graph on the same nodes.

DEFINITION 1.2. We use the notation $\langle i, j\rangle$ to indicate a directed edge from i to j, and (i, j) to denote an undirected edge between nodes i and j. When the distinction does not matter, we will use the unmodified term "edge" to refer to either a directed or an undirected edge. Note that the term "mixed graph" permits the special cases that all the edges of G are directed or are all undirected.

DEFINITION 1.3. A cycle C in G is called *traversable* if and only if it is possible to walk around it without ever walking against the direction of any directed edge on C. Similarly, a path P is traversable if it is possible to walk from its start to finish nodes without walking opposite to the direction of any edges on P. We call an edge e (directed or undirected) in G *traversable* if and only if e lies on some traversable cycle in G, and we call a mixed graph traversable if and only if all its edges are traversable.

Note that a cycle consisting of the two directed edges $\langle i, j\rangle$ and $\langle j, i\rangle$ is considered a simple cycle, as is one consisting of the undirected edge (i, j) and, say, the directed edge $\langle j, i\rangle$, but the path i, j, i that traverses a single undirected edge (i, j) in both directions is not a simple cycle.

DEFINITION 1.4. The *mixed augmentation problem* is to find a minimum number of edges of $\bar{E}-E$, such that when these edges are added to G, every edge in E is traversable.[2] Let $M(G)$ denote the size of the smallest such set of new edges, and define $M(G)=\infty$ if there are edges of G that are not traversable even when all edges

[1] We assume G is connected for simplicity. We can extend the ideas here to the case of G disconnected.

[2] If the solution is optimal then all the new edges will also be traversable.

of $\bar{G}$ are added to G. In the case that G is undirected (directed), the mixed augmentation problem is called the undirected (directed) augmentation problem.

DEFINITION 1.5. A mixed graph G is *strongly connected* if for every ordered pair of nodes (u, v) in G, there is a traversable path from u to v.

Fact 0. A connected directed graph is traversable if and only if it is strongly connected. A connected undirected graph is traversable if and only if it has no bridges, or equivalently, if and only if it can be directed to become a strongly connected directed graph (see [O] for a frivolous application of this). A mixed graph G can be directed to become a strongly connected directed graph, if and only if G is strongly connected and has no bridges [BT].

It will follow from Theorem 1 below that a connected mixed graph G is traversable if and only if its undirected edges can be directed to become a strongly connected directed graph, hence if and only if G is strongly connected and has no bridges.

When $\bar{G}$ is permitted to be an arbitrary directed graph on node set V, the directed and undirected augmentation problems are *NP*-complete [ET], [FJ]. Tarjan and Eswaran [ET] present two linear time algorithms to solve the undirected and directed problems[3] with the assumption that $\bar{G} = (V, \bar{E})$ is a complete directed graph, i.e., $\bar{E}$ contains the edges $\langle i, j\rangle$ and $\langle j, i\rangle$ for every pair of nodes i, j. We will also make this assumption and solve the resulting mixed augmentation problem.

Two recent papers are related to the mixed augmentation problem presented here. Chung, Garey and Tarjan [CGT] discuss the problem of efficiently orienting the undirected edges of a bridgeless, strongly connected mixed graph to produce a strongly connected directed graph with small diameter. The relationship of the mixed augmentation problem to orientation problems will be made clear in this paper. Kajitani and Ueno [KU] discuss the problem of augmenting a directed tree to become a k-edge-connected directed graph.

2. Linear time mixed augmentation. In this section we assume, as in [ET], that $\bar{G}$ is a complete directed graph, and give a linear time algorithm for the mixed augmentation problem. The method is presented as a sequence of linear time reductions from the mixed augmentation problem on general mixed graphs to simpler intermediate problems, resulting finally in the directed augmentation problem whose linear time solution is given in [ET]. At the end of this section we put the reductions together to form a linear time algorithm solving the mixed augmentation problem. Before presenting the reductions, we need additional discussion of the relation of augmentation to orientation.

2.1. Augmentation and orientation.

DEFINITION 2.1. Given mixed graph G, let G' denote a directed graph obtained by some orientation of the undirected edges of G.

Clearly, $M(G) \leqq M(G')$, but Theorem 1 below shows that equality holds for some G'.

THEOREM 1. *There exists a directed graph G' resulting from orienting the undirected edges of G, such that $M(G) = M(G')$.*

This theorem is very useful and somewhat surprising, since the definition of "traversable cycle" permits the use of an undirected edge in opposite directions on two different traversable cycles. The proof of the theorem will use the concept of a

[3] In [ET] graph G is not assumed to be connected, but the graph after augmentation is required to be connected. We can similarly modify the techniques here to include this case, but the exposition is simplified without it. The reader should be alerted that in [ET] the most critical implementation details (algorithm *ST*) of the method for directed augmentation are in error, but have a simple correction.

circulation. Intuitively, a circulation f is a flow in a (mixed) graph without sink or source nodes, i.e., where the conservation condition holds at *every* node. A formal definition follows.

DEFINITION 2.2. A *flow function* f is a function which assigns a real number to each *ordered* pair of nodes i, j in a mixed graph such that $f(i,j)$ is nonnegative if $\langle i,j\rangle$ is a directed edge from i to j, $f(i,j)$ is zero if there is no edge between i and j, and for every pair of nodes i, j, $f(i,j) = -f(j,i)$.

The interpretation of f is that $f(i,j)$ is positive if flow moves from node i to node j, and negative if flow moves from node j to node i. In the case that (i,j) is an undirected edge, this definition allows $f(i,j)$ (the flow *from i to j*) to be either positive, negative or zero, but in any case, $f(i,j) = -f(j,i)$.

DEFINITION 2.3. The flow function f is a circulation if and only if $\sum_j f(i,j) = 0$ for every node i, i.e., the flow into i equals the flow out of i.

DEFINITION 2.4. If f and g are two circulations, then the *addition* of f and g is the function $s(i,j) = f(i,j) + g(i,j)$. Note that by the use of negative f and g values, flows in opposing directions on an undirected edge will (partially) cancel, so that the magnitude of $s(i,j)$ may be smaller than that of either $f(i,j)$ or $g(i,j)$. The following facts are well known and easy to prove [FF].

Fact 1. The addition of two circulations is a circulation.

Fact 2. Given a circulation f, let X be the set of undirected edges (i,j) such that $f(i,j) > 0$, and let Y be the set of directed edges such that $f(i,j) > 0$. If each edge of X is directed from i to j, then every edge in $X \cup Y$ is contained in some simple directed cycle in $X \cup Y$.

Now we can prove Theorem 1.

Proof of Theorem 1. Starting with an optimal solution to the mixed augmentation problem, we will find a way to orient the undirected edges of G so that every edge is in some directed simple cycle in the optimal solution; this orientation defines G' and proves the theorem.

If each edge is traversable, then there exists a set of traversable cycles that contains every edge of G. Let S be such a set of cycles extracted from the optimal solution to the mixed problem. If each undirected edge is used in only one direction in S, then we direct the undirected edges in the direction used in S, and the theorem is proved. If this is not the case, we will use S to find a set of traversable simple cycles S' with the above property. To begin, order the cycles in S arbitrarily, and call them $C(1)$, $C(2), \cdots, C(k)$.

Let f_C be a circulation associated with cycle C, where f_c is defined by the direction of traversal of C (for a cycle C composed only of undirected edges, pick a direction of traversal arbitrarily). In particular, $f_C(i,j) = 1$ if the traversal of C moves from i to j along an edge (directed or undirected) between i and j, and $f_C(i,j) = 0$ if i and j are not connected by an edge on C. To get the idea of the proof, let F be the addition of circulations $f_{C(1)}$ through $f_{C(k)}$. By Fact 1, F is a circulation and by Fact 2, the undirected edges with nonzero flow can be directed so that all the nonzero edges in F are contained in a set of simple directed cycles. If every edge of G has nonzero flow in F, then we are done. Although no *directed* edge of G will have zero flow in F, an *undirected* edge will have zero flow if the total flows on it in opposite directions exactly cancel in F. To overcome this problem, we will use S to find a circulation in which every edge of G has a nonzero flow. Given a circulation f, define $2f$ to be the addition of f with itself. Circulation $F(k)$ is constructed as follows.

(1) Set $F(0)$ to the circulation $F(i,j) = 0$ for every ordered pair i, j.

(2) For $i = 1$ through k set $F(i)$ to $2F(i-1) + f_{C(i)}$.

We claim that $F(i)$ is a circulation with nonzero flow on every edge in the union of edges in cycles $C(1)$ through $C(i)$. This follows easily by induction on i. Clearly it is true for $i=1$. Now consider $F(2)=2F(1)+f_{C(2)}=2f_{C(1)}+f_{C(2)}$. Let (i,j) be an edge in $C(1)\cap C(2)$. If the direction of flow on (i,j) is the same in circulations $f_{C(1)}$ and $f_{C(2)}$, then in $F(2)$, edge (i,j) has flow of value three in that same direction. If the direction of flow on (i,j) in $f_{C(1)}$ is opposite to the direction of flow in $f_{C(2)}$, then in $F(2)$ there is one unit of flow in the direction of flow on (i,j) in $f_{C(1)}$. Clearly for any other edge in $C(1)\cup C(2)$, the flow in $F(2)$ is at least one. Intuitively, the purpose of adding $f_{C(1)}$ to itself before addition to $f_{C(2)}$ is to guarantee that in the resulting circulation, every edge in $C(1)\cup C(2)$ has nonzero flow. The same argument is used in the general induction step, and hence $F(k)$ is a circulation in which all edges of G have nonzero flow. Then, by Fact 2, Theorem 1 is proved. □

COROLLARY 1. *The undirected augmentation problem for connected graphs treated in* [ET] *is equivalent to the undirected problem defined in this paper, i.e. there is no loss of optimality in adding directed edges to a connected undirected graph so that every edge is traversable, instead of adding undirected edges so that every edge is in a cycle.*

COROLLARY 2. *A connected mixed graph is traversable if and only if its undirected edges can be directed so that the resulting directed graph is strongly connected. This follows from Theorem* 1 *and Fact* 0.

The proof of Theorem 1 is not intended as an efficient orientation algorithm; however, an appropriate orientation can be found in linear time. This follows from Corollary 2, and a recent paper by Chung, Garey and Tarjan [CGT] which gives a linear time algorithm to direct the undirected edges so that the resulting directed graph is strongly connected, when it is possible to do so. Hence after the (linear) time needed to find the optimal augmentation, the additional orientations can be done in linear time.

2.1.1. The communication completion problem. Corollary 2 implies that the communication completion problem is equivalent to the mixed augmentation problem. Recall that in the communication completion problem we add the fewest directed edges, and direct undirected edges so that the resulting directed graph is strongly connected. In the mixed augmentation problem we add the fewest directed edges so that the resulting mixed graph is traversable, and hence, by Corollary 2, it can be directed, in linear time by [CGT] to become strongly connected. So a solution to the mixed augmentation problem is a solution to the communication completion problem. Conversely, any solution to the communication completion problem is clearly a solution to the mixed graph augmentation problem, since any directed edge in a strong component is in a directed cycle. By Theorem 1, it is also clear that the communication completion problem, where new undirected, instead of directed, edges are permitted, does not reduce the cost of the solution.

2.2. The acyclic augmentation problem: reduction to *CG*. Given Theorem 1, we can solve the mixed augmentation problem for G if we can efficiently find the orientation of the undirected edges of G which leads to the best solution of the resulting directed augmentation problem. The cost of the optimal solution for the directed augmentation problem (when, as assumed, $\bar{G}$ is a complete directed graph) has a simple form, but it is not clear at this point how to optimally orient G. In order to be able to optimally orient, we first reduce the mixed augmentation problem to the augmentation problem on a mixed acyclic graph *CG*. We first need some definitions.

DEFINITION 2.5. In a directed graph G we call a node an *in-node* if all edges incident with it are directed into it, and call it an *out-node* if all incident edges are directed out of it; the remaining nodes are called *through* nodes.

DEFINITION 2.6. Let $I(G)$ and $O(G)$ be respectively the number in- and out-nodes of mixed graph G.

DEFINITION 2.7. Let H be a subgraph of $G=(V, E)$. The graph resulting from *condensing H in G* is a graph with node set V plus new node h, minus the nodes of H; all edges in G with both endpoints in H are removed; for any edge (u, v) where u is in H and v is not in H, edge (u, v) is replaced by (h, v), and any directed edges $\langle u, v\rangle$ or $\langle v, u\rangle$ are similarly replaced by $\langle h, v\rangle$ or $\langle v, h\rangle$.

DEFINITION 2.8. For a *directed* graph G', let SG' be the acyclic directed graph resulting from condensing each strong component in G' to a single node.

It is shown in [ET] that for a directed graph G', $M(G')=\max[I(SG'), O(SG')]$. Hence the mixed augmentation problem reduces to the problem of orienting the undirected edges of G, forming directed graph G', in a way that minimizes $\max[I(SG'), O(SG')]$. What makes this problem difficult at this point is the indirect way (due to the intermediate step of condensing G' to SG') that choices about orienting undirected edges of G affect the number of in- and out-nodes in the resulting SG'. To attack the problem more directly, we would like to reverse the order of the operations, i.e., condense first and then orient. We can do this using graph CG.

2.2.1. Graph CG. We construct the acyclic mixed graph CG as follows:

Construction of CG.

(1) Find the strongly connected components of the mixed graph G, and let A be the set of edges not in any strong components. Clearly, A consists only of directed edges.

(2) Consider each of the strong components as a separate undirected graph, ignoring the directions on any directed edges. Let B be the set of bridges in these components; clearly B consists only of undirected edges, since any directed edge in one of the strong components is in a cycle inside the strong component itself.

(3) Let K be the graph induced by the edges of G minus $A \cup B$, and let $K(1), \cdots, K(r)$ be the connected components of K. Graph CG is formed by condensing each $K(i)$ in G. Hence each node in CG is in 1-1 correspondence with some $K(i)$, and the edges of CG are exactly $A \cup B$.

Figure 1 shows an example construction of CG from G. Note that it is easy to combine steps 1 and 2 into a single algorithm, using roughly half the time needed for the two separate steps. We leave this to the reader.

We make the following claims.

LEMMA 1. *For every ordered pair of nodes u, v in any $K(i)$, there is a traversable simple path from u to v in $K(i)$, and every edge in $K(i)$ is in a traversable simple cycle in $K(i)$.*

Proof. Each $K(i)$ is a subset of a strong component χ found in step 1 above, and hence there is a traversable simple path P from u to v in χ. But if P traverses an edge in B, then it must traverse it twice, in which case P could not be simple. Hence P is strictly contained in $K(i)$, and the first part of the lemma is proved. Now any directed edge $\langle u, v\rangle$ in $K(i)$ is in the traversable simple cycle in $K(i)$ consisting of edge $\langle u, v\rangle$ followed by a traversable simple path P from v to u, so the second claim for directed edges is proved. The situation is a little more complicated for an undirected edge (u, v) in $K(i)$. Since (u, v) is not a bridge (with respect to step 2) in χ, there must be a simple path P in $K(i)$ connecting (ignoring the directions of the edges on P) node u and node v. If P is traversable, we are done. If P is not traversable, let $\langle x, y\rangle$ be the first directed edge on P such that the walk from u to v on P goes in the wrong direction, i.e., from y to x. Let P' be a traversable simple path from y to x in $K(i)$. If P' contains (u, v), then P' followed by $\langle x, y\rangle$ is a traversable simple cycle in $K(i)$ containing (u, v).

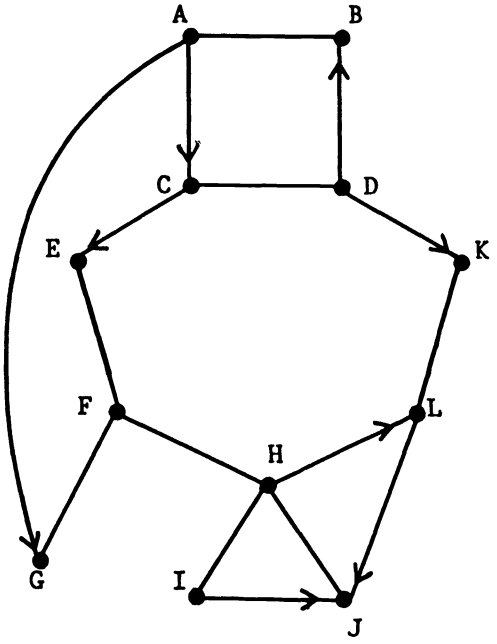


FIG. 1(a). *Mixed graph G.*

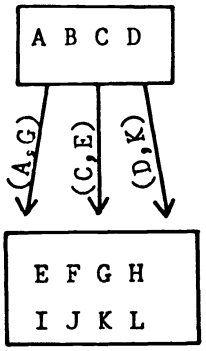


FIG. 1(b). *Strong components of G and the edge set A.*

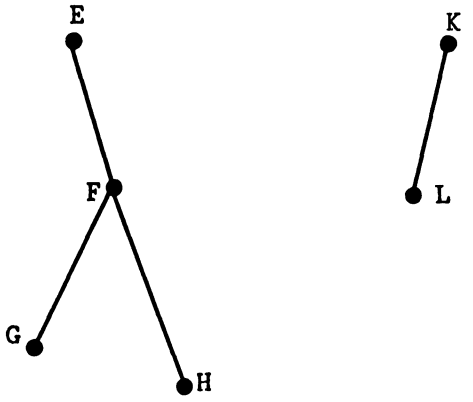


FIG. 1(c). *Undirected bridges inside strong component EFGH. IJKL*

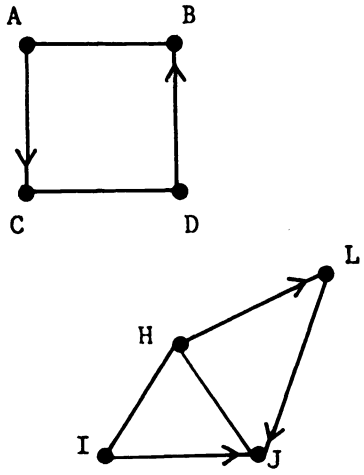


FIG. 1(d). *Graph K.*

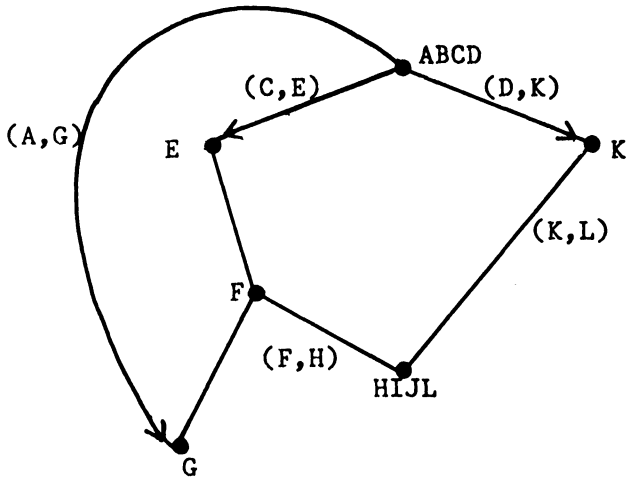


FIG. 1(e). *Graph CG. The Augmentation problem for G can be solved by adding one edge. There are many ways to do it. For example from K to B or from F to C.*

If P' does not contain (u, v) then the path consisting of P to y, P' to x, and P to v is a path which traverses one less edge in the wrong direction than does P. Hence by repeating this argument, a traversable cycle in $K(i)$ containing (u, v) is found, and from that a traversable simple cycle containing (u, v) can be extracted. □

LEMMA 2. *The optimal augmentation for CG is optimal for G.*

Proof. Clearly, any augmentation for G defines an augmentation for CG at equal or less cost since CG is derived from G by condensing nodes of G. So $M(CG) \leqq M(G)$. Conversely, any augmentation for CG is an augmentation for G since every edge in any $K(i)$ is in a traversable cycle in G, and further, there is a traversable path between any ordered pair of nodes in $K(i)$. Hence any augmentation of CG is an augmentation for G, and $M(G) \leqq M(CG)$. □

Hence the mixed augmentation problem on G reduces to the mixed augmentation problem on acyclic graph CG.

2.2.2. Advantage of CG. We have reduced the augmentation problem on G to the augmentation problem on the acyclic graph CG, but it is not clear that we have made any progress. We saw earlier that the optimal augmentation of G can be found by directing the undirected edges of G, forming directed graph G', to minimize $\max[I(SG'), O(SG')]$, but that it was difficult to choose the best G' because of the intermediate step of condensing G' to SG'. This difficulty is overcome with the use of CG: CG has no traversable cycles, so if CG' is *any* orientation of CG, then there are no edges of CG' in strong components, i.e., $SCG' = CG'$. Hence, $M(CG') = \text{Max}[I(CG'), O(CG')]$. Hence the mixed graph augmentation problem for G reduces to the problem of orienting the undirected edges of CG to minimize the maximum of the number of in- and out-nodes in the resulting directed graph.

2.3. Optimal acyclic mixed graph orientation. In this section we develop the ideas needed to find an optimal orientation of CG. In particular, we look closely at how the directed edges of CG touch the undirected trees of CG, and how this affects the optimal solution. Let T be one of the trees in the undirected forest of CG.

DEFINITION 2.9. We call an edge not in T an *in-edge* if it is directed into T and an *out-edge* if it is directed out of T.

DEFINITION 2.10. A leaf of T which is incident with no directed edges of CG is called an *isolated leaf*.

LEMMA 3. *We can reduce the orientation problem of CG to one where each leaf in the undirected trees is either isolated, or is adjacent to both an in and an out edge.*

Proof. Consider a leaf v incident with in-edges only. We can orient the undirected tree edge incident with v away from v, making v a through node in any full orientation of T. This orientation preserves optimality, since if the edge is oriented the other way in the optimal orientation, then v is an in-node and reversing the direction of the edge will not increase the number of in- or out-nodes. We can continue such orientations until no leaf (in the trees formed by the edges that remain undirected) is incident with in-edges only. For a leaf v incident with out-edges only, we direct the undirected edge towards v. □

2.3.1. Effect of edges incident on T. We assume that every tree T in CG has the reduced form specified in Lemma 3. We now examine how the number and location of directed edges incident on a tree T in CG affects the number of in- and out-nodes that can be generated by orienting the edges of T. The following lemma is fundamental in this analysis.

LEMMA 4. *Let T be an undirected tree with n isolated leaves, and let x and y be integers such that $x+y=n$, $x \geqq 1$, $y \geqq 1$. Then there exists an orientation of the edges of T, creating exactly x in-nodes and y out-nodes of T. Further, the constraint $x \geqq 1$ ($y \geqq 1$) can be changed to $x \geqq 0$ ($y \geqq 0$) if and only if there is an edge of $CG - T$ out of (into) a node of T.*

Proof. To show that there is an orientation satisfying the first relations, arbitrarily select an isolated leaf v, and y additional isolated leaves of T, and let T' be the unique smallest subtree of T connecting v and those y leaves (of course T' might have some interior nodes). Considering v as the root of T' orient the edges of T' so that there is a directed path from each leaf in T' to v. After this orientation all of the selected y leaves become out-nodes, v becomes an in-node, and every interior node of T' becomes a through node since it lies on one of the directed paths created by the orientation.

The remaining undirected edges of T form subtrees where in each such subtree, exactly one of the leaves, u, is an interior node of T', and so will become a through node in any further orientation of T. In each of these trees, orient the undirected edges so that there is a directed path from u to each of the leaves. In this way, all interior nodes of T, and all nonisolated leaves of T become through nodes, and the remaining $x-1$ isolated leaves of T become in-nodes.

When there is an edge of $CG-T$ directed out of a node u in T, we direct all paths from u towards the leaves of T, setting $x=0$, and $y=n$. It is easy to show that without such a node, $x \geqq 1$ is necessary; further, $x+y<n$ is clearly impossible. □

2.3.2. Undirected tree orientation. Given Lemma 4 we can now solve the orientation problem on CG. Suppose the undirected tree j of CG has $n(j)$ isolated leaves, and let $x(j)$ and $y(j)$ be variables associated with tree j. Then associated with tree j is the constraint $x(j)+y(j)=n(j)$, where $x(j)$ and $y(j)$ are each constrained either to be nonnegative or strictly positive, as detailed in Lemma 4. Recall that $I(CG)$ and $O(CG)$ are the number of in- and out-nodes that CG initially contains (some edges of CG are initially directed). Then the orientation problem is to minimize $\max [I(CG)+\sum_j [x(j)], O(CG)+\sum_j [y(j)]]$, where each $x(j)$, $y(j)$ satisfies the associated constraints for tree j.

For a simpler formulation of the problem, suppose, w.l.o.g., that $I(CG) \leqq O(CG)$ and define d as $O(CG)-I(CG)$; let $d(j)$ be called a *feasible difference* for the jth tree T if and only if $x(j)-y(j)=d(j)$ and $x(j)$, $y(j)$ satisfy the appropriate constraints for tree j. Then the orientation problem is to choose feasible differences to make $\sum_j d(j)$ as close to d as possible. This is a knapsack problem, but, unlike the general knapsack problem, a simple solution is known, since $d(j) \leqq n(j)$, and $\sum_j d(j)$ is bounded by the number of nodes of CG; hence this knapsack problem can be solved in polynomial time. In particular, if there are n nodes in CG, dynamic programming solves the problem in $O(n^2)$ time. However, we will show that this can be improved to linear time in the number of nodes. The following corollary of Lemma 4 is immediate, and is needed in the faster method.

COROLLARY 3. *If $n(j)$ is odd then both 1 and -1 are feasible values for $d(j)$; if $n(j)$ is even then 0 is a feasible value for $d(j)$; if $n(j)$ is even (odd) then $d(j)$ can take on all even (odd) values between its maximum and minimum possible values. Further, if $n(j)$ is even (odd), then $d(j)$ can take on only even (odd) values.*

2.4. Orienting CG in linear time.

DEFINITION 2.11. Let variable $d(j)$ be called an odd (even) variable if $n(j)$ is odd (even).

DEFINITION 2.12. For each tree j in CG, let $d(j)^*$ be the maximum value that $d(j)$ can take on. Clearly, $d(j)^*$ is either $n(j)$ or $n(j)-1$.

If $\sum_j d(j)^* \leqq d$, then the optimal solution is trivial: set $d(j)=d(j)^*$ for each j. If not, then we have the following theorem.

THEOREM 2. *Suppose $\sum_j d(j)^* > d$. If $\sum_j d(j)^*$ and d have the same parity, then values of the variables $d(j)$ can be selected to exactly add to d, and if they have different parity, then values can be found to add to $d-1$ or $d+1$, and this is optimal. These optimal values can be found in linear time, in fact, in a "greedy" manner.*

Proof. We will first examine the case when both d and $\sum_j d(j)^*$ are even; we will then reduce the three other cases to this case.

Consider any ordering of the set of variables, where the even variables appear before any odd variables. Relative to this ordering, let k be the smallest integer such that $\sum_{j=1}^{k-1} d(j)^* < d \leqq \sum_{j=1}^{k} d(j)^*$, and let I be the set of the first $k-1$ variables. Set

$d(j) = d(j)^*$ for each j in I. The proper settings of the remaining variables are detailed in the cases below.

If $d(k)^*$ is even, then set $d(k) = d - \sum_{j=1}^{k-1} d(j)^*$, which is possible by Corollary 3 and the fact that d, $\sum_{j=1}^{k-1} d(j)^*$, and $d - \sum_{j=1}^{k-1} d(j)^*$ are all even; for each other even variable, set $x(j)$ to zero, and for the odd variables, set half to 1, and half to -1. Since $\sum_j d(j)^*$ is even, there must be an even number of odd variables; hence the above $d(j)$ values sum exactly to d.

If $d(k)^*$ is odd, then there are two cases: either $\sum_{j=1}^{k-1} d(j)^*$ is odd, or it is even. If odd, then set $d(k) = d - \sum_{j=1}^{k-1} d(j)^*$, which by Corollary 3 is possible, since that difference is odd. Now since $\sum_{j=1}^{k-1} d(j)^*$ is odd, I must contain an odd number of odd variables, so k is even, and there are an even number of (odd) variables unset. For these remaining variables, set half to 1 and half to -1. Hence the variables sum exactly to d.

If $d(k)^*$ is odd, and $\sum_{j=1}^{k-1} d(j)^*$ is even, then I contains an even number of odd variables, and there are an even number of odd variables not in I. In this case, set $d(k) = d - \sum_{j=1}^{k-1} d(j)^* + 1$, and set $d(k+1) = -1$. There are an even number of odd variables left unset; set half to 1 and half to -1. Again, the values of the variables sum exactly to d.

Hence when d and $\sum_j d(j)^*$ are both even, feasible values of $d(j)$ can be found to sum exactly to d, which is optimal. We now reduce the remaining three cases to this case.

When d is odd and $\sum_j d(j)^*$ is even, then by the above analysis, values of the variables can be found to exactly sum to $d-1$, since $d-1$ is even.

When d is even and $\sum_j d(j)^*$ is odd, then consider the problem without one of the odd variables, say $d(t)$. In this new problem, d is even and $\sum_j d(j)^* - d(t)^*$ is even, so the optimal solution, of value d^* (which is either d or $\sum_j d(j)^* - d(t)$) can be found as above, and d^* is even. Now set $d(t)$ to $d - d^* + 1$, which is possible since $d - d^* + 1$ is odd, and $d(t)$ is an odd variable, and $\sum_j d(j)^* > d$.

When d and $\sum_j d(j)^*$ are both odd, then consider the problem with target value $d-1$, without the use of odd variable $d(t)$. In this problem, $d-1$ is even and $\sum_j d(j)^* - d(t)^*$ is even, so the optimal solution, of value d^* (which is either $d-1$ or $\sum_j d(j)^* - d(t)$) can be found as above, and d^* is again even. Now set $d(t)$ to $d - d^*$, which is possible since $\sum_j d(j)^* > d$, and $d - d^*$ is odd.

Hence in the cases when d and $\sum_j d(j)^*$ have the same parity, the variables sum to d exactly and in the case when they have opposite parity, the variables sum to one less or one more than d. In the latter case, this is the best possible since, by Corollary 3, any setting of the variables has the same parity as $\sum_j d(j)^*$. □

2.5. Summary of the mixed graph augmentation algorithm. We summarize the linear time solution to the mixed graph augmentation problem.

1. Given mixed graph G, find the acyclic mixed graph CG.

2. Find the undirected trees of CG, and working on the leaves of these trees, direct edges until the remaining trees conform to Lemma 3, i.e., each leaf is either isolated or is incident with both an in and an out edge.

3. Assuming $I(CG) \leqq O(CG)$, compute $d(j)^*$ for each tree j; $d(j)^*$ is either $n(j)$ or $n(j)-1$ as given in Lemma 4.

4. Set the optimal values of the variables $d(j)$, as detailed in Theorem 2. Each $d(j)$ value uniquely determines the values of $x(j)$ and $y(j)$.

5. Using the $x(j)$ and $y(j)$ given in step 4, direct tree j, as detailed in the proof of Lemma 4, so that the resulting tree has exactly $x(j)$ in-nodes and $y(j)$ out-nodes. With these orientations, graph CG contains now no undirected edges.

6. Using the linear time method in [ET], add the fewest number of directed edges to CG so that the resulting graph is strongly connected. Hence, by Corollary 2, G plus the added edges is traversable, and this optimally solves the mixed graph augmentation problem for G.

3. Additional applications. We have already shown that the communication completion problem is a mixed augmentation problem, and that the mixed augmentation problem is a proper generalization of the undirected and directed augmentation problems. In this section we briefly discuss some additional applications of the mixed augmentation problem and the methodology developed above. A full discussion requires details of the solutions for the directed and undirected augmentation problems presented in [ET]. We refer the reader to that paper, and omit some details and proofs in this section.

As a first application, consider the undirected augmentation problem. In this case, CG is a single undirected tree, and we can solve the undirected augmentation problem by first orienting CG, as in the proof of Lemma 4, with $x = \lceil n/2 \rceil$ and $y = \lfloor n/2 \rfloor$, and then using the algorithm in [ET] to solve the directed augmentation problem. This approach reduces the undirected problem to the directed problem, and we claim that this two step approach yields, as a special case, the algorithm in [ET] for undirected augmentation. The key point is that the solutions of both the mixed and the directed problems permit choices at several points, and the solution to the undirected problem given in [ET] can be derived by fixing particular choices. A proof of this is easy but is omitted as it requires details of the two algorithms.

As a second application, consider the following special form of the augmentation problem that is central in problems of statistical security [G]. Graph G contains at least three nodes, is connected, undirected and bipartite with node set R on one side of the graph, and node set C on the other. $\bar{G}$ is also bipartite with the same node sets, and for every $i \in R$ and $j \in C$, $\bar{G}$ contains the directed edges $\langle i, j \rangle$ and $\langle j, i \rangle$ if and only if (i, j) is not in G; $\bar{G}$ contains no other edges but these. Hence this special case is a mixed augmentation problem in which any added edge must run between an R node and a C node in G, and unlike the general mixed augmentation problem, no edge can be added between nodes adjacent in G. We call this form of the problem the *positive* problem. Details of the security problem and how it reduces to this augmentation problem are given in [G].

This problem at first seems harder than the case when $\bar{G}$ is a complete directed graph, since $\bar{G}$ here is so constrained. We will show, however, that the methodology so far developed in this paper can also optimally solve this problem in linear time. We will need the following fact.

Fact 3. The algorithm and implementation for the linear time solution in [ET] of the directed augmentation problem can be modified so that every added edge extends between an in-node and an out-node.

We now develop the solution to the positive problem. First, form (undirected, connected) graph CG from G as before, and give a leaf in CG a label R if it is an original R node in G, give it a label C if it is an original C node in G, and otherwise give it a label B indicating that it is a condensed node and therefore contains nodes of both labels. Now make CG a mixed graph CG' by directing any edge incident with an R leaf into the leaf, and directing any edge incident with a C leaf out of the leaf. In the resulting graph CG', all R leaves are in-nodes, all C leaves are out-nodes, all interior nodes are through-nodes, and edges incident with B leaves are undirected. Now solve the mixed augmentation problem on CG', i.e., where node labels are ignored,

and it is permitted to add any edge between two nodes in CG'. Let Z be the set of edges added in the optimal solution to the mixed problem on CG'. We will now use Z to derive an optimal solution to the positive problem on G.

We first discuss how to obtain a solution to the positive problem, and then discuss optimality. If Z contains no edges between adjacent nodes in CG, and contains no edges that touch a B node in CG, then Z itself clearly is a solution of the positive problem, i.e., it makes every edge in G traversable and contains no illegal edges. If Z does contain an edge, $e=(a, b)$, where a and b are adjacent nodes in CG; then, by Fact 3 one endpoint of e must be an in-node and one an out-node of CG'. But this can only happen if both endpoints of e are leaves in CG, and since G is connected and contains more than two nodes, this would imply that one of the endpoints of e is a B node in CG. We assume w.l.o.g., that b is a B node and a is an R node in CG.

Now each B node in CG represents at least four nodes of G, so two of these nodes are C nodes in G. Further, only one node of the nodes represented by b is adjacent with node a in G (otherwise e would not be a bridge and so would not be in CG). Hence, of the nodes represented by b, there is a C node, x, which is not adjacent to node a in G, and therefore an edge between x and a is legal. We add the edge (x, a) to G with the same orientation that e has. The set of edges produced in this way clearly makes G traversable, and hence is a legal solution to the positive problem.

To show that the solution is optimal, note first that the number of edges added to G is exactly $|Z|$, since each edge added corresponds to an edge in Z. Let r, c, and b be the number of R, C and B leaves in CG respectively. From our discussion of the mixed augmentation problem, we know that the number of edges in Z is $\max(r, c)$ if $|c-r|>b$, and otherwise is $\lceil (r+c+b)/2 \rceil$. These are clearly lower bounds on the number of new edges needed to solve the positive problem, and hence the solution is optimal.

Since the mixed augmentation problem can be solved in time linear in the number of edges in G, the positive security problem can also be. In [G] we present a solution to the security problem that is based on the algorithm in [ET] for the undirected augmentation problem. The algorithm is slower than the method outlined here, but the entire algorithm (including the parts from [ET]) is simpler and the presentation is self-contained.

Although we have been assuming throughout this paper that G, and hence CG is connected, this will not be true in general. When CG is disconnected, there exists a way to connect the trees of CG with a set of legal, node disjoint edges between leaves, so that these edges together with the solution to the positive problem on the resulting tree, is a solution to the original positive problem. Details of this are given in [G].

3.1. General security problem and mixed augmentation. The most general form of the security problem discussed in [G] can be expressed as follows: Graph G is a *mixed* bipartite graph with node sets R and C, where the only directed edges in G extend from a node in R to a node in C. As above, $\bar{G}$ contains only edges between nodes that are nonadjacent in G, and which run between R and C. For every such nonadjacent pair of nodes (i, j) in G, $\bar{G}$ will contain the edge directed edge $\langle i, j\rangle$, but may or may not contain edge $\langle j, i\rangle$. So the difference between this problem and the positive problem is that G may be a mixed graph (although it contains no directed edges from C to R), and that for some nonadjacent node pairs (i, j) in G, graph $\bar{G}$ does not contain

$\langle j, i \rangle$. It is known, however, that for the class of graphs that arise in the general security problem, G and $\bar{G}$ have the property that the addition of all edges in $\bar{G}$ to G results in a traversable graph. In contrast to the positive problem, this general security problem was recently shown, by M. Kao [K], to be NP-hard.

REFERENCES

[BT] F. BOESCH AND R. TINDELL, *Robin's theorem for mixed multigraphs*, Amer. Math. Monthly, 87 (1980), pp. 716–719.

[CGT] F. CHUNG, M. R. GAREY AND R. E. TARJAN, *Strongly connected orientations of mixed multigraphs*, Networks, 15 (1985), pp. 477–484.

[ET] K. P. ESWARAN AND R. E. TARJAN, *Augmentation problems*, this Journal, 5 (1976), pp. 653–665.

[FF] L. FORD AND D. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[FJ] G. N. FREDERICKSON AND J. JA'JA', *Approximation algorithms for several graph augmentation problems*, this Journal, 10 (1981), pp. 270–283.

[G] D. GUSFIELD, *A graphical approach to data security problems*, Dept. of Computer Science, Yale University Technical Report # Yale/DCS/TR-236, August 1984.

[KU] 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.

[O] O. ORE, *Graphs and Their Uses*, Random House, New York, 1963.

[K] M. Y. KAO, *Systematic protection of precise information on two-dimensional cross tabulated tables*, Ph.D. dissertation, Dept. of Computer Science, Yale University, New Haven, CT, 1986.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics

ORACLES FOR DETERMINISTIC VERSUS ALTERNATING CLASSES*

WILLIAM GASARCH†

Abstract. We construct oracles that force all possible relationships between NP and $EXP_k = DTIME(2^{O(n^k)})$. We generalize these results to obtain a theorem about oracles that force relationships between deterministic and nondeterministic (and alternating) classes, from which many corollaries follow. The corollaries are interesting because we compare a powerful type of machine (e.g., nondeterministic, alternating) to a less powerful type of machine that can use more time. One of our corollaries is that the result $DTIME(n \log^*(n)) \subseteq \Sigma_2 - TIME(n)$ does not relativize.

Key words. oracle, NP, alternation

AMS(MOS) subject classifications. 68Q15, 03O15

1. Introduction.

In a series of papers Book [7], [8] develops and applies padding techniques to a variety of problems. All the results obtained are inequalities (e.g., $NP \neq EXP_k = DTIME(2^{O(n^k)})$) without additional information; it is not known whether one class contains another. All the techniques relativize; hence oracle results demonstrate their limitation (see [2]). In particular, we show that such techniques will not suffice to give additional information about the nature of the inequalities Book obtained.

A technique or theorem that remains valid or true when all the machines are replaced by machines with oracles is said to *relativize.* Virtually all techniques in recursion theory relativize, since usually such techniques are only concerned with the input-output behavior of the machines, and could just as well be dealing with an oracle machine's input-output behavior. Baker, Gill and Solovay [2] showed that such techniques will not suffice to solve $P = ?NP$ by showing there exists oracles A and B such that $P^A \neq NP^A$ and $P^B = NP^B$. Oracles have been used to show that recursion theoretic techniques will not suffice to resolve questions about the classes R [29], U [15], $NP \cap \text{co-}NP$ [2], Σ_2^p [3], ZPP [21], PP, $\#P$ [1], exponential time [14], [17], various parallel classes [35], Σ_n^p, and $PSPACE$ [36] (easier proofs of the results in [36] are in [16]). Other topics shown hard to explore by purely recursion theoretic means include the Berman–Hartmanis Conjecture [23], P-immunity, P-simplicity [5], [18], complete sets for R or $NP \cap \text{co-}NP$ [32], and the existence of sparse sets in $NP - P$ [24].

We construct oracles that induce all possible relationships (with immunity) between NP and EXP_k. Other results involving exponential time, of a different nature, appear in [17]. Some of the oracle constructions we use for NP and EXP_k have appeared in [12]; however, here we modify them to obtain results about other complexity classes. A general theorem about oracles that force relationships between deterministic and nondeterministic (or alternating) classes is presented. This theorem has many corollaries. These corollaries involve comparing two complexity classes under an oracle. The comparisons are interesting because we compare a powerful type of machine (e.g., alternating) to a less powerful type of machine that can use more time. The classes considered include EXP_k, $D\#P$ (machines with a $\#P$ oracle (see [33],

* Received by the editors December 24, 1985; accepted for publication (in revised form) September 30, 1986. This work is part of the author's Ph.D. dissertation. This work was supported by National Science Foundation grants MCS-80-05-05386, MCS-82-00269 and MCS-80-10707.

† Department of Computer Science, University of Maryland, College Park, Maryland 20742.

[34])), $DTIME(c^n)$, $NTIME(d^n)$, $DTIME(n^c)$, $BTIME(n^d)$, and alternating classes [19]. One of our corollaries is that the result $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ [28] does not relativize.

In many oracle constructions in the literature, a set A is constructed to force $L^A \notin K^A$, where $L^{()}$ is an oracle dependent language and $K^{()}$ is a relativized complexity class. However, it is possible that L^A can still be approximated by a language in K^A, i.e., some infinite subset of L^A is in K^A. Oracles for which this does not happen were shown to exist in [6] and constructed recursively in [18], [31]. We borrow their terminology. (In the definitions below: L is a language; $\bar{L}$ is the complement of L, i.e., $\Sigma^* - L$; K, K_1, and K_2 are complexity classes.)

DEFINITION. L is *K-immune*, if L is infinite and has no infinite subset in K.

DEFINITION. L is *K-bi-immune* if both L and $\bar{L}$ are K-immune.

DEFINITION. $K_1 \subsetneqq K_2$ *with immunity* if $K_1 \subsetneqq K_2$, and there exists a language $L \in K_2$ that is K_1-immune.

DEFINITION. $K_1 \perp K_2$ *with immunity* (read as "K_1 is incomparable to K_2 with immunity") if there exist languages $L_1 \in K_1$, $L_2 \in K_2$, such that L_1 is K_2-immune and L_2 is K_1-immune.

DEFINITION. $K_1 \subsetneqq K_2$ *with bi-immunity* if there exists a language $L \in K_2$ that is K_1-bi-immune and $K_1 \subsetneqq K_2$.

DEFINITION. $K_1 \perp K_2$ *with bi-immunity* (read as "K_1 is incomparable to K_2 with bi-immunity") if there exist languages $L_1 \in K_1$, $L_2 \in K_2$, such that L_1 is K_2-bi-immune and L_2 is K_1-bi-immune.

We assume throughout that we have enumerations of:

(1) all oracle polynomial time bounded Turing Machines, $\{P_1^{()}, P_2^{()}, \cdots\}$, where machine $P_i^{()}$ takes time at most $p_i(x) = i + |x|^i$ on input x;

(2) all oracle nondeterministic polynomial time bounded Turing Machines $\{NP_1^{()}, NP_2^{()}, \cdots\}$, where machine $NP_i^{()}$ takes at most time $p_i(x) = i + |x|^i$ on input x along any path;

(3) (for a fixed k) all oracle $DTIME(2^{O(n^k)})$ time bounded Turing Machines, $\{E_1^{()}, E_2^{()}, \cdots\}$, where machine $E_i^{()}$ takes at most time $h_i(x) = 2^{i|x|^k}$ on input x.

We use the symbol "$L(M)$" to denote the language recognized by that machine M. Throughout this paper Σ is the fixed alphabet $\{0, 1\}$. The symbol "$\langle , \rangle$" denotes an easily computed mapping from $\Sigma^* \times \Sigma^*$ to N or $\Sigma^* \times \Sigma^*$ to Σ^*; usage will be clear from context, e.g., "$\langle i, j \rangle < s$" means that "$\langle i, j \rangle$" is a number which is less than s; and "$\langle i, x \rangle w$" means that $\langle i, x \rangle$ is a string and represents the concatenation of the string $\langle i, x \rangle$ with the string w. If a number appears as an argument to $\langle , \rangle$ then we mean its binary representation. If w_1 and w_2 are strings, then $|\langle w_1, w_2 \rangle| > \text{Max}(|w_1|, |w_2|)$. If n_1 and n_2 are numbers then $\langle n_1, n_2 \rangle > \text{Max}(n_1, n_2)$. The symbol $\langle , , \rangle$ denotes an easily computed mapping from $\Sigma^* \times \Sigma^* \times \Sigma^*$ to Σ^*.

2. *NP* versus EXP_k. One difference between NP and $EXP_k = DTIME(2^{O(n^k)})$ ($k \geqq 1$ arbitrary but fixed) is that NP is closed under "polynomial padding" while EXP_k is not. Book [7] exploited this difference to prove $NP \neq EXP_k$. His proof relativizes, i.e., it can be modified easily to show that for any oracle A, $NP^A \neq EXP^A$. This has also been observed by Dekhtyar [10].

The proof of $NP \neq EXP_k$ does not indicate in what way NP and EXP_k are related. $NP \subsetneqq EXP_k$, $EXP_k \subsetneqq NP$, and $NP \perp EXP_k$ are possibilities. Dekhtyar [10] claims (without proof) that oracles A, B, and C exist such that $NP^A \subsetneqq EXP_1^A$, $EXP_1^B \subsetneqq NP^B$, and $NP^C \perp EXP_1^C$. This shows that any technique that relativizes will not suffice to resolve how NP and EXP_1 relate.

It is possible that every set in NP has an infinite EXP_k subset or vice versa. By constructing oracles that induce all possible relationships, with immunity, between NP and EXP_k, we show that any technique that relativizes will not suffice to resolve any of these questions.

2.1. $P^A \subsetneqq NP^A \subsetneqq EXP_k^A$. One of the first and most basic oracle constructions in complexity theory was given in Baker, Gill and Solovay [2], where they showed the existence of an oracle A such that $P^A = NP^A$. For such an oracle we have $NP^A = P^A \subsetneqq EXP_k^A$ by the relativized version of the Time Hierarchy Theorem [20]. We construct an oracle A such that $P^A \subsetneqq NP^A \subsetneqq EXP_1^A$, and, moreover, we obtain these inequalities with immunity. Clearly this implies the same result for EXP_k^A, $k > 1$, instead of EXP_1^A.

The following proof is an arithmetic forcing argument similar to [24]. A priority argument proof of the same theorem is in [14].

THEOREM 1. *There exists a recursive oracle A such that $P^A \subsetneqq NP^A \subsetneqq EXP_1^A$, and both proper inclusions hold with immunity.*

Proof. Let

$$L_1^A = \{x \mid \exists_y |y| = |x|,\ xy \in A, |xy| \text{ is not a power of two}\},$$

$$L_2^A = \{0^n \mid 1^{2^n} \in A\}.$$

Note that all strings in L_1^A are of a length that is not a power of two. This will ensure that L_1^A, L_2^A, and the coding of NP^A into EXP_1^A are affected by disjoint sets of strings.

Clearly we have $L_1^A \in NP^A$ and $L_2^A \in EXP_1^A$. We need to ensure that $L_1^A (L_2^A)$ is infinite and contains no infinite subset in $P^A (NP^A)$. We state this in terms of requirements:

$$RP_i\colon\ L(P_i^A) \cap L_1^A \text{ infinite} \to L(P_i^A) \cap \overline{L_1^A} \neq \varnothing,$$

$$RNP_i\colon\ L(NP_i^A) \cap L_2^A \text{ infinite} \to L(NP_i^A) \cap \overline{L_2^A} \neq \varnothing.$$

The infinitude of $L_1^A (L_2^A)$ follows from the construction and need not be stated as a formal requirement.

We need a function to help code NP^A into EXP_1^A. Let $c\colon N \times N \times N \to 2N+1$ be an injective function computable in polynomial time such that for all (x_1, x_2, x_3), $c(x_1, x_2, x_3) > x_i$.

We code $NP^A \subseteq EXP_1^A$ via:

For all i, for all x

$$NP_i^A(x) \text{ accepts in } t \text{ steps iff } 0^{c(i,1x,t)} \in A.$$

(Note that $1x$ is interpreted as a binary number.)

We construct A in stages. At the end of each stage we define A_s and $r(s)$: A_s is the set of strings placed into A through stage s, $r(s)$ is a "restraint function" in that at no later stage may an element w, $|w| < r(s)$, be placed into A. The restraint function guarantees that requirements which are declared satisfied at stage s remain satisfied; and that $A = \bigcup_{s=0}^{\infty} A_s$ is recursive.

CONSTRUCTION

Stage 0: $A_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$:

There are six substages. To avoid notational problems the phrase "$A_s \leftarrow A_s \cup \{w\}$" means add w to A_s. If "A_s" is referenced later we mean the updated version. The phrase "decide a string" means to decide whether it is in A or not. "Restrain a string" means to decide that the string will never enter A.

We pick a length n so that later (in substage c) we can run $P_i^{A_s}$ $(0 \leqq i \leqq s/2)$ on some string of length $n/2$ and diagonalize (to satisfy some RP_i). Since the diagonalization will involve strings of length n, we pick n large (so the number of strings of length n is large) and we are careful to not put too many strings of length n into A_s in substages a and b.

Let $n_s = n$ be the least even number, not a power of 2, such that $n > r(s)$ and $p_s^2(n) < 2^{n/2}$.

Substage a: (Decide all strings of length k, $r(s) < k < n$). For each k, $r(s) < k < n$, if $k = c(i, 1x, t)$ for some (i, x, t), then run $NP_i^{A_s}(x)$ for t steps. If it accepts then set $A_s \leftarrow A_s \cup \{0^{c(i,1x,t)}\}$. Since $t < c(i, 1x, t)$, placing $0^{c(i,1x,t)}$ into A does not affect the computation that this string codes.

Substage b: (Decide all strings of length k, $n < k < p_s(n)$). We want to do a similar construction as in substage a, but there is one glitch: strings of length $n < k$ may enter A at a later time and invalidate a computation which we coded. To avoid this we decide certain strings of length n to preserve computations. The key point is that when we finish coding there will still be enough strings of length n undecided to help satisfy requirements RP_i, $0 \leqq i \leqq s/2$.

For each k, $n < k \leqq p_s(n)$, such that $k = c(i, 1x, t)$ for some (i, x, t) run $NP_i^{A_s}(x)$. If it accepts then $A_s \leftarrow A_s \cup \{0^{c(i,1x,t)}\}$ and we restrain from A all strings of length n that were queried.

If it rejects then we search for a set of unrestrained strings B (of length n), such that $NP_i^{A_s \cup B}(x)$ accepts. If we find such a B then, for a particular accepting path, restrain all elements queried of length n that are not in $A_s \cup B$, and set

$$A_s \leftarrow A_s \cup \{x \mid x \in B \text{ and } x \text{ is queried on that path}\} \cup \{0^{c(i,1x,t)}\}.$$

Note that the number of elements of length n which are decided is less than $t < p_s(n)$.

If none accept then we do not put $0^{c(i,1x,t)}$ into A because (key point!) whichever elements of length n enter A, NP_i^A will not accept, since we tried all possibilities.

Substage c: (Decide remaining strings of length n, and satisfy RP_i). Note that the number of strings of length n not yet decided is less than (number of machines run during substage b) $\times$ (max number of strings queried in such a run) $< p_s(n) \times p_s(n) = p_s^2(n) < 2^{n/2}$.

For each x, $|x| = n/2$, let $D_x = \{xy : |y| = n/2\}$. Since there are less than $2^{n/2}$ decided strings, one of the D_x must contain only undecided strings. Let D_{x_0} be the least such (for definiteness). Note that $x_0 \in L_1^A$ iff $D_{x_0} \cap A \neq \emptyset$. Run $P_i^{A_s}(x_0)$ for all i such that $0 \leqq i \leqq s/2$ and RP_i not satisfied. If any accept then declare all such RP_i satisfied, since $x_0 \notin L_1^{A_s}$ and the restraint will be set to insure $x_0 \notin L_1^A$. If all reject then take the least (for definiteness) $w \in D_{x_0}$ not queried in any of the $P_i^{A_s}(x_0)$ computations and set $A_s \leftarrow A_s \cup \{w\}$. This does not satisfy any RP_i but it does help to make L_1^A infinite.

Substage d: (Requirements RNP_i, $1 \leqq i \leqq s/2$). Run $NP_i^{A_s}(0^n)$ for all i such that $1 \leqq i \leqq s/2$, and RNP_i not satisfied. Since all strings of length $\leqq p_s(n)$ are now decided, all the $NP_i^{A_s}(0^n)$ computations are valid. If $NP_j^{A_s}(0^n)$ accepts then declare all such RNP_j satisfied, since $1^{2^n} \notin A_s$, and $r(s)$ will be set to ensure $1^{2^n} \notin A$. If all reject then set $A_s \leftarrow A_s \cup \{1^{2^n}\}$. This does not satisfy any RNP_i but it helps make L_2^A infinite.

Substage e: (Decide strings of length k, $p_s(n) < k \leqq 2^n$). Similar to substage a.

Substage f: Set $A_{s+1} \leftarrow A_s$ as it stands at the end of substage e.

Set $r(s+1) \leftarrow 2^n + 1$.

END OF CONSTRUCTION

$L_1^A(L_2^A)$ is infinite because during stages $1, 2, \cdots, s+1$ we consider only machines with indices less than $s/2$; hence in at most $s/2$ stages nothing enters $L_1^A(L_2^A)$ so $|L_1^{A_s}| \geqq s - (s/2) = s/2$ ($|L_2^{A_s}| \geqq s - s/2 = s/2$).

LEMMA 2. $NP^A \subseteq EXP_1^A$.

Proof. We show that for any i, $L(NP_i^A) \in EXP_1^A$. The string $x \in L(NP_i^A)$ iff $NP_i^A(x)$ accepts in $p_i(|x|)$ steps which happens exactly when $0^{c(i,1x,p_i(x))} \in A$. Since, looking at $1x$ as a binary number, $1x < 2^{|1x|}$, $c(i, 1x, p_i(x)) < c(i, 2^{|1x|}, p_i(x)) = 2^{O(|x|)}$; hence an EXP_1^A machine has time to write down the query. □

Note that $1x$ was the largest argument to the function $c(\cdots)$. The "t" in $c(i, 1x, t)$ is not strictly necessary but makes the proofs easier. Without it the code would work for almost all x, as (finitely often) the code strings would interfere with the computation. It is necessary to be able to code only one string per length for the sake of substage c.

All elements $x \in L_1^A$ are there because we put them there during substage c of stage $s+1$ where $|x| = n$; no element enters L_1^A accidentally. Formally, let $x_0(s)$ denote the x_0 used in substage c of stage s. Note that $L_1^A \subseteq \{x_0(t) \mid t \in N\}$.

LEMMA 3. *Each requirement RP_i is satisfied.*

Proof. If a requirement ever receives attention, then it is permanently satisfied and will never receive attention again; hence, there exists a least stage s_0 such that for all $s \geqq s_0$, for all $j < i$, RP_j does not receive attention at stage s.

We show that if $P_i^A \cap L_1^A$ is infinite then RP_i receives attention. Let $s_1 = \max(s_0, 2i)$. Let s_2 be the least number greater than s_1 such that $x_0(s_2) \in P_i^A$ (such a string must exist as $P_i^A \cap L_1^A$ is infinite and all elements of L_1^A are $x_0(s)$ for some s). If RP_i has not been satisfied then at stage s_2, during substage c, RP_i receives attention and is thus satisfied. □

LEMMA 4. *Each requirement RNP_i is satisfied.*

Proof. Similar to Lemma 3. □

2.2. $EXP_k^B \subsetneqq NP^B$. In this section we show that the opposite inclusion of the previous section is possible as well. We obtain a stronger type of result than Theorem 1 in that we exhibit an oracle B such that there is an infinite language $L^B \in NP^B$ with no infinite subset of L^B or $\overline{L}^B$ in EXP_k^B. We say that L^B is EXP_k^B-bi-immune. Albert Meyer, who suggested the problem to us, pointed out the following consequence, which is proved rigorously in [14].

PROPOSITION. *Let B be an oracle such that there is an infinite $L \in NP^B$ which is EXP_k^B-bi-immune. Then any deterministic algorithm for L must operate in time greater than $2^{O(n^k)}$ on all but a finite set of points.*

Lynch [25] proved that if $L \notin DTIME(f(n))$ then there exists a set $L' \subseteq L$ such that every infinite subset of L' is not in $DTIME(f(n))$ in the following very strong way: any deterministic algorithm that decides L' must take more than time $f(n)$ on all but a finite number of values. L' is called the "hard core" of L, because it is the reason that L is hard ("hard" meaning $L \notin DTIME(f(n))$). In the above proposition, the set L is its own hard core with respect to EXP_k^B, that is, there is no need to look for a proper subset of L that makes L hard; the set L is hard in and of itself. For more on hard cores see [11], and for more on bi-immunity see [4].

THEOREM 5. *There exists a recursive oracle B such that $EXP_k^B \subsetneqq NP^B$, and this inequality is witnessed by a language $L^B \in NP^B$ such that L^B is EXP_k^B-bi-immune.*

Proof. For clarity we present the proof for $k = 1$. The proof for larger values of k is similar.

We construct the set B in stages. During the construction we code EXP_1^B into NP^B by the following method:

For all i and x, E_i^B accepts x iff there exists w such that $\langle i, x\rangle w \in B$ and $|w| = |\langle i, x\rangle|^8 + 1 - |\langle i, x\rangle|$. Clearly this implies $EXP_1^B \subseteq NP^B$. Note that $|\langle i, x\rangle w| = |\langle i, x\rangle|^8 + 1$ is not a fourth power. This will ensure that no code string is ever placed into B by accident.

We let

$$L^B = \{x \mid \exists_y y \in B, |w| = |x|^4\}.$$

Note that $L^B \in NP^B$ for all B.

To take care of infinite subsets of L^B and $\overline{L^B}$ we have the following requirements:

$$R_{\langle 1,i\rangle}\colon L(E_i^B) \text{ infinite} \rightarrow L(E_i^B) \cap \overline{L^B} \neq \varnothing.$$

$$R_{\langle 2,i\rangle}\colon L(E_i^B) \text{ infinite} \rightarrow L(E_i^B) \cap L^B \neq \varnothing.$$

The requirements inherit a priority ordering from the ordering given by the pairing function $\langle \text{-}, \text{-}\rangle$.

Recall that h_i bounds the running time of machine E_i. We let B_s denote the strings put into B through the first s stages.

CONSTRUCTION

Stage 0: $B_0 \leftarrow \varnothing$.

Stage $s+1$:

(1) For each $i \leqq s$, if $h_i(s) < 2^{s^2}$, the run $E_i^{B_s}$ on all strings of length s and preserve each of these computations by restraining from B all the strings queried in the computation which were not in B_s. Note that the total number of strings restrained at this stage is at most $s \times 2^s \times 2^{s^2} < 2^{s^3}$.

(2) Find the least $e = \langle j, i\rangle < s$ such that

(a) R_e is not satisfied.

(b) There is an $x \in \Sigma^*$ such that $|x| = s$ and $E_i^{B_s}$ accepts x.

(c) $h_i(s) < 2^{s^2}$.

If $j = 1$, then to ensure $E_i^B \cap \overline{L^B} \neq \varnothing$ we restrain all strings of length s^4 from B.

If $j = 2$, then to ensure $E_i^B \cap L^B \neq \varnothing$ we place into B the least string of length s^4 that is not restrained from B. (Note that such a string will not be a code string.) There must be such a string since the total number of strings restrained from B up to this point in the construction is less than $\sum_{i=1}^{s} 2^{i^3} < 2^{s^4}$. (Note that the $j = 1$ case of the previous stages restrains only strings of length less than s^4.)

R_e is said to have acted and is now declared satisfied.

(3) For each $E_i^{B_s}(x)$ which has just been run and which accepted x, find some w such that $|w| = |\langle i, x\rangle|^8 - |\langle i, x\rangle| + 1$, and $\langle i, x\rangle w$ is not restrained from B. Put $\langle i, x\rangle w$ into B. Such a w must exist since the number of possible w is $2^{|\langle i,x\rangle|^8 - |\langle i,x\rangle| + 1}$. Note that $|\langle i, x\rangle| \geqq |x| = s$. The function $f(n) = n^8 - n + 1$ in domain $n \geqq s$ has its only minimum at $n = s$. Hence $2^{|\langle i,x\rangle|^8 - |\langle i,x\rangle| + 1} \geqq 2^{s^8 - s + 1}$, which exceeds 2^{s^4}, the number of strings restrained. Hence such a w must exist.

END OF CONSTRUCTION

We prove that each R_e is eventually satisfied. Note that each R_e is acted upon at most once. Assume that $e = \langle j, i\rangle$ and E_i^B accepts an infinite set. Let s_0 be a stage beyond which no R_k, $k < e$, will act, and such that for all $s > s_0$, we have $h_i(s) < 2^{s^2}$. Past s_0, all computations of E_i^B are preserved. Since E_i^B is infinite, there is a stage $s_1 \geqq s_0$ where $E_i^{B_{s_1}}$ accepts some string of length s_1. At this stage R_e acts and becomes satisfied.

2.3. NP^C incomparable to EXP_k^C. It may be that neither NP nor EXP is contained in the other. We show there is an oracle for which this is the case.

THEOREM 6. *There is a recursive oracle C such that NP^C is incomparable to EXP_k^C with immunity.*

Proof. Let

$$L_1^C = \{0^n \mid \exists_x x \in C |x| = n^{k+1} \text{ and } n \text{ is odd}\},$$

$$L_2^C = \{0^n \mid 1^{2^n} \in C\}.$$

(The condition that n be odd in L_1^C will make actions taken for L_1^C and L_2^C independent of each other.)

Clearly $L_1^C \in NP^C$ and $L_2^C \in EXP_k^C$. We need to ensure that $L_1^C(L_2^C)$ is infinite and contains no infinite subset in $EXP_k^C(NP^C)$. We state this in terms of requirements:

$$RE_i\colon L(E_i^C) \text{ infinite} \to L(E_i^C) \cap \overline{L_1^C} \neq \varnothing,$$

$$RNP_i\colon L(NP_i^C) \text{ infinite} \to L(NP_i^C) \cap \overline{L_2^C} \neq \varnothing.$$

We construct C in stages. At the end of each stage we define C_s and $r(s)$: C_s is the set of strings placed into C through stage s, $r(s)$ is a "restraint function" in that at no later stage may an element w, $|w| < r(s)$, be placed into C. The restraint function guarantees that a requirement satisfied at stage s remains so, and that $C = \bigcup_{s=0}^{\infty} C_s$ is recursive.

CONSTRUCTION

Stage 0: $C_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$: There are two cases:

Case 1: s is even (look at RE_i requirements). Let $n_s = n$ be the least odd number such that $n \geqq r(s)$. Run $E_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $h_i(n) < (1/s) \times 2^{n^{k+1}}$, and RE_i not satisfied. If any accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RE_i satisfied, since $0^n \notin L_1^{C_s}$ and the restraint will be set to insure $0^n \notin L_1^C$. If no $E_i^{C_s}(0^n)$ accepts then take the least x (for definiteness), such that $|x| = n^{k+1}$, and x was not queried in any of the above computations, and set $C_{s+1} \leftarrow C_s \cup \{x\}$. Such an x exists because there are at most $(s/4) \times (1/s) \times 2^{n^{k+1}} < 2^{n^{k+1}}$ strings queried. This action puts 0^n into L_1^C, which increases the cardinality of L_1^C, and thus helps it to be infinite.

Set $r(s+1) \leftarrow 2^{n^{k+1}} + 1$. The value $r(s+1)$ exceeds the length of any string queried in the above computations so $E_i^{C_s}(0^n) = E_i^C(0^n)$. Since $r(s+1) > \text{Max}\,(n^{k+1}, r(s))$, we have $0^n \notin L_1^{C_{s+1}}$, and $0^n \notin L_1^C$. Therefore if an RE_i requirement is satisfied at stage $s+1$ then it remains satisfied at all later stages.

Case 2: s is odd (look at RNP_i requirements). Let $n = r(s)$. Run $NP_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $p_i(n) < 2^{n^k}$, and RNP_i not satisfied. If any $NP_i^{C_s}(0^n)$ accepts then let $C_{s+1} \leftarrow C_s$ and declare all such RNP_i satisfied, since $0^n \notin L_2^{C_s}$ and the restraint will be set to insure $0^n \notin L_2^C$. If no $NP_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s \cup \{1^{2^{n^k}}\}$. This action puts 0^n into L_2^C, which increases the cardinality of L_2^C, and thus helps it to be infinite.

Set $r(s+1) \leftarrow 2^{n^k} + 1$. If an RNP_i requirement is satisfied at stage $s+1$, then it remains satisfied at all later stages by the same reasoning used for the R_{e_i} requirements.

END OF CONSTRUCTION

$L_1^C(L_2^C)$ are infinite by the same argument used in Theorem 1. It remains to show the requirements are satisfied. While reading the proof bear in mind that the elements in L_1^C and L_2^C are there because we purposely put them in; no element is in by accident.

LEMMA 7. *Each requirement RE_i is satisfied.*

Proof. Assume not and let i_0 be the least i with RE_{i_0} not satisfied. Then $L(E_{i_0}^C)$ is an infinite subset of L_1^C, and i_0 is never declared satisfied. There must exist an s_0 such that

(a) $i_0 \leqq s_0/4$.

(b) For all $n > n_{s_0}$, $h_i(n) < (1/s) \times 2^{n^{k+1}}$.

(c) For all $i < i_0$, if RE_i will be declared satisfied (as opposed to being satisfied by having E_i^C accept a finite set) then it has been declared satisfied at some time earlier than s_0.

For any even $s > s_0$, at stage s $E_{i_0}^{C_s}(0^{n_s})$ will be run. If it ever accepts then RE_{i_0} will be satisfied, contrary to hypothesis. Hence it rejects 0^{n_s} for all $s > s_0$. Since $L(E_{i_0}^C) \subseteq L_1^C \subseteq \{0^{n_s} \mid s = 0, 1, 2, \cdots, s_0\}$, $L(E_{i_0}^C)$ is finite contrary to hypothesis. Thus RE_{i_0} is satisfied. □

LEMMA 8. *Each requirement RNP_i is satisfied.*

Proof. Similar to Lemma 7. □

3. A general theorem. The proofs of Theorems 1, 5, and 6 only use certain relations between polynomial and exponential functions; hence, they are a natural target for generalization. From our general theorem we obtain oracles relating

(a) $DTIME(n^c)$ and $NTIME(n^d)(c > d > 0)$,
(b) $DTIME(c^n)$ and $NTIME(d^n)(c > d > 0)$,
(c) $DTIME(n^{O(\log n)})$ and $NTIME(n^k)$ $(k > 1)$,
(d) P and $NTIME(n^k)$ $(k > 1)$,
(e) $DTIME(n \log n)$ and $NTIME(n)$,
(f) $DTIME(n \log^* (n))$ and $NTIME(n)$,
(g) $D \# P$ and EXP_k.

Items (a) and (b) answer questions raised by Janos Makowsky [26]. These comparisons are interesting because the $DTIME$ machines have more time to operate, but the $NTIME$ machines are nondeterministic; therefore, there is no obvious relation between the two. $NTIME(f)$ can be replaced by Σ_j-$TIME(f)$ in the list above. $NTIME(f)$ can also be replaced by $ATIME(f)$, which allows an unbounded number of alternations, though any path of the machine must halt in time f.

THEOREM 9. *Let $D_1, D_2, \cdots$ and $N_1, N_2, \cdots$ be sequences of computable functions such that*

(1) *For all n and i, $N_{i+1}(n) \geqq N_i(n) \geqq n$, $D_{i+1}(n) \geqq D_i(n) \geqq n$.*

(2) *For all i*

$$\lim_{n\to\infty} D_i(n) = \infty, \qquad \lim_{n\to\infty} N_i(n) = \infty, \qquad \lim_{n\to\infty} \frac{N_i(n)}{D_1(n)} = 0, \qquad \lim_{n\to\infty} \frac{D_i(n)}{2^{N_1(n)}} = 0.$$

(3) *D_1 and N_1 are time constructible, so one can compute $D_1(n)(N_1(n))$ in $O(D_1(n))(O(N_1(n)))$ steps. By the linear speed-up theorem* [20] *we may assume the order constant is one.*

(4) *For all i the function that maps x to $N_i(|x|)$ is computable in $D_1(|x|)$ steps.*

(5) *$N_i(n)$ is odd infinitely often.*

Let

$$D^{()} = \bigcup_{i=0}^{\infty} DTIME^{()}(D_i(n)), \qquad N^{()} = \bigcup_{i=0}^{\infty} NTIME^{()}(N_i(n)).$$

Then:

(a) *There exists an oracle A such that $N^A \subsetneqq D^A$ with immunity.*

(b) *There exists an oracle B such that* $D^B \subsetneqq N^B$ *with bi-immunity.*

(c) *There exists an oracle C such that* N^C *and* D^C *are incomparable with immunity.*

Proof. All three proofs are similar to those in Theorems 1, 5, and 6, and hence we shall be brief. We use the same symbols N_i and D_i for machines as well as time bounds; usage will be clear from context.

(a) $N^A \subsetneqq D^A$ with immunity. Let $L^A = \{0^n \mid 1^{D_1(n)} \in A\}$. Clearly $L^A \in D^A$. Oracle A is constructed such that the following requirements are satisfied:

$$RN_i\colon L(N_i^A) \cap L^A \text{ infinite} \rightarrow N_i^A \cap \overline{L^A} \neq \varnothing.$$

We code N^A into D^A via

$$N_i^A(x) \text{ halts in } t \text{ steps iff } \langle i, x, 0^t\rangle \in A.$$

CONSTRUCTION

Stage 0: $A_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$ (satisfy RN_s):

Let $n_s = n$ be the least number such that $n > r(s)$ and $D_1(n) > N_s(n)$. For each k, $n \leqq k \leqq D_1(n)$, and for each w, $|w| = k$, $w = \langle i, x, 0^t\rangle$, if $N_i^{A_s}(x)$ accepts in t steps along any computation path then place w into A. Note that the string w is so long that placing it into A does not interfere with the computation it codes.

Run $N_s^{A_s}(0^n)$; if it rejects then place $1^{D_1(n)}$ into A. Since $N_s(n) < D_1(n)$, and we will not place any string of length less than $D_1(n)$ into A, $N_s^{A_s}(0^n) = N_s^A(0^n)$ and the requirement is satisfied.

Set $r(s+1) \leftarrow D_1(n)$.

END OF CONSTRUCTION

We see that $N^A \subseteq D^A$: For a fixed i

$$x \in N_i^A \quad \text{iff } \langle i, x, 0^{N_i(|x|)}\rangle \in A.$$

An easy calculation shows the query length is less than $D_1(|x|)$ for $|x|$ large.

Each RN_i is satisfied by the same reasoning used in Lemma 2.

(b) $D^B \subsetneqq N^B$ with bi-immunity. Let $L^B = \{x \mid \exists_y y \in B, |y| = 3N_1(|x|)\}$. Note that $L^B \in N^B$ for all B. B is constructed to satisfy the following requirements:

$$R_{\langle 1,i\rangle}\colon L(D_i^B) \text{ infinite} \rightarrow L(D_i^B) \cap \overline{L^B} \neq \varnothing,$$

$$R_{\langle 2,i\rangle}\colon L(D_i^B) \text{ infinite} \rightarrow L(D_i^B) \cap L^B \neq \varnothing.$$

We code D^B into N^B by:

$$D_i^B(x) \text{ accepts iff there exists } w, |w| = 3N_1(|x|)+1 \text{ and } \langle i, x\rangle w \in B.$$

(We modify $\langle,\rangle$ so that it always produces a string of length divisible by three.)

Note that the coding depends on strings of length $\equiv 1 \bmod 3$ in B while L^B depends on strings of length $\equiv 0 \bmod 3$ in B.

We construct B and $\bar{B}$ simultaneously. The expression "restrain from B" means "place into $\bar{B}$."

CONSTRUCTION:

Stage 0: $B_0 \leftarrow \varnothing$.

Stage $s+1$:

(1) For each $i \leqq s$, if $D_i(s) < 2^{N_1(s)}$, then run $D_i^{B_s}$ on all strings of length s and preserve each of these computations by restraining from B all the strings queried in

the computation that were not in B_s. Note that the total number of strings restrained at this stage is at most $s \times 2^s \times 2^{N_1(s)} < 2^{2N_1(s)}$.

(2) Find the least $e = \langle j, i \rangle < s$ such that

(a) R_e is not satisfied.

(b) There is an $x \in \Sigma^*$ such that $|x| = s$ and $D_i^{B_s}$ accepts x.

If $j = 1$, then to ensure $D_i^B \cap L^B \neq \emptyset$ we restrain all strings of length $3N_1(|x|)$.

If $j = 2$, then to ensure $D_i^B \cap L^B \neq \emptyset$ we place into B_{s+1} the least string of length $3N_1(|x|)$ that is not restrained from B. There must be such a string since the total number of strings restrained from B up to this point in the construction is less than $\sum_{i=1}^{s} 2^{2N_1(i)} < 2^{3N_1(s)}$.

Note that the $j = 1$ case of the previous stages restrains only strings of length $3N_1(|x|)$. These strings will not be relevant when we try to code D_i^B into NP^B.

R_i is now said to be satisfied.

(3) For each $D_i^{B_s}(x)$ that has just been run and has accepted x, find some w such that $|w| = 3N_1(|x|) + 1$ and $\langle i, x \rangle w$ is not restrained from B. Put $\langle i, x \rangle w$ into B_{s+1}. Such a w will exist, since the number of possible w's is $2^{3N_1(|x|)+1}$ which exceeds $2^{3N_1(s)}$, the number of strings restrained (not including those possibly restrained in the $j = 1$ case above, since they are all of length $3N_1(|x|)$).

END OF CONSTRUCTION

The requirements are satisfied for reasons similar to those stated at the end of Theorem 5.

(c) N^C incomparable to D^C with immunity. Let

$$L_1^C = \{0^n \mid \exists_x x \in C |x| = 2N_1(n) + 1, N_1(n) \text{ odd}\},$$

$$L_2^C = \{0^n \mid 1^{2D_1(n)} \in C, D_1(n) \text{ even}\}.$$

The parity of n is included so that actions taken to place or restrain elements from $L_1^C (L_2^C)$ do not affect $L_2^C (L_1^C)$. Clearly we have $L_1^C \in N^C$ and $L_2^C \in D^C$. We need to ensure that $L_1^C (L_2^C)$ is infinite and contains no infinite subset in $D^C (N^C)$. We state this in terms of requirements:

$$RD_e\text{: } L(E_e^C) \cap L_1^C \text{ infinite} \rightarrow L(E_e^C) \cap \overline{L_1^C} \neq \emptyset,$$

$$RN_e\text{: } L(N_e^C) \cap L_2^C \text{ infinite} \rightarrow L(N_e^C) \cap \overline{L_2^C} \neq \emptyset.$$

We construct C in stages. The same conventions apply here as in Theorem 6.

CONSTRUCTION

Stage 0: $C_0 \leftarrow \emptyset$, $r(0) \leftarrow 0$.

Stage $s+1$: There are two cases:

Case 1: s is even (look at RD_e requirements). Let $n_s = n$ be the least odd number such that $n \geqq r(s)$. Run $D_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $D_i(n) < (1/s) \times 2^{2N_1(n)+1}$, and RE_i not satisfied. We denote n_s by n throughout. If any accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RE_i satisfied, since $0^n \notin L_1^{C_s}$ and the restraint will be set to insure $0^n \notin L_1^C$. If no $D_i^{C_s}(0^n)$ accepts then take the least x (for definiteness), such that $|x| = 2N_1(n) + 1$, x was not queried in any of the above computations, and set $C_{s+1} \leftarrow C_s \cup \{x\}$. Such an x exists because there are at most $(s/4) \times (1/s) \times 2^{2N_1(n)+1} < 2^{2N_1(n)+1}$ strings queried. This action puts 0^n into L_1^C, which helps it to be infinite.

Case 2: s is odd (look at RN_e requirements). Let n be the least number larger than $r(s)$. Run $N_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $N_i(n) < 2D_1(n)$, and RNP_i not satisfied. If any $N_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RN_i satisfied, since $0^n \notin L_2^{C_s}$ and the restraint will be set to insure $0^n \notin L_2^C$. If no $N_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s \cup \{1^{2D_1(n)}\}$. This action puts 0^n into L_2^C which helps it to be infinite.

In either case set $r(s+1) \leftarrow 2^{n^k}+1$. This preserves all computations; hence if any requirement is satisfied at stage s then it is permanently satisfied.

END OF CONSTRUCTION

The same reasoning used in Theorem 6 applies to prove L_1^C, L_2^C infinite and all the requirements satisfied.

In all the constructions in this paper (and in the above proof), where we diagonalize a language out of a nondeterministic class we do so by making the questions a machine has to ask too long. The same technique can be used to diagonalize a language out of any class where no computation path can write a long question on the query tape. In particular, we can diagonalize out of Σ_i^p and other alternating classes (see [19] for definition of alternating machines). We will use this fact in some of the corollaries below.

4. Corollaries. We list some corollaries of the general theorem. Due to the nature of the general theorem each corollary consists of three statements about the existence of oracles to force relationships between complexity classes. Some of the statements are not logically independent, e.g., Corollary 10-(1) implies Corollary 13-(1). In some cases we can, using padding techniques, prove that for all oracles the classes in question cannot be equal. No proofs are given for these, as the reader may find similar proofs in [7] and [8]. By the comment following Theorem 9, in all the corollaries below *NTIME* can be replaced by *ATIME* or Σ_i-*TIME*.

Corollaries that mention $DTIME(f)$ and/or $NTIME(f)$, (Σ_j-$TIME(f)$) are proved using Theorem 9 with $D_i(n)=f(n)$ and/or $N_i(n)=f(n)$. Corollaries that mention $DTIME(O(f))$ and/or $NTIME(O(f))$ use $D_i(n)=if(n)$ and/or $N_i(n)=if(n)$. Similar remarks apply to $DTIME(n^{O(f)})$, $NTIME(n^{O(f)})$, and Σ_n-$TIME(n^{O(f)})$. In most of these cases, proofs are omitted.

COROLLARY 10. *Given c, d real numbers such that* $1<d<c$, *there exist oracles A, B, and C such that*

(1) $NTIME^A(n^d) \subsetneq DTIME^A(n^c)$,

(2) $DTIME^B(n^c) \subsetneq NTIME^B(n^d)$,

(3) $NTIME^C(n^d)$ *incomparable to* $DTIME^C(n^c)$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 11. *Given c, d real numbers such that* $0<d<c$ *there exist oracles A, B, and C such that*

(1) $NTIME^A(d^n) \subsetneq DTIME^A(c^n)$,

(2) $DTIME^B(c^n) \subsetneq NTIME^B(d^n)$,

(3) $NTIME^C(d^n)$ *incomparable to* $DTIME^C(c^n)$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 12. *Given a* $k \geqq 1$, *there exist oracles A, B, and C such that*

(1) $NTIME^A(n^k) \subsetneq DTIME^A(n^{O(\log n)})$,

(2) $DTIME^B(n^{O(\log n)}) \subsetneq NTIME^B(n^k)$,

(3) $NTIME^C(n^k)$ *incomparable to* $DTIME^C(n^{O(\log n)})$,

(4) *For all oracles X,* $NTIME^X(n^k) \neq DTIME^X(n^{O(\log n)})$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 13. *Given a* $k>1$, *there exist oracles A, B, and C such that*

(1) $NTIME^A(n^k) \subsetneq P^A$,

(2) $P^B \subsetneq NTIME^B(n^k)$,

(3) $NTIME^C(n^k)$ *incomparable to* P^C,

(4) *For all oracles X,* $NTIME^X(n^k) \neq P^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 14. *There exist oracles A, B, and C such that*

(1) $NTIME^A(n) \subsetneqq DTIME^A(n \log n)$,
(2) $DTIME^B(n \log n) \subsetneqq NTIME^B(n)$,
(3) $NTIME^C(n)$ *incomparable to* $DTIME^C(n \log n)$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 15. *There exist oracles A, B, and C such that*
(1) $NTIME^A(n) \subsetneqq DTIME^A(n \log^*(n))$,
(2) $DTIME^B(n \log^*(n)) \subsetneqq NTIME^B(n)$,
(3) $NTIME^C(n)$ *incomparable to* $DTIME^C(n \log^*(n))$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

By the note following Theorem 9, we may replace *NTIME* with any alternating class. This yields Corollaries 16, 17, and 18.

COROLLARY 16. *Given a positive real k and an integer j there exist oracles A, B, and C such that*
(1) $P^A \subsetneqq \Sigma_j^{p,A} \subsetneqq EXP_k^A$,
(2) $EXP_k^B \subsetneqq \Sigma_j^{p,B}$,
(3) $\Sigma_j^{p,C}$ *incomparable to* EXP_k^C,
(4) *For all oracles X,* $\Sigma_j^{p,X} \neq EXP_k^X$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. For (1) a simple modification of the construction in Theorem 1 will suffice. For (2) and (3) let $D_i(n) = 2^{in^k}$ and $N_i(n) = n^i$ and use the note following Theorem 9. □

It is known that $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ [28], (which [28] showed implies $DTIME(n) \neq NTIME(n)$). This was proven by techniques that do not appear to relativize. Corollary 17 verifies that, indeed, the proof does not relativize, nor can it be made to relativize with modification. The containment is known to be false for almost all oracles [12].

COROLLARY 17. *Given* $k \geqq 1$ *there exist oracles A, B, and C such that*
(1) $\Sigma_2\text{-}TIME^A(n) \subsetneqq DTIME^A(n \log^*(n))$,
(2) $DTIME^B(n \log^*(n)) \subsetneqq \Sigma_2\text{-}TIME^B(n)$,
(3) $\Sigma_2\text{-}TIME^C(n)$ *incomparable to* $DTIME^C(n \log^*(n))$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 18. *Given a* $k \geqq 1$ *there exist oracles A, B, and C such that*
(1) $ATIME^A(n^{O(1)}) \subsetneqq EXP_k^A$,
(2) $EXP_k^B \subsetneqq ATIME^B(n^{O(1)})$,
(3) $ATIME^C(n^{O(1)})$ *incomparable to* EXP_k^C,
(4) *For all oracles X,* $ATIME^X(n^{O(1)}) \neq EXP_k^X$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Since $ATIME(n^{O(1)}) = PSPACE$ [19] one may think that we can replace $ATIME^A(n^{O(1)})$ with $PSPACE^A$ in the above corollary; however, this depends on the definition of relativized space bounded machines. If the space bound applies to the oracle tape then $ATIME(n^{O(1)}) = PSPACE$ relativizes, but there exists A such that $A \notin DSPACE^A(\log n)$ (any nonrecursive A will suffice). There are other definitions of relativized space [9], [30] that allow the machine to ask long questions. Under these definitions, $PSPACE^A$ is very powerful and the proof that $ATIME(n^{O(1)}) = PSPACE$ does not relativize [27] (for the definition in [25] Savitch's Theorem and the simulation of space by time also do not relativize, though in the definition in [30] they do). Oracle constructions using the [25] definition of relativized space appear (not surprisingly) in [25]. Oracle constructions using the [30] definition of relativized space appear in [13]. Recently J. Buss [9] proposed a definition of relativized space such that A is in $DSPACE^A(\log n)$, Savitch's Theorem relativizes, the simulation of space by time relativizes, and $ATIME(n^{O(1)}) = PSPACE$ relativizes.

COROLLARY 19. (*Using a definition of relativized space such that* $ATIME(n^{O(1)}) = PSPACE$ *relativizes.*) *Given a* $k \geq 1$ *there exist oracles A, B, and C such that,*

(1) $PSPACE^A \subsetneq EXP_k^A$,

(2) $EXP_k^B \subsetneq PSPACE^B$,

(3) $PSPACE^C$ *incomparable to* EXP_k^C,

(4) *For all oracles X* $PSPACE^X \neq EXP_k^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. Since $PSPACE = ATIME(n^{O(1)})$, relativizes we know that for all oracles X $PSPACE^X = ATIME^X(n^{O(1)})$. Using this, Corollary 18 yields the desired result. □

Valiant [34] defined the class of functions $\#P$ to be the set of all functions f such that there exists a polynomial bounded *NDTM M* such that

$$f(x) = \text{the number of accepting paths of } M \text{ on } x.$$

He proves that computing the permanent of a matrix, and other problems, are $\#P$-complete [33], [34]. $D\#P$ denotes the set of languages computed by polynomial bounded Turing Machines with an oracle to compute some functions in $\#P$. Aside from $P^{SAT} \subseteq D\#P$ little else is known about how $D\#P$ compares with the other complexity class. $D^A\#P$ denotes languages recognized by polynomial bounded oracle Turing Machines that have two oracle tapes, one to query A, and the other to make $\#P$-function calls. $D\#P^A$ denotes languages recognized by polynomial bounded oracle Turing Machines that can query any $\#P^A$ function. Angluin [1] has exhibited an oracle A such that $D\#P^A - (\Sigma_2^{p,A} \cup \Pi_2^{p,A}) \neq \emptyset$.

Some languages in $D\#P$ are hard, as the ability to call a $\#P$ function is powerful; however, such a language can not take too much time to recognize, as the machine used to recognize it is polynomially bounded. Our next corollary shows how the powerfulness of the questions that a $D\#P$ machine can ask contrasts with the limited time it has to work with.

COROLLARY 20. *There exist oracles A, B, and C such that,*

(1) $D^A\#P \subsetneq EXP_k^A$,

(2) $EXP_k^B \subsetneq D^B\#P$,

(3) $D^C\#P$ *incomparable to* EXP_k^C,

(4) *For all oracles X,* $D^X\#P \neq EXP_k^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. Since $NP^X \subseteq D^X\#P$ for all X, but a $D^X\#P$ machine cannot query exponentially long strings, constructions like the three in Theorem 9 will work to construct A, B and C. □

5. Conclusions. Nondeterministic, alternating, and $D\#P$ machines are powerful in one way; deterministic machines that can use a lot of time are powerful in another. The oracles constructed in this paper indicate that it will be hard to say how the two kinds of power relate to each other, in that recursion theoretic techniques will not suffice.

The languages L_1^C and L_2^C in Theorem 9 provide examples of the kinds of languages that may separate the two kinds of classes. Indeed, it looks like it would be very hard for a D^C machine to recognize L_1^C or for an N^C machine to recognize L_2^C *regardless* of what C is. This intuition is justified by the result that for the classes $D^{()}$ and $N^{()}$ defined in Theorem 9, for almost all oracles X, D^X is incomparable to N^X [12]. This also holds for all the pairs of classes mentioned in the corollaries. This may tempt one to believe that these classes are incomparable in the unrelativized case, but the result $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ proves this false. This result combined with [12] yields another counterexample to the Random Oracle Hypothesis (see [6] for the Hypothesis, [22] for the first refutation). It may be the case that if the classes involved

are above a certain time limit then the deterministic and nondeterministic classes are incomparable. On the other hand, all *natural NP*-problems are actually in EXP_1, so it may be the case that for low time bounds the deterministic classes are contained in the nondeterministic ones, and for higher ones, the reverse is true. If this is the case, it would be interesting to know where the turning point is, and if for some classes in between we have incomparability.

It is an open question whether our immunity results can be extended to bi-immunity. It seems hard to construct an oracle A such that a language L^A is bi-immune with respect to a nondeterministic time class. It is also open if an oracle can be constructed to force equality in Corollaries 10, 11, 14, and 15.

Acknowledgments. I wish to thank Harry Lewis and Albert Meyer for useful conversations on the topics discussed in this paper; Amihood Amir and Clyde Kruskal for proofreading; and an anonymous referee for pointing out some subtle mistakes in an older version of the proof of Theorem 1.

I would also like to thank both the University of Maryland and Harvard University for providing computer time.

REFERENCES

[1] D. ANGLUIN, *On counting problems and the polynomial-time hierarchy*, Theoret. Comput. Sci., 12 (1980), pp. 161-173.

[2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the P = ?NP question*, this Journal, 1 (1975), pp. 305-322.

[3] T. BAKER AND A. SELMAN, *A second step towards the polynomial hierarchy*, Theoret. Comput. Sci., 8 (1979), pp. 177-187.

[4] J. L. BALCAZAR, *Simplicity for relativized complexity classes*, manuscript.

[5] J. L. BALCAZAR AND U. SCHONING, *Bi-immune sets for complexity classes*, manuscript.

[6] C. G. BENNET AND J. GILL, *Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability*, this Journal, 10 (1981), pp. 96-113.

[7] R. BOOK, *Translational lemmas, polynomial time, and* $(\log n)^n$*-space*, Theoret. Comput. Sci., 8 (1976), pp. 177-187.

[8] ———, *On languages accepted in polynomial time*, this Journal, 1 (1972), pp. 281-287.

[9] J. BUSS, *Relativized alternation*, in Lecture Notes in Computer Science 223, Springer, Berlin-New York, 1986, pp. 66-76. (Structure in Complexity Theory—Proceedings of 1st Structure Conference.)

[10] M. DEKHTYAR, *On the relation of deterministic and nondeterministic complexity classes*, in Lecture Notes in Comput. Sci., 45, Springer, Berlin-New York, 1977, pp. 255-259.

[11] D. DU AND R. BOOK, *The existence and density of generalized complexity cores*, J. Assoc. Comput. Mach., to appear.

[12] W. I. GASARCH, *More on the random oracles hypothesis: What's true almost always is not necessarily so*, Tech. Rep. 1596, Univ. of Maryland, Baltimore, MD.

[13] ———, *Relativized space with immunity*, in Math. Systems Theory, Springer, New York-Berlin, to appear.

[14] W. I. GASARCH AND S. HOMER, *Relativizations comparing NP and exponential time*, Inform. and Control, 58 (1983), pp. 88-100.

[15] J. GRESKE AND J. GROLLMAN, *Relativizations of unambiguous and random polynomial time classes*, this Journal, 15 (1986), pp. 511-519.

[16] J. HASTAD, *Almost optimal lower bounds for small depth circuits*, Proc. 18th Annual ACM Symposium on the Theory of Computing, May 1986, pp. 6-20.

[17] H. HELLER, *On relativized exponential and probabilistic complexity classes*, manuscript.

[18] S. HOMER AND W. MAASS, *Oracle dependent properties of the lattice of NP sets*, Theoret. Comput. Sci., 24 (1983), pp. 279-289.

[19] J. HOPCROFT, A. K. CHANDRA, D. C. KOZEN AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[20] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[21] S. KURTZ, *Another oracle with strange properties*, manuscript, 1983.
[22] ———, *On the random oracle hypothesis*, Inform. and Control, 57 (1983), pp. 40-47.
[23] ———, *A Relativized Failure of the Berman-Hartmanis Conjecture*, Tech. Rep. Dept. of Computer Science, Univ. of Chicago, Chicago, IL.
[24] ———, *On Sparse sets in NP − P: Relativizations*, this Journal, 14 (1985), pp. 113-119.
[25] N. LYNCH, *Log space machines with multiple oracle tapes*, Theoret. Comput. Sci., 6 (1978), pp. 25-39.
[26] J. MAKOWSKY, personal communication.
[27] P. ORPONEN, *Complexity classes of alternating machines with oracles*, in Lecture Notes in Computer Science 154, Springer, Berlin-New York, 1983. Proceedings 10th ICALP Colloquium, Barcelona, Spain, July 1983.
[28] W. PAUL, N. PIPPENGER, E. SZEMEREDI AND W. TROTTER, *On determinism and nondeterminism and related problems*, Proceedings 24th IEEE Symposium of the Foundation of Computer Science, 1983, pp. 429-437.
[29] C. RACKOFF, *Relativized questions involving probabilistic algorithms*, J. Assoc. Comput. Mach., 29 (1982), pp. 261-268.
[30] W. RUZZO, J. SIMON AND M. TOMPA, *Space-bounded hierarchies and probabilistic computations*, J. Comput. System Sci., 28 (1982), pp. 216-230.
[31] U. SCHONING AND R. BOOK, *Immunity, relativization, and nondeterminism*, this Journal, 13 (1984), pp. 329-337.
[32] M. SIPSER, *On relativizations and existence of complete sets*, in Lecture Notes in Computer Sci., 140, Springer, Berlin-New York, 1982. Proc. 9th ICALP Conference, Aarhus, Denmark.
[33] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189-201.
[34] ———, *The complexity of enumeration and reliability problems*, this Journal, 3 (1979), pp. 410-421.
[35] C. WILSON, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169-181.
[36] A. C. C. YAO, *Separating the polynomial time hierarchy by oracles*, Proc. 26th IEEE Symposium on the Foundation of Computer Science, 1985, pp. 1-10.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

THE LEAST WEIGHT SUBSEQUENCE PROBLEM*

D. S. HIRSCHBERG† AND L. L. LARMORE†

Abstract. The least weight subsequence (LWS) problem is introduced, and is shown to be equivalent to the classic minimum path problem for directed graphs. A special case of the LWS problem is shown to be solvable in $O(n \log n)$ time generally and, for certain weight functions, in linear time. A number of applications are given, including an optimum paragraph formation problem and the problem of finding a minimum height B-tree, whose solutions realize improvement in asymptotic time complexity.

Key words. algorithms, dynamic programming

AMS(MOS) subject classifications. 68P05, 68Q25

1. Introduction. We define an instance of the *Least Weight Subsequence* problem (hereinafter referred to as the LWS problem) on $[a, b]$ as follows:

We are given a real-valued function *weight* (i, j), defined for all $a \leqq i < j \leqq b$. If $i_0, i_1, \cdots, i_t$ is a strictly monotone increasing sequence of integers in the range $[a, b]$, we define the *weight* of the sequence to be $\sum_{1 \leqq s \leqq t}$ *weight* (i_{s-1}, i_s).

The *single pair* LWS problem (or simply, LWS problem) is to find that monotone sequence of integers starting with a and ending with b which has minimum weight. The *all pairs* LWS problem is to solve the LWS problem for all pairs (a_0, b_0) such that $a \leqq a_0 \leqq b_0 \leqq b$.

We say that the weight function *weight* is *concave* if, for all $i_0 \leqq i_1 < j_0 \leqq j_1$,

$$\textit{weight}\ (i_0, j_0) + \textit{weight}\ (i_1, j_1) \leqq \textit{weight}\ (i_0, j_1) + \textit{weight}\ (i_1, j_0).$$

This concavity condition is exactly the *quadrangle inequality* introduced by Yao [6]. If *weight* is concave, we say it defines an instance of the *concave* LWS problem.

The LWS problem is equivalent to a minimum path problem for a weighted directed graph. Given an instance of the LWS problem, let $G = (V, E)$ be the directed graph with $V = \{0 \cdots n\}$ and $E = \{(i, j) \mid i < j\}$, and let *weight* be a weight function on the edges of G. Then the minimum path problem is to find a minimum weight path from 0 to n.

Conversely, given any instance P of the min path problem on directed acyclic graph $G = (V, E)$ with a weight function *weight*, choose a topological order on V. A solution to the LWS problem, where V has been numbered 0 to n in accord with the topological ordering, yields a solution to the minimum weight path problem on G.

Aside from theoretical interest, the LWS problem (and its solution) are useful in bringing to light improved solutions to a number of practical problems. In the ensuing sections, we discuss three algorithms for the LWS problem and their applications.

In § 2, we discuss the Traditional Algorithm, which solves the general LWS problem in quadratic time. In § 3, we introduce our Basic Algorithm, which solves the general concave LWS problem in $O(n \log n)$ time. In § 4, we describe an algorithm which we call the Scale-Assisted Algorithm, which solves the concave LWS problem in linear time, provided that the weight function is supported by a suitable scale function, and

* Received by the editors March 25, 1985; accepted for publication (in revised form) October 7, 1986. A preliminary version of this paper was presented at the 26th Annual Symposium on Foundations of Computer Science held in Portland, Oregon, October 21-23, 1985. An extended abstract of the paper appears in the Proceedings, pp. 137-143. © 1985 IEEE.

† Department of Information and Computer Science, University of California, Irvine, California 92717.

that a certain inequality can be solved in constant time. In § 5, we give some applications, including the paragraph breaking problem with quadratic penalties.

2. The Traditional Algorithm. In this section we describe a straightforward algorithm for solving the LWS problem, using dynamic programming. A variation of this Traditional Algorithm is used in TEX to solve the paragraph breaking problem [3].

Suppose we are given an instance of the LWS problem on $[0, n]$, defined by a weight function *weight*. Let $f(i)$ be the least weight of any subsequence which begins with 0 and ends with i. (Thus $f(0)=0$, and $f(n)$ is the weight of the solution to the LWS problem.) The function f exists abstractly. The Traditional Algorithm uses dynamic programming to construct an array $f[*]$ which contains the values of this abstract function.

We define an abstract function $g(i,j)=f(i)+weight\ (i,j)$, for $i<j$ integer in $[0, n]$. We may consider $g(i,j)$ to be a candidate value for $f(j)$, i.e., $f(j)$ is the minimum, over all $i<j$, of $g(i,j)$. During execution of the Traditional Algorithm, the value of $g(i,j)$ can be computed in constant time, provided $f[i]$ has already been evaluated.

Define *bestleft* (j) to be the second from the last element in a solution to the LWS problem restricted to $[0,j]$, i.e., *bestleft* $(j)=i$, where $f(j)=f(i)+weight\ (i,j)$. Since there need not be a unique least weight subsequence, there may be more than one equally good choice of *bestleft*, in which case, we allow one of them to be chosen arbitrarily. During execution of the algorithm, an array *bestleft* is defined which contains the values of the abstract function *bestleft*.

THE TRADITIONAL ALGORITHM

```
f[0] ← 0
for m from 1 to n do
  begin
    f[m] ← g(0, m)
    bestleft [m] ← 0
    for i from 1 to m − 1 do
      if g(i, m) < f[m] then
        begin
          f[m] ← g(i, m)
          bestleft [m] ← i
        end
  end
//  Comment: the array f has now been defined.
//  The remainder of the algorithm recovers the LWS using bestleft.
L ← (n)
m ← n
while m > 0 do
  begin
    m ← bestleft [m]
    prepend m to L
  end
return (f[n], L)
```

Analysis of the Traditional Algorithm. During each iteration of the main loop of the Traditional Algorithm, the value of $f[m]$ is computed, under the assumption that

the values of $f[i]$, for all $i < m$, have already been evaluated. The value of *bestleft* $[m]$ is simultaneously computed.

The computation of each $f[m]$ requires examining $f[i]$ for all $i < m$. Thus the Traditional Algorithm requires quadratic time.

Application of the Traditional Algorithm to the All Pairs LWS Problem. Only a slight modification is necessary to solve the all pairs problem. If the entire solution were given in output, it would consist of the approximately $n^2/2$ separate subsequences. Instead, the solution to the LWS problem for a pair (a, b) such that $0 \leqq a \leqq b \leqq n$ is implicitly stored in the values of $bestleft_a$. For any fixed a and b, the solution to the LWS problem for the pair (a, b) can be recovered in linear time using the pointer array $bestleft_a$. Thus, $\Theta(n^3)$ time and only $\Theta(n^2)$ space is required.

3. The Basic Algorithm. In this section, we introduce a new algorithm, which we call the *Basic* Algorithm since it forms the basis for the more complicated algorithms which are developed later.

We shall assume that we are given an instance of the concave LWS problem on $[0, n]$, defined by a weight function *weight*.

The Basic Algorithm makes use of an input-restricted deque D. At all times, D will be a subsequence of the integers $0, \cdots, n$. We refer to the least and greatest elements of the subsequence D as the *front* and *rear*, respectively. Three updating procedures are permitted on D: *Hire* (m), which appends a new element m to the rear of D; *Fire*, which removes the rear element of D; and *Retire*, which removes the front element of D.

In addition, we will need to access the second from the front element of D, which we call *front*2, and the second from the rear element of D, which we call *rear*2. These will, of course, be defined only when $|D| > 1$.

The deque D contains all current candidates for *bestleft* $[m]$, for all m which are yet to be considered. A new element joins D when it might be the value of *bestleft* in the future. The front element is removed (i.e., *Retire* is executed) when it is no longer possible for it to be *bestleft* for any future m, and an element is removed from the rear (i.e., *Fire* is executed) when it can be determined that it will *never* be the correct value of *bestleft* for any m. It is the determination of whether *Fire* should be executed that takes most of the time of the Basic Algorithm.

THE BASIC ALGORITHM

```
f[0] ← 0
D ← (0)
for m from 1 to n − 1 do
  begin
    f[m] ← g(front, m)
    bestleft [m] ← front
    while |D| > 1 and g(front2, m + 1) ≦ g(front, m + 1) do
      Retire
    while |D| > 1 and Bridge (rear2, rear, m) do
      Fire
    if g(m, n) < g(rear, n) then
      Hire (m)
  end
f[n] ← g(front, n)
bestleft [n] ← front
```

```
// Comment: the array f has now been defined.
// The remainder of the algorithm recovers the LWS using bestleft.
L ← (n)
m ← n
while m > 0 do
  begin
    m ← bestleft [m]
    prepend m to L
  end
return (f[n], L)
```

The key to the Basic Algorithm is the Boolean function *Bridge*. Intuitively, *Bridge* (i_0, i_1, i_2) if and only if i_1 can be ignored in the future because either i_0 or i_2 is always at least as good a choice for *bestleft*. Formally, for any $i_0 < i_1 < i_2$, *Bridge* (i_0, i_1, i_2) is false if and only if there exists some $k > i_2$ such that $g(i_1, k) < g(i_0, k)$ and $g(i_1, k) < g(i_2, k)$. Since *weight* is concave, *Bridge* can be evaluated by a binary search algorithm. We give an example of one such algorithm below.

```
BRIDGE (a, b, c)

if c = n then
    return (true)
else if g(a, n) ≦ g(b, n) then
    return (true)
else
  begin
    low ← c
    high ← n
    while high − low ≧ 2 do
      begin
        mid ← ⌊(low + high)/2⌋
        if g(a, mid) ≦ g(b, mid) then
            low ← mid
        else
            high ← mid
      end
    if g(c, high) ≦ g(b, high) then
      return (true)
    else
      return (false)
  end
```

The above *Bridge* Algorithm works as follows. Let *Right* be the set of k in $[c+1, n]$ for which $g(b, k) < g(a, k)$ and, similarly, let *Left* be the set of k in $[c+1, n]$ for which $g(b, k) < g(c, k)$. By definition, *Bridge* (a, b, c) is true iff *Right* and *Left* are disjoint. By the concavity condition, *Left* is a left subinterval of $[c+1, n]$ and *Right* is a right subinterval. Thus, their intersection is nonempty iff *Right* is nonempty and its minimum element is also in *Left*.

The first two steps of *Bridge* determine whether *Right* is empty. If so, *true* is returned. The next step uses binary search to find the minimum value of *Right* (stored in *high*), as it is known that this is a right subinterval. Finally, the correct value of the function is determined by checking whether *high* is in *Left*.

Analysis of the Basic Algorithm.

The Boolean Function Bridge. The value of *Bridge* (a, b, c) is false if and only if there exists $c < k \leqq n$ such that $g(b, k)$ is smaller than is either $g(a, k)$ or $g(c, k)$, i.e., if there exists some k where b is a strictly better candidate for *bestleft* $[k]$ than either a or c. We are interested in evaluating *Bridge* because if *Bridge* (*rear*2, *rear*, m) then *rear* must be removed from D (i.e., *Fire* must be executed) since it will not be the correct choice of *bestleft* $[k]$ for any future k.

Time Analysis of the Basic Algorithm. Each integer from 0 to n is entered onto the deque D exactly once. Since each execution of *Retire* or *Fire* decreases $|D|$ by one, the total number of executions of these two procedures cannot exceed $n+1$. The function *Bridge* must be evaluated once during each iteration of the main loop of the algorithm, plus one additional time after each execution of *Fire*, a total of not more than $2n$ times altogether. Each execution of *Bridge* requires $O(\log n)$ time. (We assume the reader is familiar with the technique of binary search.) All other parts of the basic algorithm require at most linear time, so the time requirement of the Basic Algorithm is $O(n \log n)$.

All Pairs Version. The all pairs LWS problem can be solved by iterating the basic algorithm n times.

Correctness of the Basic Algorithm. The proof of correctness is given in the Appendix.

4. The Scale-Assisted Algorithm. We now consider a method for speeding up the basic algorithm, that makes use of a *scale function* and a *difference-of-weight* function. We give an algorithm, called the Scale-Assisted Algorithm, a modification of the Basic Algorithm, which runs in linear time, provided a zero of the difference-of-weight function can be found in constant time.

More generally, the Scale-Assisted Algorithm runs in time $O(n \cdot \mathit{difftime})$, where *difftime* is the time required for finding a zero of the difference-of-weight function. Thus, the scale-assisted algorithm is asymptotically faster than the basic algorithm, if $\mathit{difftime} = o(\log n)$.

Let us consider an instance of the concave LWS problem on the sequence $0, \cdots, n$, where *weight* is the weight function. Let us suppose that we are given two real-valued functions

Scale (i)	strictly monotone increasing, defined for all integers $0 \leqq i \leqq n$,
diff (i, j, x)	defined for integers $0 \leqq i < j < n$ and real $x >$ *Scale* (j), monotone increasing in x for fixed i and j,

such that the following condition holds:

$$\mathit{diff}\,(i, j, \mathit{Scale}\,(k)) + \mathit{weight}\,(j, k) = \mathit{weight}\,(i, k) \quad \text{for all integers } 0 \leqq i < j < k \leqq n.$$

We then say that the weight function *weight* is *supported* by the scale function *Scale* and the difference-of-weight function *diff*.

The reader may easily verify that, in any instance of the concave LWS problem, the weight function is supported by some choice of scale and difference-of-weight functions. However, we do not start with a weight function and then find choices of *Scale* and *diff* which support it. Rather, we only apply the Scale-Assisted Algorithm in problems where *Scale* is given *prior* to *weight*, and *weight* depends on *Scale* in some simple way.

One such problem is the paragraph breaking problem, with quadratic penalty function, where hyphenation is allowed. We outline this application in § 5.

The only difference between the Basic Algorithm and the Scale-Assisted Algorithm is the meaning (and hence implementation) of the Boolean function *Bridge*. If $a < b < c$ are integers in $[0, n]$, *Bridge* (a, b, c) will be defined to be *true* if and only if, for all $x >$ *Scale* (c), at least one of the following two conditions holds:

(1) $f(a) - f(b) + diff(a, b, x) \leqq 0$,

(2) $f(b) - f(c) + diff(b, c, x) \geqq 0$.

In the Basic Algorithm, *Bridge* (a, b, c) is said to be true if there is no integer $d > c$ such that b is a strictly better candidate for *bestleft* $[d]$ than is either a or c. In the Scale-Assisted Algorithm, *Bridge* (a, b, c) is said to be true if there is no real number x which could be the value of *Scale* (d) for any possible $d > c$ for which b could be a better choice of *bestleft* than could either a or c.

Thus, it is not necessary to look at the various d, which is what the binary search implementation of *Bridge* does. Rather, it is only necessary to look at various choices of real x. In principal, there need be no time saving. But in practice (such as the paragraph breaking example in § 5), the function *diff* is frequently simple enough to allow evaluation of *Bridge* in constant time.

The Scale-Assisted Algorithm is identical to the basic algorithm. Only the *Bridge* function is implemented differently.

FUNCTION BRIDGE (a, b, c)—SCALE-ASSISTED ALGORITHM

```
if f(a) − f(b) + diff(a, b, Scale(n)) ≦ 0 then
    return (true)
else if f(b) − f(c) + diff(b, c, Scale(n)) < 0 then
    return (false)
else
  begin
    x0 ← lub {x | diff(a, b, x) ≦ 0}
    x1 ← glb {x | diff(b, c, x) ≧ 0}
    if x0 < x1 then
        return (false)
    else if x0 > x1 then
        return (true)
    else if diff(a, b, x0) > 0 and diff(b, c, x0) < 0 then
        return (true)
    else
        return (false)
  end
```

5. Some applications.

Example 1. *Airplane refueling.* Suppose that an airplane needs to fly between two given airports, which are distance R apart. Suppose there are $n-1$ optional refueling stops, at distances $x_1, \cdots, x_{n-1}$ from the departure point. For simplicity, assume that all these stops lie on the interval connecting the departure point and the destination. We can let $0 = x_0 < x_1 < \cdots < x_{n-1} < x_n = R$.

We suppose that the fuel consumption of the airplane is proportional to the weight of the plane, including unused fuel, and also that there is a fixed fuel cost of landing, a cost of taking off, and a landing fee which varies at the different stops. The problem of minimizing cost of the flight is then an instance of the concave LWS problem, where *weight* $(i, j) = e^{\alpha + \beta(x_j - x_i)} + L_j$, where L_j is the landing fee at the jth stop, and α and β are constants which absorb the other parameters of the problem.

The airplane refueling problem can be solved in linear time, using the Scale-Assisted Algorithm, by choosing the scale function to be simply the distance from the departure point, i.e., $Scale(i) = x_i$. For any $i < j$, we then have $diff(i, j, x) = C_{i,j} e^{\beta x} + L_i - L_j$, where $C_{i,j}$ does not depend on x. For fixed i and j, the solution to the equation $diff(i, j, x) = 0$ can thus be found in the amount of time it takes to compute a logarithm (which we take to be constant). Thus *Bridge* takes constant time, and the problem can be solved in linear time.

Example 2. *Optimum paragraph formation.* We are given a scroll of words, and each word consists of one or more syllables. Let n be the number of syllables, and let w_i be the length of the ith syllable. The space between two words will be considered to be attached to the last syllable in the earlier word. We wish to form this scroll into a paragraph, minimizing the total penalty. A paragraph may be defined by a break-sequence, consisting of a subsequence of $0, \cdots, n$, where each element of the break-sequence is the index of the last syllable on a line. (0 shall always be the first element of the breaksequence, being the last syllable on the fictitious 0th line.)

We assume that penalties are assigned as follows. There is an optimum length for a line, namely *lineopt*. There are also minimum and maximum lengths for a line, i.e., *linemin* and *linemax*. We assume that the penalty for a line being too short or too long is proportional to the square of the difference between the length of that line and the optimal length, except that the last line cannot be penalized for being too short. We also assume that there is fixed penalty for breaking any word.

A weight function *weight*, along with supporting scale function *Scale* and difference-of-weight function *diff*, can be assigned as follows:

Let w_i be the length of the ith syllable.

Let *spacelen* be the length of a space (between words).

Let *hyphenlen* be the length of a hyphen, assumed never to exceed the length of the rest of the word.

Let $Scale(i) = w_1 + \cdots + w_i - spacelen$, if the ith syllable is the last syllable of its word. Remember that w_i includes the length of a space.

Let $Scale(i) = w_1 + \cdots + w_i + hyphenlen$, if the ith syllable is not the last syllable of its word.

For real x, let $penalty(x) = A \cdot (x - lineopt)^2$ if $linemin \leqq x \leqq linemax$, where A is some positive constant; otherwise, let $penalty(x) = \infty$.

For $0 \leqq i < j \leqq n$, define $weight(i, j) = 0$, if $j = n$ and $Scale(n) - Scale(i) \leqq lineopt$, otherwise let $weight(i, j) = penalty(Scale(j) - Scale(i)) + \delta \cdot B$, where $\delta = 0$ if the jth syllable is at the end of its word, and $\delta = 1$ if the jth syllable is not at the end of its word. B is the hyphen penalty, a positive constant.

There is a problem in defining the function $diff(i, j, x)$ in some cases, since $\infty - \infty$ is an indeterminate form. There are two ways to resolve this problem. One way is to force the definition of *diff* in those cases in the unique way which causes the algorithm to work. Another, more elegant way, is to eliminate infinite penalties entirely, replacing them with very large penalties which maintain the concavity condition on *weight*. Accordingly, we define, in the following, a weight function which agrees with the previous definition in all finite cases, which is very large in cases where the previous definition requires ∞, and which is concave.

First, a value $\mathbf{M}$ is calculated which is larger than the total penalty of any actual paragraph. For example, we could let $\mathbf{M} = B \cdot n + A \cdot n\,(linemax - linemin)^2$. Let $\varepsilon > 0$ be less than the difference of the scale values of any two syllables, i.e.,

$\varepsilon < Scale\,(i) - Scale\,(i-1)$ for all i. Now, redefine

$$penalty\,(x) = A \cdot (x - lineopt)^2 \quad \text{if } linemin \leqq x \leqq linemax,$$
$$\mathbf{M} \cdot 2^{|x - lineopt|/\varepsilon} \quad \text{otherwise.}$$

Finally, define $weight\,(i, j)$ as before, but with the new formula for $penalty\,(Scale\,(j) - Scale\,(i))$. Then, for any $0 \leqq i < j < n$, and any real $Scale\,(j) < x \leqq Scale\,(n)$, $diff\,(i, j, x) = weight\,(Scale\,(i), x) - weight\,(j, x)$, and we observe that

$$\begin{aligned}
diff\,(i, j, x) &> 0 \quad \text{if } x - Scale\,(i) > linemax,\\
&< 0 \quad \text{if } x - Scale\,(j) < linemin \quad \text{and} \quad x < Scale\,(n),\\
&> 0 \quad \text{if } x = Scale\,(n) \text{ and } x - Scale\,(i) > lineopt,\\
&= 0 \quad \text{if } x = Scale\,(n) \text{ and } x - Scale\,(i) \leqq lineopt,\\
&= A \cdot [Scale\,(i)^2 - Scale\,(j)^2 - (Scale\,(i) - Scale\,(j)) \cdot (lineopt + 2x)]\\
&\qquad \text{otherwise.}
\end{aligned}$$

For fixed i and j, $diff\,(i, j, x)$ is a monotone increasing function of x, and is linear in the interval where it may be zero. Thus, it takes only constant time to compute the values of the least upper bound and greatest lower bound needed in the evaluation of *Bridge*. It follows that the optimum paragraph may be found in linear time.

There are additional recent discussions of the paragraph problem [2], [5].

Example 3. *Minimal Height* B-*Trees.* Diehr and Faaland [1] give an algorithm for finding a minimum height B-tree structure on a scroll (i.e., list) of words, in $O(n^3 \log n)$ time. We show how to use the all pairs version of the Basic Algorithm to solve the problem in $O(n^2 \log^2 n)$ time, and also how to use the all pairs version of the Scale-Assisted Algorithm to solve the problem in $O(n^2 \log n)$ time.

Given a scroll of n words, where the ith word has length w_i, we give a recursive definition of a B-tree structure of height h on the scroll.

A B-tree structure of height 0 is defined to be the empty subsequence of the scroll. A B-tree structure of height $h > 0$ consists of:

(1) A subsequence of the scroll, called the *boundary sequence.*

(2) A B-tree structure of height $h-1$ on the *boundary sequence* (itself a scroll).

We define a *page* of the B-tree structure to be either the substring consisting of all words between consecutive elements of the breaksequence (such a page is called a *leaf*), or a page of the B-tree structure on the *boundary sequence.* The *length* of a page is defined to be the sum of the lengths of all the words which constitute that page.

The *height* of each page can be recursively defined. A leaf has height 0. A page of height h is a page of height $h-1$ of the B-tree structure on the *boundary sequence.* Each word of the scroll lies in just one page; we define the *height* of a word to be the height of the page which contains that word.

The pages of a B-tree of height structure form a tree. A leaf consists of a substring of the scroll. If the leaf is the entire scroll, that leaf is the root, and is the unique page. Otherwise, the leaf is bounded on at least one end by a word of height 1. We then define the parent of the leaf to be the page of height 1 which contains that word. (Note that if both ends are bounded by words of height 1, they belong to the same page.) Parents of pages of height greater than 0 are defined recursively.

It is important to note that a page of a B-tree can be empty, and that an empty page can have either zero or one child.

In applications, B-trees normally have a restriction on the total length of any page.

Suppose that no page can have length exceeding some positive constant *pagemax*. We can then ask two questions about a given scroll consisting of n words:

(1) Does there exist a B-tree structure on that scroll?

(2) What is the minimum height B-tree structure on the scroll?

The answer to the first question is yes, if and only if $w_i \leqq pagemax$ for all i. If so, there certainly exists a B-tree structure of height $O(\log n)$. The Basic Algorithm can be used to solve an instance of the all pairs LWS problem for each h from 0 to the actual minimum height, constructing the minimum height B-tree structure in time $O(n^2 \log^2 n)$. We outline the method below.

The Diehr-Faaland Graph. It is convenient to enlarge the scroll by introducing fictitious words in positions 0 and $n+1$, where $w_0 = w_{n+1} = 0$. For any integer $h \geqq 0$ we define the hth *Diehr-Faaland graph* G_h of the scroll to be an acyclic directed graph where

(1) the vertices of G_h are the integers $[0, n+2]$, and

(2) the edges of G_h consist of all ordered pairs (i, j) such that $i < j$ and there exists a B-tree structure of height h on the scroll consisting of all words strictly between the ith and the jth words.

We note:

(1) $(i, i+1)$ is an edge of G_h.

(2) If (i, k) is an edge of G_h, then (i, j) is also an edge of G_h for all $i < j < k$.

(3) If (i, j) is an edge of G_h, then (i, j) is also an edge of G_{h+1}.

(4) The smallest h such that $(0, n+1)$ is an edge of G_h is the height of the minimum height B-tree structure on the scroll.

In passing, we note that condition 2 allows the directed graph G_h to be represented using only linear storage. For each $0 \leqq i < n+1$, let $farthest_j_h[i]$ be the largest j such that (i, j) is an edge of G_h. Any pair (i, j) is an edge of G_h if and only if $j \leqq farthest_j_h[i]$, thus only the array $farthest_j_h$ need be stored to represent G_h.

Our method is to first construct the graph G_0, then use the all pairs version of the Scale-Assisted Algorithm to construct G_{h+1} from G_h for all h. We stop when $(0, n+1)$ is an edge.

It is convenient to refer to $Sum(i) = w_1 + \cdots + w_i$. All values of *Sum* can be precomputed in linear time.

Construction of G_0. For any i, $farthest_j_0[i]$ is the largest j such that $Sum(j-1) - Sum(i) \leqq pagemax$. It takes linear time to evaluate $farthest_j_0$.

We now show how to construct G_{h+1} from G_h. Pick $\varepsilon > 0$, smaller than the smallest w_i. Let $\mathbf{M}$ be a sufficiently large number. For $0 \leqq i < j \leqq n+1$, let

$$weight(i, j) = \begin{cases} w_j & \text{if } (i, j) \text{ is an edge in } G_h, \\ \mathbf{M} \cdot 2^{Sum(j) - Sum(i)} & \text{otherwise.} \end{cases}$$

The value of $weight(i, j)$ when (i, j) is not an edge of G_h is a substitute for ∞. $\mathbf{M}$ must be chosen large enough that the edge (i, j) is never used in the LWS.

We now use the Basic Algorithm to compute the weight of the LWS between all pairs (i, j). Since the Basic Algorithm requires $O(n \log n)$ time, the all pairs version requires $O(n^2 \log n)$ time. Let Let $L_h(i, j)$ be the weight of this LWS. Finally, G_{h+1} may be defined as follows: (i, j) is an edge of G_{h+1} if and only if $L_h(i, j) \leqq pagemax + w_j$.

The method outlined above requires $O(n^2 \log^2 n)$ time to find the minimum height B-tree structure, and requires $O(n^2 \log n)$ storage. In fact, however, because of the special nature of the B-tree problem, the all pairs version of the Scale-Assisted Algorithm

may be used instead. Thus, the algorithm may always be sped up to require only $O(n^2 \log n)$ time. The reason is that it is unnecessary to use binary search to evaluate *Bridge*. In fact, because of the simple formula for *weight* (i, j), *Bridge* (a, b, c) is true if and only $f[b] \geqq f[c]$, and thus takes constant time to evaluate.

We note that Diehr and Faaland [1] also give a heuristic for finding a minimum height B-tree structure by optimizing the pagination of a scroll for each level of the tree. Their algorithm uses time $O(n \log n)$ per level. This has been improved to $O(n)$ per level [2], [4].

Appendix: Correctness of the Basic Algorithm. We assume the correctness of the Traditional Algorithm. The Basic Algorithm differs from the Traditional Algorithm only in the contents, but not the purpose, of the main loop. The purpose of the mth iteration of the main loop (for both algorithms) is to assign the correct values of $f[m]$ and *bestleft* $[m]$.

The Forward Property. Concavity of the weight function *weight* guarantees that, for $0 \leqq a < b < c < d \leqq n$, $g(b, c) < g(a, c) \Rightarrow g(b, d) < g(a, d)$. We call this the *forward property* of g. Intuitively, the forward property is that if b is superior to a as a candidate for *bestleft* $[c]$, it is also a superior candidate for *bestleft* $[d]$ for all $d > c$.

The Best-Candidates Condition. We say that the deque D satisfies the *Best-Candidates condition* up to m, which we denote by BC (m), if:

BC1 (m). For any $j > m$, there exists some $i \in D$ such that $g(i, j) \leqq g(k, j)$ for all $0 \leqq k \leqq m$. That is, there is some element of D which is at least as good a choice for *bestleft* as any other element in the range $[0, m]$.

BC2 (m). For any $i \in D$, there exists some $j > m$ such that $g(i, j) < g(k, j)$ for all $k \in D$, $k \neq i$. That is, there exists some element beyond m for which i is not only the best choice of *bestleft* $[j]$, but also the *strictly* best choice among the elements of D.

We show that the Best-Candidates Condition is a loop invariant, i.e., D satisfies BC (m) after m iterations of the main loop of the Basic Algorithm, for any $m < n$.

After 0 iterations of the main loop, $D = (0)$, and thus D satisfies BC (0).

Suppose that D satisfies BC $(m-1)$ before the mth iteration of the main loop. We need to show that the updating of D within that iteration causes D to satisfy BC (m). (We only consider $m < n$, since if $m = n$, no updating of D is done because it is no longer needed.)

It is helpful to realize that BC1 (m) is a positive condition, i.e., is fulfilled if D has "enough" elements, while BC2 (m) is a negative condition, i.e., is fulfilled if D has "not too many" elements. Precisely stated, BC1 (m) cannot be made false by adding elements to D, while BC2 (m) cannot be made false by removing elements from D.

When m is appended to D, D will satisfy BC1 (m), provided nothing is removed. We need to show that those items removed do not contribute to BC1 (m). When *Retire* is executed, *front* is removed from D. But this only occurs if *front*2 is at least as good a choice for *bestleft* $[m+1]$ (and hence, by the forward property of g, for all $k > m$) as *front*. Thus execution of *Retire* does not harm BC1 (m). When *Fire* is executed, *rear* is removed from D. But this only occurs if either *rear*2 or m (both of which will still be on D) is at least as good a choice for *bestleft* $[k]$ as *rear* for all $k > m$. Thus, execution of *Fire* does not harm BC1 (m).

In order to show that after the mth iteration of the main loop BC2 (m) is satisfied, we need to show that iteration removes every item which would cause BC2 (m) to fail.

For each $i \in D$ before the iteration, let $j_i > m-1$ be the element required by the condition BC2 $(m-1)$. Without loss of generality, j_i is the smallest such element. By

the forward condition on g, $i_0 < i_1 \Rightarrow j_{i_0} < j_{i_1}$. Suppose $i \in D$ before the iteration, but BC2 $(m) \Rightarrow i \notin D$. There are two possible reasons for this. Either $j_{front2} = m+1$, in which case $i = front$ and is deleted when *Retire* is executed, or m is as least as good a choice for *bestleft* $[j_i]$ as i itself. In the latter situation, by the forward condition, and by the condition BC2 $(m-1)$, m will be the best choice for *bestleft* $[j]$ for all $j_i \leqq j \leqq n$. Thus, *Fire* will be iterated, removing all elements of D from *rear* down to and including i. By the forward condition, when *Bridge* (*rear*2, *rear*, m) is false, BC2 (m) is satisfied.

Finally, we show that, since D satisfies BC $(m-1)$ before the mth iteration, $front = bestleft\ (m)$. Suppose not. Then $bestleft\ (m) = i < m$, for some $i \in D$, by BC1 $(m-1)$. By BC2 $(m-1)$, there exists some $j \geqq m$ such that $g(front, j) \leqq g(i, j)$. By the concavity condition on *weight*, $g(front, m) \leqq g(i, m)$, since $m \leqq j$, a contradiction.

Acknowledgments. The authors wish to thank Alan Friese and an anonymous referee for pointing out errors in an early draft of this paper.

REFERENCES

[1] G. DIEHR AND B. FAALAND, *Optimal pagination of B-trees with variable-length items*, Comm. ACM, 27 (1984), pp. 241-247.

[2] D. S. HIRSCHBERG AND L. L. LARMORE, *New applications of failure functions*, J. Assoc. Comput. Mach., to appear.

[3] D. E. KNUTH AND M. F. PLASS, *Breaking paragraphs into lines*, Software—Practice & Experience (1981), pp. 1119-1184.

[4] L. L. LARMORE AND D. S. HIRSCHBERG, *Efficient optimal pagination of scrolls*, Comm. ACM, 28 (1985), pp. 854-856.

[5] ———, *Breaking a paragraph into lines in linear time*, Proc. 22nd Annual Allerton Conference on Comm., Control, and Computing, Monticello, IL, October 1984.

[6] F. FRANCES YAO, *Efficient dynamic programming using quadrangle inequalities*, Proc. 12th Annual ACM Symposium on the Theory of Computing, April 1980, pp. 429-435.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

© 1987 Society for Industrial and Applied Mathematics
004

A COMMUNICATION-TIME TRADEOFF*

CHRISTOS H. PAPADIMITRIOU† AND JEFFREY D. ULLMAN†

Abstract. We show a nontrivial tradeoff between the communication c and time t required to compute a collection of values whose dependencies form a grid, i.e., value (i,j) depends on the values $(i-1,j)$ and $(i,j-1)$. No matter how we share the responsibility for computing the nodes of the $n \times n$ grid among processors, the law $(c+n)t = \Omega(n^3)$ must hold. Further, there must be a single path through the grid along which there are d communication steps, where $(d+1)t = \Omega(n^2)$. Depending on the machine organization, either law may be the more significant.

Key words. multiprocessing, parallel computation, tradeoff, communication, dag, diamond dag, lower bound

AMS(MOS) subject classification. 68Q25

1. Introduction. The coming use of multiprocessor systems necessitates the development of a general methodology for utilizing such systems efficiently, something like the methodology of algorithm design and analysis for single-processor systems. There is a considerable added difficulty, however, in that the underlying processor interconnection may now be part of the design, and different interconnections may cause performance to vary drastically. As a first step towards such a methodology, it seems important to identify certain parameters which capture the performance of parallel algorithms in a machine-independent fashion, and could therefore be used in optimizing machine architecture. We use *time* and *communication* as our principal performance criteria.

We model the computational problem to be solved as a directed acyclic graph (dag), with nodes corresponding to computed values, and arcs denoting dependencies (i.e., the children of a node are the values used to compute the node). Naturally, in most cases this dag is part of the algorithm design sought, but it seems reasonable, at least at this point, to consider the dag fixed and given. We model the calculation of such a dag in a multiprocessor system by assigning to each node one or more processors that compute the node. (Note: Our proofs could be simplified considerably if we assigned a unique processor to each node, but as we shall observe, allowing several processors to compute the same value can save communication at no additional time delay. It is an open problem whether recomputation can save more than a constant factor in time or communication.) The *time* to compute a dag is determined when we assign a schedule to the nodes and processors, subject to the obvious constraints that: (1) A node cannot be computed unless all its children have been computed at previous time units. (2) No processor can compute two different nodes at the same time. Thus, time is a measure of the processing time of the schedule, when communication delays are neglected.

Let us consider a schedule obeying rules (1) and (2). Given such a schedule, we can define a relation D of *dependence* between processor-node pairs. Suppose that processor P computes u, and v is a child of u. We say that (P, u) *depends* on the processor-node pair (P', v) if P' computed v before P computed u; if several such P' exist, we pick one for each (P, u) and v. The schedule together with D completely

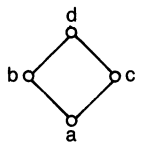
* Received by the editors September 4, 1984; accepted for publication (in revised form) December 9, 1986. This work was supported by National Science Foundation grants MCS-82-03405 and MCS-83-14575, and by Defense Advanced Research Projects Agency contract N0039-83-C-0136.

† Department of Computer Science, Stanford University, Stanford, California 94305.

determines how the dag is to be computed on a multiprocessor system. An arc $((P, u), (P', v))$ of D such that $P \neq P'$ is called a *communication* arc. The *communication* c required by a particular schedule and a corresponding D is the number of communication arcs in D. This definition of communication captures one measure of the actual communication in a multiprocessor system, namely the total message traffic generated by the algorithm. Another interesting measure is the total elapsed time due to communication, measured in the length of chains of interdependent messages. This matter is taken up in § 3.

We shall next define a useful estimate for the communication. Consider a node u such that, for some child v of u and processor P, P computes the value of u at a time when v has not yet had its value computed by P (we do not care whether or not v is later computed by P). Such a node u is called a *communication node.* Note that a communication node may contribute either 1 or 2 (or even more) to the total communication. However, since we obtain only an order-of-magnitude result for a dag with maximum outdegree two, we shall lower bound the communication by counting the communication nodes.

Example 1. We see below the 2×2 *diamond*, i.e., grid turned so the children of each node are below the node. Suppose we wish to compute this dag in three time units. We might use the two-processor schedule.



P_1	a	b	d
P_2		c	
time	1	2	3

This schedule uses two units of communication; the value a is passed from P_1 to P_2 after time 1, and the value of c is passed from P_2 to P_1 after time 2.

Note that if we allow P_2, which is idle at time 1, to compute a, then the communication is reduced to 1. This dag is an example where allowing more than one processor to compute a node is essential for minimizing the communication, given a time limit.

For many computation dags there is no real issue regarding the best way to trade communication for time.

Example 2 (Ernst Mayr). Consider dags that are complete binary trees of n nodes. If we allow k processors to cooperate on the computation, the time is constrained by $t \geqq n/k$. As each processor must either compute the root or compute a value used by some other processor, the communication is constrained by $c \geqq k-1$. Thus, $ct = \Omega(n)$ whenever more than one processor is used to compute the tree.

That is really all there is to the complete binary tree, because we can find a partition of the nodes so that both bounds, on c and t, are met simultaneously.

The diamond dag. The *diamond dag*, shown in Fig. 1, is an $n \times n$ square rotated 45 degrees, where the children of each node are the nodes immediately to the southeast and southwest, if they exist. It turns out that the diamond does not share with the tree of Example 2 the nice property that the best lower bounds on time and communication can be met simultaneously. Rather, there is a law about the product ct that is stronger than what is implied by the best lower bounds on c and t individually.

This interesting tradeoff law is one of a number of surprising properties of this dag. For example [4] showed a tradeoff involving time and the amount of external memory needed to compute the diamond on a chip. In [3] and [5], the "pyramid,"

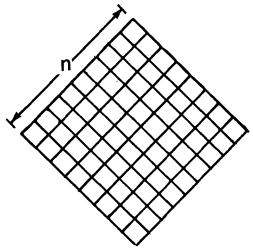


FIG. 1. *The diamond dag.*

which is the upper half of the diamond, is studied from the point of view of single processor computations. The pyramid represents the straightforward algorithm for estimating the nth derivative at a point, while the diamond represents the straightforward way of computing longest common subsequences (see [1], e.g.).

Suppose we compute an $n \times n$ diamond using $k \leqq n$ processors. We can argue that $t \geqq n^2/k$, since at least one processor must compute n^2/k nodes. We can meet this lower bound, to within a constant factor, by the *stripes* method shown in Fig. 2. There, the first processor begins to compute the rightmost diagonal of the first stripe. After n/k time units, it has reached the end of that diagonal, and the second processor can begin. Similarly, within n time units, all the processors are working on their stripes, and values will always be available so no processor is idle until it finishes its stripe.

Now, let us consider a lower bound on communication. If we divide responsibility for computation of the nodes among $k \geqq 2$ processors evenly, then each computes n^2/k nodes, and it is not hard to show that there must be among the nodes computed by any one of these k processors at least $\sqrt{n^2/k} = n/\sqrt{k}$ communication nodes. Thus, $c = \Omega(n\sqrt{k})$.

We can meet this lower bound by the partition suggested in Fig. 3. There, the number of communication nodes for each processor's territory is $O(n/\sqrt{k})$, so the total amount of communication is $O(n\sqrt{k})$. However, it is possible to show that only

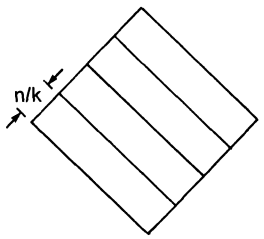


FIG. 2. *The fast "stripes" method.*

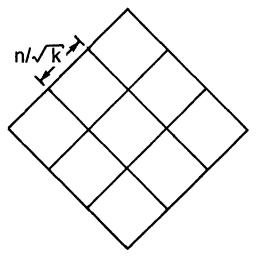


FIG. 3. *Low-communication method.*

one processor in any column can be active at any time, so the duration of the computation is $O(n^2/\sqrt{k})$, rather than $O(n^2/k)$ as we might have hoped.

Note that both Fig. 2 and Fig. 3 represent schedules where exactly one processor computes each node. However, no modification of Fig. 2 where we allow several processors to compute a node can improve the communication in Fig. 2, nor can it improve the time in Fig. 3.

2. The main theorem. We prove the following result.

THEOREM 1. *Any schedule and assignment of processors that computes all the nodes of an $n \times n$ diamond dag, uses communication c, and takes time t, where $(c+n)t = \Omega(n^3)$.*

If $t = n^2$ then the theorem is immediate. We shall therefore assume that $t < n^2$; that is, there is no time for a single processor to compute all nodes. In this case, we shall in fact show that $ct = \Omega(n^3)$. We shall prove this by a series of lemmas. The argument centers around a partition of the nodes of the diamond into squares of side m in the manner suggested by Fig. 4.[1]

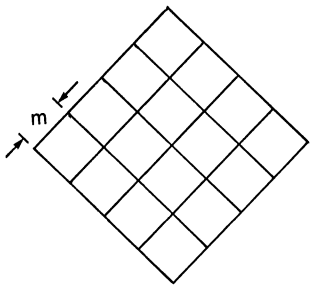


FIG. 4. *The partition for the proof of Theorem* 1.

In what follows, we focus on an arbitrary $m \times m$ *subdiamond* in the $n \times n$ diamond.

First note that the bottom node of a subdiamond must be computed before any other, and the top node of a subdiamond cannot be computed until all other nodes have been computed by at least one processor. The time between the first computation of the bottom node and the first computation of the top node will assume special significance; we call it the *window* for that subdiamond. Note that processors may continue to compute nodes of a subdiamond after the window has passed, but no node is computed by any processor before the window opens.

LEMMA 1. *Suppose that the duration of the window for some subdiamond is less than $4m^2/9$. Then there are at least $m/3$ communication nodes in that subdiamond.*

Proof. Consider the m "diagonals" (paths from lower right to upper left, say) of the subdiamond. If there are fewer than $m/3$ communication nodes, then there are $2m/3$ diagonals with no communication nodes on them. Any node on such a diagonal is computed by the same set of processors that compute the next node up in the diagonal (it may be computed by other processors as well). Similarly, there are at least $2m/3$ diagonals from lower left to upper right computed by the same set of processors. Now each of the latter diagonals meets each of the former, and as a result all $4m^2/9$ nodes in the intersection are computed by the same set of processors that computes the uppermost one (and perhaps by other processors as well). The processors in this

[1] However, there is no relationship between m and k, the number of processors used (compare with Fig. 3). In fact, it turns out that we shall ultimately wish to choose $m = 9t/n$, which makes a plausible partition of the diamond.

set must spend time at least equal to the number of nodes in the intersection, that is, at least $4m^2/9$, and thus the window can be no smaller. □

When we partition the diamond as in Fig. 4, we can naturally group the subdiamonds into columns, each consisting of the subdiamonds whose centers are aligned vertically. One column is illustrated in Fig. 5.

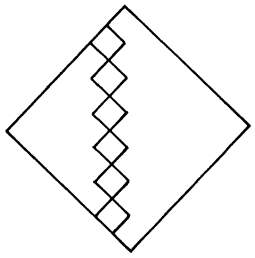


FIG. 5. *A column of subdiamonds.*

The following lemma is obvious:

LEMMA 2. *If a schedule of the diamond finishes in time t, then in any column, the sum of the durations of the windows for the subdiamonds in that column does not exceed t.*

If we divide the $n \times n$ diamond into $m \times m$ subdiamonds, there are $2n/m - 1$ columns, ranging in length from n/m in the center to 1 at the sides. There are at least n/m columns that each consist of $n/2m$ or more subdiamonds. By Lemma 2, the average window duration for a subdiamond in any of these columns is at most $t/(n/2m) = 2mt/n$. At least half the subdiamonds in any of these columns must have a window duration no more than twice the average, so we have proved.

LEMMA 3. *There are at least $n^2/4m^2$ subdiamonds with window durations of at most $4mt/n$.*

In order to apply Lemma 1 to these subdiamonds, we need to know that their window duration is at most $4m^2/9$, that is,

$$\frac{4mt}{n} \leqq \frac{4m^2}{9}$$

from which we derive $m \geqq 9t/n$.

If we take $m = 9t/n$, we can apply Lemma 1 to the $n^2/4m^2$ subdiamonds that we know exist by Lemma 3. Each of these, by Lemma 1, has at least $m/3$ communication nodes, so we know

$$c \geqq \left(\frac{n^2}{4m^2}\right)\left(\frac{m}{3}\right) = \frac{n^2}{12m} = \frac{n^2}{108(t/n)} = \frac{n^3}{108t}$$

from which $ct = \Omega(n^3)$ is immediate.

COROLLARY 1. *The stripes method of Fig. 2 gives, to within a constant factor, the smallest communication and the smallest number of processors for a given time limit. The method of Fig. 3 gives the smallest communication, but not the smallest number of processors, for a given time limit.*

3. Bound on the communication delay. We next study a tradeoff between processing time and *communication delay*. Let us consider a schedule obeying rules (1) and (2) in the introduction, and the dependence relation D among processor-node pairs. The *communication delay* d of a computation is the largest possible number of communication arcs in any path in D.

THEOREM 2. *If d is the communication delay of a schedule for the diamond, then* $(d+1)t = \Omega(n^2)$.

Proof. In the case where no recomputation is allowed, there is a very simple proof of Theorem 2, directly from Theorem 1: Since all nodes of the diamond can be covered by $O(n)$ paths from the source to the sink, and since there are $\Omega(n^3/t)$ communication nodes on the diamond, one of these paths must have $\Omega(n^2/t)$ communication nodes on it, and this completes the proof. Unfortunately, this simple argument does not seem to generalize easily to the case in which there may be recomputation of nodes, because in this case it is not true that all nodes of the dependence dag D can be covered by $O(n)$ paths. We next present a proof of Theorem 2 in the case in which recomputation is allowed.

Let us define the distance from node x of the diamond to node y to be the smallest number of edges going from lower left to upper right that must be used in order to reach y from x. Consider a pair (P, a), where a is a node on the main (vertical) diagonal of the diamond (see Fig. 6, ignoring subscripts for the time being), and P is a processor that computes a. Consider the nodes of the dag on which a depends, are also computed by P (that is, all nodes b such that there is a path from (P, a) to (P, b) in D free of communication arcs), and whose distance to some node of the main diagonal is at most half their distance to a (see the singly and doubly shaded area in Fig. 6). Define $s(a)$ to be the largest integer so that all these nodes with distance to a less than $s(a)$ are computed by the same processor P. Notice that, by the definition of $s(a)$, there will always be a communication arc $((P, b), (P', c))$ with c just beyond the lower left boundary of the shaded region (unless, of course, the shaded region extends to the lower edge of the diamond). Also, $s(a)$ could be one, in which case the shaded region would contain just a.

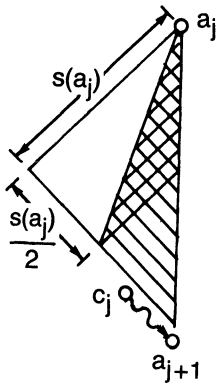


FIG. 6. *The proof of Theorem* 2.

We are going to define a sequence of such nodes a_j, as follows: a_1 is the top node of the diamond, without loss of generality computed by just one processor, P_1. For a general node a_j in the sequence and a processor P_j among those that compute a_j, consider the diamond-shaped area with side s_j, and, if the lower boundary of this area is not the boundary of the diamond, also consider the communication node c_j on the other side of the boundary, computed by some processor Q_j. We let a_{j+1} and Q_{j+1} be the node on the main diagonal, and the corresponding processor, so that the pair (c_j, Q_j) depends on (a_{j+1}, P_{j+1}). We continue until the construction stops, after m steps, because the lower boundary of the diamond has been reached.

Notice that, by our construction, each node in the *doubly shaded* region (see Fig. 6) that corresponds to a_j—call it R_j—depends on each node of R_{j+1} and thus, if by $|A|$ we denote the area of A,

$$t \geqq \sum_{j=1}^{m} |R_j| = \frac{1}{8} \sum_{j=1}^{m} s(a_j)^2.$$

Also, notice that $\sum_{j=1}^{m} s(a_j) = n$. Finally, the m communication arcs $((P_j, b_j), (P'_j, c_j))$ are by construction on the same path in D, and thus $d \geqq m$. We conclude from these three inequalities that $dt \geqq \frac{1}{8}n^2$. The case $d = 0$ (no communication whatever) is obvious. □

4. Applications and further remarks. Let us consider a computation of the diamond dag (which we may by Corollary 1 assume is by the stripes method) using a collection of processors linked by a local area network. Such a network has a communication cost that involves a large amount of overhead to open communication between two processors, plus a smaller cost to communicate each of a number of values once the overhead is paid. Theorem 2 says that there must be some path along which $d = \Omega(n^2/t)$. Thus, d overhead charges must be paid. On the other hand, Theorem 1 implies a lower bound on the total number of values communicated, and therefore, no matter how many values are bundled into one message, Theorem 1 implies a lower bound on the amount of time spent shipping values across the net. Depending on the overhead and on n, either Theorem 1 or Theorem 2 may put a more stringent limit on the number of processors that may effectively share the calculation of the diamond dag.

For a more concrete example, suppose that the overhead is C, constant and independent to the traffic in the network. If we use k processors, then the "real" time needed for the computation is something like

$$T = \frac{n^2}{k} + C\frac{n^2}{n^2/k}$$

(constants are ignored in these calculations). The first term is the delay due to computation time, the second is the total communication delay, by Theorem 2. The optimum number of processors to use is $k \approx n/\sqrt{C}$, for a total time $T \approx n\sqrt{C}$.

Usually the communication overhead depends on the traffic of the network; for example, the ethernet has a performance characteristic of the form $C(\tau) = 1/(b - \tau)$, where b is a cut-off traffic, and τ is the traffic measured in messages exchanged in unit time. If we assume that our computation will distribute traffic evenly throughout its duration (as the stripes method nearly does), then the "real" time becomes

$$T = \frac{n^2}{k} + \frac{n}{n^2/k} \Big/ \left(b - \frac{n^3}{(n^2/k)^2}\right).$$

The first term is again the computation time, while the second captures delays due to communication. The numerator is the number of distinct message transmissions, by Theorem 2, while the denominator is the limit b minus the average message traffic (by Theorem 1, n^3/t^2). The optimum choice of number of processors is $k \approx \sqrt{nb}$, for $T \approx n^{3/2}/b$.

From this discussion, we observe that results such as Theorems 1 and 2 do lead to a kind of detailed analysis of the efficient use of multiprocessors to solve a particular problem, as well as its limitations, that would be impossible without these results. We hope that such results, for wider classes of problems, will lead to a theory of parallel algorithms nearly as comprehensive as the one we have for sequential ones.

It is interesting to consider what other dags exhibit communication-time tradeoffs similar to the diamond. We already mentioned the $ct = \Omega(n)$ result of Mayr for the complete binary tree. Maria Klawe and Mike Paterson have shown a $ct = \Omega(n^4)$ result for the *dynamic programming dag*, where the children of node (i, j) are all the nodes (k, j), $k > i$ and (i, l), $l < j$. This graph models the computation of many "dynamic programming" algorithms [1], and can be calculated in $O(n^3)$ sequential time. Andivahis has improved this tradeoff to $c^2 t = \Omega(n^7)$ [2]. An important question that remains open is to state generalized such tradeoffs, that hold for any dag, and involve certain graph-theoretic parameters of the dag (like the bisection width used for VLSI lower bounds [6]). Another interesting question is deriving a more detailed tradeoff, involving time, number of messages, and *granularity* of messages, i.e., number of values that can be sent in a single message.

Note added in proof. There are some more recent results on the problem of analyzing dags in terms of their time and communication requirements. Jayasimha and Loui [8] show a $ct = \Omega(n^3)$ tradeoff for a dag that corresponds to the solution of a triangular system of equations. Afrati, Papadimitriou and Papageorgiou [7] show that the problem of finding, for a given dag, a schedule which minimizes time, and also minimizes communication as a secondary criterion, is NP-complete, even if the dag is a tree, and the number of processors to be used is either given or open (the problem is polynomial if the amount of communication is fixed). Finally, Papadimitriou and Yannakakis [9] develop a polynomial algorithm which, for any dag, calculates within a factor of two the optimum weighted sum $t + \lambda d$, for any constant λ, when no bound on the number of processors is specified.

Acknowledgments. We would like to thank several people for comments on this work and on the earlier proof of a weaker theorem that assumed no node was computed by more than one processor, including Dimitri Andivahis, Peter Hochschild, Don Knuth, Harry Mairson, Ernst Mayr and Mike Paterson. Also, many thanks to two anonymous referees for many insightful comments, criticisms and suggestions for improving the presentation, and especially one of them for suggesting the present proof of Lemma 1.

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] D. ANDIVAHIS, *Time-communication tradeoff in multiprocessing systems*, Diploma dissertation, National Technical University of Athens, Athens, Greece, 1984. (In Greek.)

[3] S. A. COOK AND R. SETHI, *Storage requirements for deterministic polynomial time recognizable languages*, Proc. 6th Annual ACM Symposium on the Theory of Computing, May 1974, pp. 33–39.

[4] J.-W. HONG AND H.-T. KUNG, I/O *complexity: the red-blue pebble game*, Proc. 13th Annual ACM Symposium on the Theory of Computing, May 1981, pp. 326–333.

[5] M. M. KLAWE, *A tight bound for black and white pebbles on the pyramid*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, November 1983.

[6] C. THOMPSON, *A Complexity Theory for* VLSI, Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[7] F. AFRATI, C. H. PAPADIMITRIOU AND G. PAPAGEORGIOU, *Scheduling dags to minimize time and communication*, Technical Report, National Technical University of Athens, Athens, Greece, 1985.

[8] D. N. JAYASIMHA AND M. C. LOUI, *The communication complexity of parallel algorithms*, Technical Report CSRD 629, University of Illinois at Urbana-Champaign, 1986.

[9] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Towards an architecture-independent analysis of parallel algorithms*, manuscript, 1987.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics
005

THE DISCRETE GEODESIC PROBLEM*

JOSEPH S. B. MITCHELL†, DAVID M. MOUNT‡ AND CHRISTOS H. PAPADIMITRIOU§

Abstract. We present an algorithm for determining the shortest path between a source and a destination on an arbitrary (possibly nonconvex) polyhedral surface. The path is constrained to lie on the surface, and distances are measured according to the Euclidean metric. Our algorithm runs in time $O(n^2 \log n)$ and requires $O(n^2)$ space, where n is the number of edges of the surface. After we run our algorithm, the distance from the source to any other destination may be determined using standard techniques in time $O(\log n)$ by locating the destination in the subdivision created by the algorithm. The actual shortest path from the source to a destination can be reported in time $O(k+\log n)$, where k is the number of faces crossed by the path. The algorithm generalizes to the case of multiple source points to build the Voronoi diagram on the surface, where n is now the maximum of the number of vertices and the number of sources.

Key words. shortest paths, computational geometry, geodesics, Dijkstra's algorithm

AMS(MOS) subject classification. 68E99

1. Introduction. Recent research in the algorithmic aspects of robot motion and terrain navigation has resulted in a number of interesting variants of the shortest path problem. The problem of determining the shortest path between two points in three dimensions in the presence of polyhedral obstacles seems to be very hard, and only extremely inefficient algorithms [16], [17] or approximation algorithms [14] are known for it. The two-dimensional special case with polygonal obstacles can be solved easily in $O(n^2 \log n)$ time by constructing the *visibility graph* [4], [6], [8], [9], [17] and $O(n \log n)$ is attainable in special cases when shortest paths possess certain monotonicity properties [5], [9], [17] (here, n is the total number of vertices in the polygonal obstacles). Recently, an algorithm for the two-dimensional shortest path problem which runs in time $O(nk+n \log n)$ (where k is the number of disjoint simple polygonal obstacles) has been announced [16].

In this paper we examine a special case of the three-dimensional shortest path problem called the *Discrete Geodesic Problem.* We are given two points which lie on the surface of a given polyhedron, and we are asked to find the shortest path between them which lies on the polyhedral surface. The geodesic problem is of considerable interest in terrain navigation, where a moving vehicle is bound to move along a surface that could be modeled by a polyhedron. Notice that this measure of distance may not be the shortest distance in three-space which avoids the polyhedron, since we are constrained to travel along the surface. To see why this is a special case of the three-dimensional problem, consider two polyhedral obstacles, namely the open polyhedron and the complement of its closure; a path between two points which avoids both polyhedra is exactly a path on the polyhedral surface. On the other hand, the

* Received by the editors April 29, 1985; accepted for publication (in revised form) December 5, 1986. This research was partially supported by a grant from the Hughes Aircraft Company.

† Department of Operations Research, Stanford University and Hughes Artificial Intelligence Center, Stanford, California 94305. This author was supported by a Howard Hughes Doctoral Fellowship and is affiliated with the Hughes Artificial Intelligence Center, Calabasas, California.

‡ Department of Computer Science, University of Maryland, Baltimore, Maryland 21201. This author was partially supported by the Air Force Office of Scientific Research under contract F-49620-83-C-0082.

§ Department of Operations Research and Department of Computer Science, Stanford University, Stanford, California 94305.

problem we solve is a generalization of the two-dimensional problem, since any polygonal obstacle can be simulated by a tall prism emanating from the plane whose base is the given polygon.

The shortest path problem on a surface was first posed in [17], where an $O(n^3 \log n)$ algorithm was given for the case of *convex* polyhedra. Here n is a measure of the complexity of the scene, which in our case we may take as the number of edges, say, of the polyhedral surface. Mount [10] has given an algorithm for convex polyhedra which improves the running time to $O(n^2 \log n)$. For *nonconvex* polyhedra, [13] gives an $O(n^5)$ algorithm.

We improve these results as follows: We handle general (possibly nonconvex, and even of higher genus) polyhedra, and we improve the time bound to $O(n^2 \log n)$. Furthermore, we solve the single-source shortest path problem (and by a simple extension, the multiple-source shortest path problem; see [11]), creating a subdivision for a given source point so that we can find the length of the shortest path to any destination point simply by locating it in the subdivision. Thus, in $O(\log n)$ time, the length of the shortest path is determined to any other destination, and the shortest path can then be listed in time $O(k)$, where k is the number of edges crossed by the path. Our algorithm therefore generalizes the work in [4] and [17] for the two-dimensional shortest path problem among obstacles. We concentrate here on the case of *bounded* polyhedra; however, the generalization of the algorithm to handle unbounded faces is straightforward.

Our algorithm uses a technique we call "continuous Dijkstra," as it closely resembles in structure the famous algorithm of Dijkstra for finding shortest paths in a graph [1]. Edges of the polyhedron behave like nodes of a graph, except that here there is no unique distance from the source to an edge. Instead, there is a *function* that serves as a label for a node, and we keep track of a discrete description of the minimum such function. This involves keeping track of "intervals of optimality" on each edge, which subdivide the edge into regions for which the shortest path to points in the region have the same discrete structure, passing through the same sequence of vertices and edges. These intervals are similar to the notion of "slices" used in [17].

2. Preliminaries. We are given a polyhedral surface, $\mathcal{S}$, specified by a set of faces, edges, and vertices, with each edge occurring in two faces and two faces intersecting either at a common edge, a vertex, or not at all. We consider faces to be *closed* polygons (they include their boundaries) and edges to be closed line segments (they include their endpoints, which are vertices). We are also given two special points s and t (the *source* and *destination*, respectively). Without loss of generality, we assume that all faces are triangles (since polygons may be triangulated in time $O(n \log n)$ [2] and the number of edges introduced is linear in the number of vertices) and that s and t are vertices of the polyhedral surface. We are asked to find the shortest path from the source to the destination which lies completely on the surface.

DISCRETE GEODESIC PROBLEM (DGP).

Instance: Two points s and t on a polyhedral surface $\mathcal{S}$, which has been triangulated.

Question: Find a shortest path in the Euclidean metric from s to t such that the path stays on the surface $\mathcal{S}$.

In fact, our algorithm solves the following query form of the problem.

SINGLE-SOURCE DISCRETE GEODESIC PROBLEM (SS-DGP).

Instance: Two points s and t on a polyhedral surface $\mathcal{S}$, which has been triangulated.

Question: Build a structure which allows one to compute a shortest path from s to any query point t such that the path stays on the surface $\mathcal{S}$.

Thus, we are able to find the shortest path between the source and *any* point on the surface, and we know of no algorithm which solves DGP in less time (worst-case, asymptotic) than SS-DGP. Consequently, in our exposition we shall not concern ourselves with a specific destination point.

Two faces f and f' are said to be *edge-adjacent* if they share a common edge e. A *sequence of edge-adjacent faces* is a list of one or more faces $\mathcal{F}=(f_1, f_2, \cdots, f_{k+1})$ such that face f_i is edge-adjacent to face f_{i+1} (sharing common edge e_i) (see Fig. 1). We then refer to the (possibly empty) list of edges $\mathcal{E}=(e_1, e_2, \cdots, e_k)$ as an *edge sequence* and to the vertex of face f_1 which is opposite e_1 as the *root*, $r(\mathcal{E})$, of $\mathcal{E}$ ($r(\varnothing)$ is undefined). If no face (resp., edge) appears more than once in $\mathcal{F}$ (resp., $\mathcal{E}$), then we call the sequence *simple*.

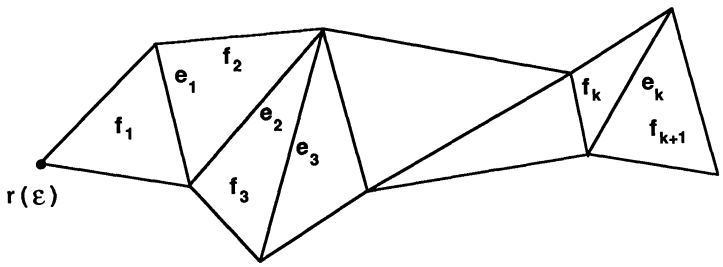


FIG. 1. *A sequence of edge-adjacent faces.*

Each face has a two-dimensional coordinate system associated with it. If faces f and f' are edge-adjacent sharing edge e, we define the *planar unfolding of face f' onto face f* as the image of points of f' when rotated about the line through e into the plane of f such that the points of f' fall on the *opposite* side of e to points of f (i.e., do not unfold face f' *on top of* face f). Points in the planar unfolding of f' onto f are written in the coordinate system of face f. Extending this notion, we say that we *unfold* an edge sequence $\mathcal{E}=(e_1, e_2, \cdots, e_k)$ as follows: Rotate f_1 around e_1 until its plane coincides with that of f_2, rotate f_1 and f_2 around e_2 until their plane coincides with that of f_3, continue in this way until all faces $f_1, f_2, \cdots, f_k$ lie in the plane of f_{k+1}. The resulting *planar unfolding along* $\mathcal{E}$ represents points of faces $f_1, f_2, \cdots, f_k$ in the coordinate system of face f_{k+1}. If x is a point on face f_i ($1 \leqq i \leqq k+1$), then let $U_{\mathcal{E}}(x)$ be the *unfolded image of x along* $\mathcal{E}$ (written in the coordinate system of face f_{k+1}).

A *path* on $\mathcal{S}$ will always be understood to be a *simple* (that is, not self-intersecting) path whose intersection with any face is a union of disjoint line segments. A *geodesic path* is a path which is locally optimal and cannot, therefore, be shortened by slight perturbations. We say that path p *connects* edge sequence $\mathcal{E}=(e_1, e_2, \cdots, e_k)$ if p consists of segments which join interior points of $e_1, e_2, \cdots, e_k$ (in that order). (Note that such a path cannot pass through the endpoint of any e_i, for otherwise two distinct consecutive edges of $\mathcal{E}$ would have to be collinear.) Path p *goes through* edge sequence $\mathcal{E}$ if it has a subpath which connects $\mathcal{E}$ (note that it may have many such subpaths). If p connects edge sequence $\mathcal{E}$, then the *planar unfolding of p along* $\mathcal{E}$ is simply the unfolded image of p along $\mathcal{E}$.

3. Characterization of geodesics and optimal paths. We begin a characterization of geodesics and optimal paths with a simple existence lemma stating that geodesics and optimal paths *exist* (but they may not be *unique*).

LEMMA 3.1. *There exists a geodesic path from s to any other point $x \in \mathcal{S}$. Furthermore, among the geodesic paths from s to x there exists at least one of minimal length.*

We have assumed that paths are simple, so they do not go through any point more than once. (This is certainly true for shortest paths.) Optimal paths also have the property that they do not go through any face more than once, as the following lemma states.

LEMMA 3.2. *The intersection of an optimal path p with a face f is a (possibly empty) line segment.*

Proof. If $p \cap f \neq \emptyset$, then let y be the *first* point of p on f, and let y' be the *last*. Then the part of p from y to y' must also be optimal (otherwise, p could be improved). The shortest path on $\mathcal{S}$ from y to y' is the line segment connecting them. Thus, the intersection of p with f must be a line segment. □

Note that it is possible for the intersection of a *geodesic* path with a face to be a union of many disjoint line segments. For example, if one tightly wraps a string many times around a long thin rectangular bar, the resulting path of the taut string is locally optimal but clearly not globally optimal. This also shows that not all geodesics are optimal paths and that there can be infinitely many geodesic paths (although only finitely many of them can pass through each face at most once).

We now state a simple property of geodesics (observed by [7] and [17]) which says that if one unfolds a geodesic path along an edge sequence, it becomes a straight line segment.

LEMMA 3.3. *If p is a geodesic path which connects the edge sequence $\mathcal{E}$, then the planar unfolding of p along the edge sequence $\mathcal{E}$ is a straight line segment.*

Proof. If the image were *not* a straight line segment, then there exist segments $\overline{\alpha x}$ and $\overline{x\alpha'}$ of p such that x is interior to edge $e = f \cap f' \in \mathcal{E}$ and the angle $\angle \alpha x \alpha'$ is not equal to π. Then, there exists a "shortcut" $\overline{\beta\beta'}$ for p, with $\beta \in \alpha x$, $\beta' \in x\alpha'$, and segment $\overline{\beta\beta'}$ lies entirely within the faces f and f'. This contradicts the local optimality of p. □

In the case of convex polyhedra, an optimal path will pass through no vertices other than its endpoints, and this property plays a critical role in the algorithm of [17]. In the case of nonconvex polyhedra, however, an optimal path may pass through many other vertices. For example, in Fig. 2 the optimal path from s to t goes through vertex v. (Vertex v has the property that the sum of the angles at v on the faces adjacent to v is greater than 2π.) The optimal path may, in fact, follow a long chain of adjacent edges (see Fig. 3).

We can characterize the way in which a geodesic path passes through a vertex. If path p passes through vertex v while going from face f to face f', then let $\overline{\alpha v}$ be the segment of p on f leading into v, and let $\overline{v\alpha'}$ be the segment of p on f' leading

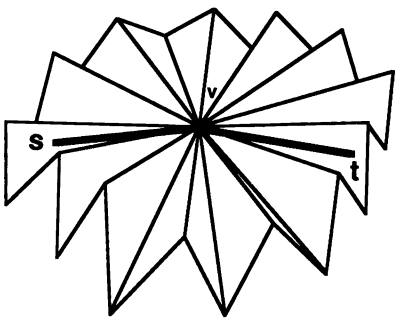


FIG. 2. *An optimal path going through a vertex.*

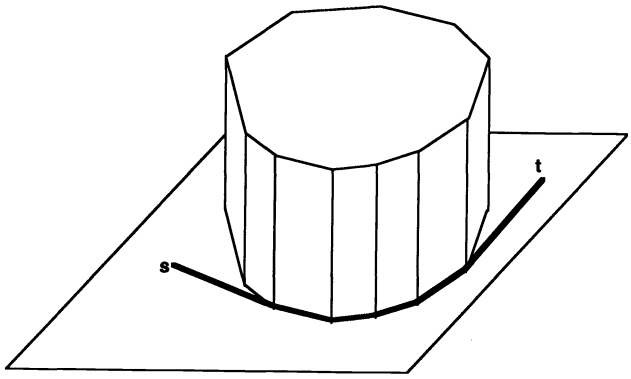


FIG. 3. *An optimal path following a chain of edges.*

out of v. Define the *clockwise* angle of p at v as the angle *from* $\overline{\alpha v}$ *to* $\overline{v\alpha'}$ when the faces containing v from f to f' (inclusive) *in a clockwise order about* v are "flattened" (simply add the angles formed by the edges incident on v; this may give you an angle far greater than 2π). Similarly, define the *counterclockwise* angle from $\overline{\alpha v}$ to $\overline{v\alpha'}$. Define the *angle made by path p through vertex v* as the minimum of these two angles. We then have the following.

LEMMA 3.4. *If a geodesic path p passes through vertex v, then the angle made by p at v is* $\geqq \pi$.

Proof. Assume that the angle is *less than* π and that (without loss of generality) it is clockwise. When the faces $f_1, f_2, \cdots, f_k$ are flattened about v, we get the situation depicted in Fig. 4. Now note that there exists a point β on the segment $\overline{\alpha v}$ and a point β' on the segment $\overline{v\alpha'}$ such that the segment $\overline{\beta\beta'}$ lies entirely in the set of flattened faces $f, f_1, f_2, \cdots, f_k, f'$. Clearly, the segment $\overline{\beta\beta'}$ represents a "short cut" for the path through v, so p cannot be a geodesic. □

The subpath of p between any two consecutive vertices v and v' connects some edge sequence $\mathscr{E}$ with root $v = r(\mathscr{E})$ (if $\mathscr{E} = \varnothing$, then v and v' are the endpoints of some

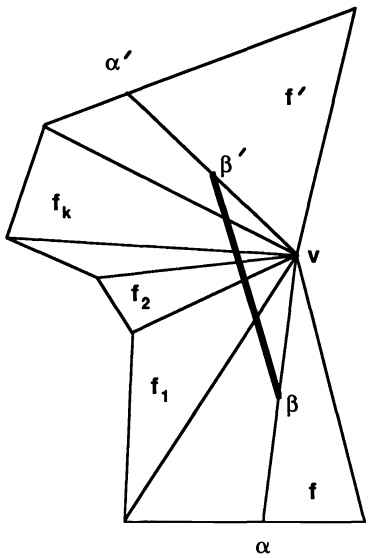


FIG. 4. *A shortcut for a path through a vertex.*

edge e which is contained in p). If p is geodesic, then by Lemma 3.3 the subpath from v to v' is completely determined by giving the edge sequence $\mathscr{E}$. Thus, we can describe a geodesic path p from s to x by writing it as a list, $p = (v_1 = s, \mathscr{E}_1, v_2, \mathscr{E}_2, v_3, \cdots, v_K, \mathscr{E}_K, x)$, where $v_1, v_2, \cdots, v_K$ are the vertices (in order) through which p passes, and $\mathscr{E}_1, \mathscr{E}_2, \cdots, \mathscr{E}_K$ are (possibly empty) edge sequences which are connected by p. Note that $v_i = r(\mathscr{E}_i)$ for each i such that $\mathscr{E}_i \neq \emptyset$. We call v_K the *root of path p* and $\mathscr{E}_K$ the *last* edge sequence of p. For geodesic paths p, the edge sequences $\mathscr{E}_i$ may not be simple (e.g., recall the example of the string wound around the rectangular bar); however, if p is *optimal*, then each $\mathscr{E}_i$ *must* be simple and no edge is contained in more than one of the edge sequences $\mathscr{E}_i$, $1 \leqq i \leqq K$ (otherwise, p would enter a face more than once, violating Lemma 3.2).

In summary, we have the following characterization of geodesics and optimal paths on (possibly) nonconvex surfaces.

LEMMA 3.5. *The general form of a geodesic path is a path which goes through an alternating sequence of vertices and (possibly empty) edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of the path passing through a vertex is greater than or equal to π. The general form of an optimal path is the same as that of a geodesic path, except that no edge can appear in more than one edge sequence and each edge sequence must be simple.*

4. *f*-free paths and intervals of optimality. For any point x, let $p(x)$ be an optimal path from s to point x and let $d(x)$ be the length of $p(x)$. The set of all points y on edge e such that there exists an optimal path to y with root r and last sequence $\mathscr{E}$ is some subset of e which can, in general, consist of up to $O(n)$ subsegments. This leads to a considerable complication when our algorithm tries to "propagate" signals from r across edge e. In the case of a convex polyhedron, this problem does not arise. In that case, the "wedges" in the planar unfolding of the surface are all convex since their boundaries are straight lines (which follows from the fact that optimal paths do not pass through vertices other than s, implying that bisectors between "wedges" are straight lines). In the nonconvex case, bisectors will be hyperbolic arcs, so the "wedges" will not be convex.

To overcome this difficulty, we define the notion of an *f-free path*: Given a face, f, and e, one of its three edges (we call (e, f) an *edge-face pair*), an *f-free path* to $y \in e$ is defined to be a path on $\mathscr{S}$ from s to y which does not intersect the interior of face f. A *shortest* f-free path to y, $p_f(y)$, is an f-free path to y which has minimal length. We will need to show the following.

LEMMA 4.1. *An optimal path $p(x)$ to a point $x \in e$ passes through the interior of at most one face containing x.*

Proof. If x is a vertex, there can be many faces containing x; if x is interior to e, there are exactly two. Let $f_1, f_2, \cdots, f_k$ be the faces containing x. If $p(x)$ passes through the interior of no face containing x, then we are done. Otherwise, let f_i be the *first* such face for which the optimal path, $p(x)$, has a point on its interior, and let y be a point on $p(x)$ which is interior to f_i. Then the subpath of $p(x)$ from y to x must be optimal. But the shortest path from y to x on $\mathscr{S}$ is just the straight segment from y to x, since this entire segment lies on face f_i and therefore on the surface $\mathscr{S}$. Thus, no other face $f_j \neq f_i$ containing x has the path $p(x)$ pass through its interior. □

Lemma 4.2 shows us how to obtain an optimal path to a point x on the interior of $e = f \cap f'$ from a shortest f-free and a shortest f'-free path to x.

LEMMA 4.2. *An optimal path to x on the interior of e is given by the shorter of $p_f(x)$ and $p_{f'}(x)$.*

Lemma 4.3 states another important property of shortest f-free paths: that they are not allowed to cross each other.

LEMMA 4.3. *If $p(x)$ and $p(y)$ are optimal paths from s to points x and y, then they can intersect only at vertices of $\mathcal{S}$, and if they do intersect at v, then that subpath of $p(x)$ from s to v has the same length as that subpath of $p(y)$ from s to v. A similar statement holds for shortest f-free paths $p_f(x)$ and $p_f(y)$.*

Proof. If $p(x)$ and $p(y)$ intersect at z, then certainly the subpaths to z must be equal in length (otherwise, one of the two paths could be shortened by replacing the portion of it from s to z). If z were interior to a face, then both paths could be locally improved by taking a shortcut around z. Similarly, if z were interior to an edge, a local improvement would be possible for both paths (look now at the unfolded paths and then improve them as if z were interior to a face). The same arguments hold for shortest f-free paths, since such local improvements will keep both paths f-free. □

Note that while $p_f(x)$ and $p_f(y)$ cannot cross, it is possible for shortest f-free paths to cross for *different* f's (e.g., $p_{f_1}(x)$ may cross $p_{f_2}(y)$; see Fig. 5). The problem is that if we try to improve one of the paths (say $p_{f_1}(x)$) by replacing part of it with a subpath of $p_{f_2}(y)$, then it may cease to be f_1-free.

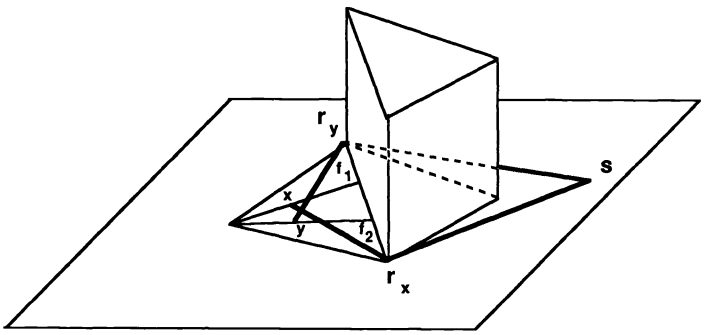


FIG. 5. $p_{f_1}(x)$ *may cross* $p_{f_2}(y)$.

The importance to our algorithm of the concept of shortest f-free paths is evident from the following lemma.

LEMMA 4.4. *For edge-face pair (e, f), edge sequence $\mathcal{E}$, and some vertex r, the set $\mathcal{I}$ of points x on e for which there exists a shortest f-free path to x with root r and last edge sequence $\mathcal{E}$ is connected (and, therefore, a subsegment of e).*

Proof. If there are no points of e which can be reached along a shortest f-free path with root r and last edge sequence $\mathcal{E}$, then we are done. Otherwise, let x and x' be points of e such that $p_f(x)$ and $p_f(x')$ have root r and last edge sequence $\mathcal{E}$. Then we wish to show that if y is a point of e between x and x', then $p_f(y)$ also has root r and last edge sequence $\mathcal{E}$. Now, unfold the edge sequence $\mathcal{E}$ and let $\bar{r} = U_{\mathcal{E}}(r)$ be the resulting unfolded image of r. Note that the segments $\overline{\bar{r}x}$, $\overline{\bar{r}x'}$, and $\overline{xx'}$ form a triangle in the unfolded image, since, by Lemma 3.3, the unfolded images of $\overline{rx}$ and $\overline{rx'}$ are straight line segments. Any f-free path, $p_f(y)$, to y cannot go through the interior of f; thus, $p_f(y)$ must pass through the interior of the triangle. Also, since shortest f-free paths do not cross (Lemma 4.3), the unfolded image of $p_f(y)$ cannot intersect the interiors of segments $\overline{\bar{r}x}$ and $\overline{\bar{r}x'}$. Thus, $p_f(y)$ must intersect the triangle at the point $\bar{r}$, meaning that $p_f(y)$ has root r and last edge sequence $\mathcal{E}$. □

The interval $\mathcal{I}$ of points can be open, closed, or half-open and half closed. (An open endpoint arises in the degenerate case in which the unfolding of the shortest f-free path to the point passes through two or more vertices (see Fig. 6). The fact that the resulting interval has an open endpoint is really an artifact of our definition of the root of a path as being the *first* vertex encountered when tracing back along the path.) Let the subsegment $I = [a, b]$ be the closure of $\mathcal{I}$. (Normally, we would write $I = \overline{ab}$ to indicate the segment with endpoints a and b; however, we write $[a, b]$ to emphasize the fact that coordinate values can be attached to points of e, and I can be thought of as an interval of these coordinates.) The endpoint a (resp., b) is the one on the *left* (resp., *right*) when viewed looking into face f. We call I the *interval of optimality* for r and $\mathcal{E}$ *with respect to* (e, f). An alternative specification of an interval of optimality is to give its *unfolded root* $\bar{r} = U_{\mathcal{E}}(r)$, its *depth* $d = d(r)$, and the corresponding edge-face pair (e, f). Note that r may be an endpoint of e (and hence of I) in the degenerate case. (Let us establish the convention that $\mathcal{E}$ does not include the edge e, so that $U_{\mathcal{E}}(r)$ writes the image of r in the coordinate system of face f'. If $\mathcal{E} = \emptyset$, then r will be one of the vertices of f'.) A point x is an element of the interval of optimality of $\bar{r}$ with respect to (e, f) if there exists a shortest f-free path to x whose unfolded image (along its last edge sequence) in the plane of f contains the segment $\overline{\bar{r}x}$.

We illustrate the notion of unfoldings and intervals of optimality in Fig. 6. On the left we show the interval I' of points on e which are "accessible" from r through the given edge sequence along paths that unfold into straight lines. Note that I' is determined by the left and right "clipping" vertices, r_L and r_R. (I' is simply that part of e which is visible from r within the polygon obtained by unfolding the sequence of adjacent faces.) In the figure on the right, we show the interval of optimality $I = [a, b]$. (Note also that in this case, $\mathcal{I} = (a, b]$ is half open, since the root of the

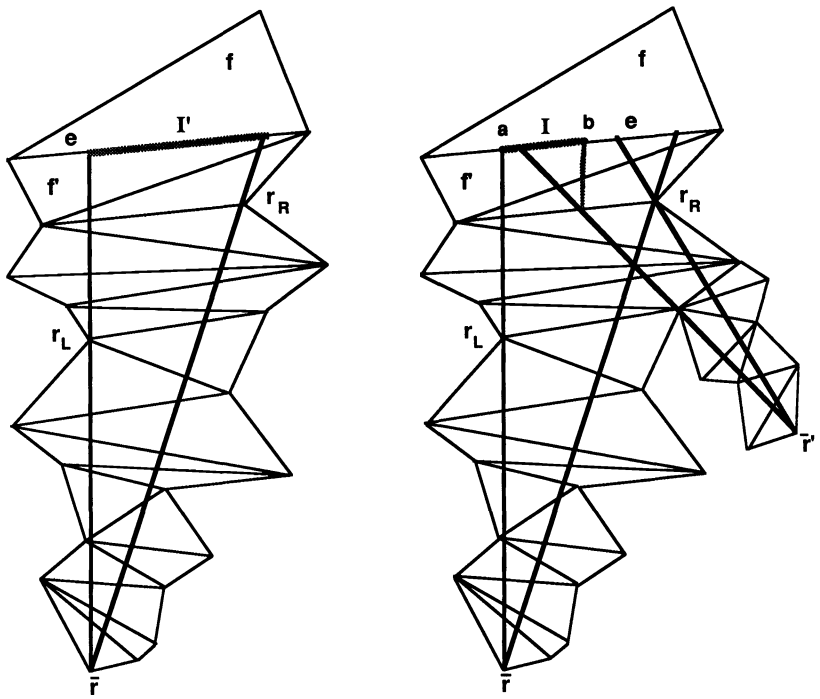


FIG. 6. *Unfoldings and intervals of optimality.*

path to a is r_L, not r.) The interval to the left of I is simply that which is rooted at r_L. The right endpoint, b, was determined by clipping I' along the hyperbolic bisector between $\bar{r}$ and $\bar{r}'$. Points right of b may be accessible from r, but they are better reached from r'.

It is easy to see that intervals of optimality with respect to (e,f) form a covering of e and that the intervals can touch but cannot overlap.

LEMMA 4.5. *Intervals of optimality with respect to an edge-face pair (e,f) form a covering of e and have mutually disjoint interiors.*

Proof. First note that every point $x \in e$ must lie in *some* interval of optimality with respect to (e,f) (if $p_f(x)$ is any shortest f-free path to x, then x lies in the interval of optimality of the unfolded image of the root of $p_f(x)$ with respect to (e,f)). Now, for a point $x \in e$ to be in the intervals of optimality of $\bar{r}$ and of $\bar{r}'$ with respect to (e,f) it must satisfy the equation $d(r)+|\bar{r}x| = d(r')+|\bar{r}'x|$. But this equation represents a hyperbola (which degenerates to a straight line if $d(r)=d(r')$). The intersection of this hyperbola with the edge e is a point (in the degenerate case, the straight line cannot contain e since the points $\bar{r}$ and $\bar{r}'$ would be on opposite sides of the line through e). Thus, there can be no open subinterval of e all of whose points lie in two or more intervals of optimality. This implies that the intervals must have disjoint interiors. □

The point, c, in I which is closest to r is called the *frontier point* of the interval. Since the distance from a point r to a closed line segment $[a, b]$ is minimized at exactly one point of $[a, b]$, we know that a unique such c exists. It will be either an endpoint of I or a point in the interior of I depending on whether the perpendicular projection of r onto the line through e lies outside or inside (respectively) the interval I.

The intersection of the f-free path to c with the face f' (which shares edge e with face f) is a line segment, $\overline{\beta c}$, on face f'. We call point β the *access point* of interval I. Note that when c is a vertex, it is possible for β to equal c, so that $\overline{\beta c}$ degenerates to a point. If we draw the line segments $\overline{\beta c}$ for every interval of optimality with respect to (e,f), then we get a partitioning of face f' into *access channels* (since no two such segments can intersect). Note that an access channel is either a triangle, a quadrilateral, or a pentagon. We will refer to an access channel by giving a pair, (I_1, I_2), where I_1 and I_2 are the intervals of optimality which give rise to the bordering segments, $\overline{\beta_1 c_1}$ and $\overline{\beta_2 c_2}$. If $I_1 = \text{NIL}$, the channel is simply that part of f' to the left of $\overline{\beta_2 c_2}$; and if $I_2 = \text{NIL}$, the channel is that to the right of $\overline{\beta_1 c_1}$.

5. The algorithm. Our algorithm operates in the precise spirit of Dijkstra's algorithm [1]. It is an important example of the power of the technique we are calling "continuous Dijkstra." A "signal" is propagated from the source to the rest of the surface. Once a point x of the surface receives the signal for the first time, it propagates it further; point x is considered to be *permanently labeled* with the time, $d(x)$, at which it received the signal, which is, of course, the minimum distance from the source to x. Fortunately, we have to do this labeling and repropagation for only finitely many (in fact, as we shall show, $O(n^2)$) points (called *event points*) of the surface. (Actually, we do a "directed" form of continuous Dijkstra, since we label points on an edge with the lengths of paths that are incident to the edge from either of the two sides of the edge. This comes from the fact that our intervals of optimality were defined with respect to f-free paths.)

The algorithm uses a few simple data structures. We keep a list, ILIST, of *candidate* intervals of optimality. A *candidate interval* (or *interval* for short) is a subsegment of an edge which is a supersegment of some (possibly empty) interval of optimality and

has exactly the same structure as that interval (i.e., it has the same type of information associated with it in its data structure). We call them *candidate* intervals because, at the conclusion of our algorithm, all remaining candidate intervals will be intervals of optimality. A candidate interval, I, has the following information associated with it: its extent, $[a, b]$; its edge-face pair, (e,f); its root, r; its unfolded root, $\bar{r}$; its depth, d; its frontier point, c; its access point, β; and its predecessor, $\bar{I}$ (which is the candidate interval (or vertex) whose "propagation" originally gave rise to I). We then will write $I = ([a, b], (e,f), r, \bar{r}, d, c, \beta, \bar{I})$.

An edge-face pair, (e,f), has associated with it a sorted list of candidate intervals with respect to (e,f), with the intervals in the order that they appear along the edge (following the edge in the direction which keeps the face f on the *left*). Such an ordering is possible since candidate intervals will always have disjoint interiors. Initially and during the execution of the algorithm, the *interval list* of (e,f) need not *cover* the edge e (that is, there may be points of e that lie in no candidate interval), but at the conclusion of the algorithm, we will see that the list does form a covering.

Each vertex v has associated with it a pointer to each candidate interval of which it is a root. Additionally, there is a *permanent label*, $d(v)$, which indicates the shortest path length to vertex v. Initially, $d(v) = +\infty$ for every vertex v. As the algorithm proceeds, vertices will become *permanently labeled*, meaning that they are assigned a label $d(v) < +\infty$. We will also be labeling some nonvertex points (frontier points which are interior to edges). These are also assumed to have infinite initial labels.

The algorithm also maintains a log n priority queue (called the *event queue*) whose entries are points of some candidate interval (either an endpoint or the frontier point) and have labels which are the best-known distances back to the source. (We do not actually need to keep track of all of the endpoints of candidate intervals as event points. It would suffice to keep track only of events associated with vertices and with frontier points that are interior to their intervals; however, the efficiency of our algorithm is not affected by this change.)

The algorithm proceeds as follows.

ALGORITHM.

(0) (**Initialize**). Permanently label s with 0. Initialize the event queue and ILIST to be empty. For each edge-free pair, initialize its interval list to be empty. For each edge opposite to s, create a candidate interval whose extent is the entire edge and whose root is s. Compute c for each such interval and place each such c and the endpoints of the edges in the event queue, each point having as label its distance from s. Push the resulting intervals onto the list ILIST and onto interval lists of the corresponding edge-face pairs.

(1) (**Main Loop**). While there is an entry in the event queue, remove the one with the smallest label and permanently label it. If it is being labeled as the frontier point of some candidate interval, I, then do *Propagate* (I).

Intuitively, to *propagate* an interval I means to allow the "wave front" of signals from the root of I to pass through I to the other two edges of the face and to record on those edges the intervals of points that are optimally reached by the signal paths through I. One interval can propagate into two (one on each of the opposite edges), so we will have to be careful how we propagate intervals so that the number of them does not explode exponentially. The critical observation which allows for a "trimming" of the number of intervals is that optimal paths and shortest f-free paths *do not cross* (Lemma 4.3).

Thus, the propagation of intervals can take place only through the access channels already established by intervals on the opposite edges. We give now the details of the procedure *Propagate* (I):

PROCEDURE *Propagate* (I).

(0) Assume that I is an interval with respect to (e, f), where $e = f \cap f'$. Let $c \in e$ be the frontier point of I. Let $e_1 = f \cap f_1$ and $e_2 = f \cap f_2$ be the edges of f opposite edge e. Refer to Fig. 7.

(1) (c **is not a vertex**). Let $I_i = Project\ (I, e_i)$ and, if $I_i \neq$ NIL, do *Insert-Interval* (I_i, c), for $i = 1, 2$.

(2) (c **is a vertex**). For each edge e_0 opposite c such that some point of e_0 makes an angle greater than π with segment $\overline{\beta c}$, create a candidate interval, I_0, on e_0 whose extent is the entire edge e_0 and whose root is c. Do *Insert-Interval* (I_0, c).

Step (2) makes use of Lemma 3.4 to specify when it is reasonable to introduce vertex c as a root to a candidate interval on an opposite edge. (This is not essential to the correctness of the algorithm or to its complexity, but only serves to make it run slightly faster.) We only introduce intervals on edges for which there is a chance that the angle that the resulting paths will make at c will be greater than π.

The function *Project* (I, e_1) simply finds the subset of e_1 that is hit by paths through I. We just apply the local optimality criterion (that paths unfold into straight lines; Lemma 3.3) to determine how the "wedge" of paths is extended through the next face. The function returns a candidate interval, I_A, on e_1 whose points are accessible through I. (If no part of e_1 is accessible, we return NIL.) The root of I_A is the same as that of I, and the unfolded root of I_A is obtained from that of I simply by rotating it (about e) into the coordinate system of face f'.

The procedure *Insert-Interval* (I, c) simply inserts a new candidate interval, I, between the intervals I_1 and I_2 that define the access channel containing c. The interval I may "dominate" I_1 or I_2 (which can happen only if the corresponding frontier point of I_1 or I_2 has not yet been permanently labeled), in which case the dominated candidate

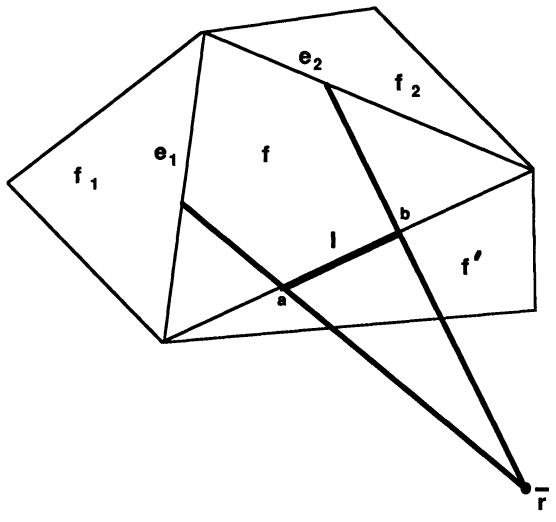


FIG. 7. *Propagation of an interval.*

must be deleted. Also, I will possibly be "trimmed" at the "tie points" between the root of I and the roots of I_1 and I_2. Thus, the portion of I which survives the trimming will correspond to that part of the edge containing I which will be "hit" first by signals from the root of I which pass through the last edge sequence of I.

PROCEDURE *Insert-Interval* (I, c).

(0) Assume $I=[a, b]$ is an interval with respect to (e, f) and that it has root r, unfolded root $\bar{r}$, and depth d. Point c is the frontier point of $\bar{I}$, the predecessor of I.

(1) (**Locate point c in access channel**). Locate point c in an access channel, say (I_1, I_2), of (e, f). The wavefront can only get to I through this channel. Let $I_i=[a_i, b_i]$ (provided that $I_i \neq$ NIL), for $i=1, 2$. Let r_i be the root, $\bar{r}_i$ be the unfolded root, and d_i be the depth of I_i.

(2) (**Delete dominated candidate intervals**). If $I_1 \neq$ NIL, $a_1 \in I$, and $d_1+|\bar{r}_1 a_1| \geqq d+|\bar{r} a_1|$ then do *Delete* (I_1), set I_1 to be the predecessor of I_1 in the interval list of e_1 (set I_1 to NIL if no predecessor exists), and go back to (2). Similarly, if $I_2 \neq$ NIL, $b_2 \in I$, and $d_2+|\bar{r}_2 a_2| \geqq d+|\bar{r} a_2|$, then do *Delete* (I_2), set I_2 to be the successor of I_2 in the interval list of e_1 (set I_2 to NIL if no successor exists), and go back to (2).

(3) (**Trim at tie points**). If $I_1 \neq$ NIL and $b_1 \in I$, then set a to be the point in $I \cap I_1$ such that $d_1+|\bar{r}_1 a|=d+|\bar{r} a|$ (if no such point exists, let a be b_1). Then a will be the point in $I \cap I_1$ which can be reached equally well through I and through I_1. Remove b_1 from the event queue, set b_1 to a, and update c_1 if necessary. If $I_2 \neq$ NIL and $a_2 \in I$, then set b to be the point in $I \cap I_2$ such that $d_2+|\bar{r}_2 b|=d+|\bar{r} b|$ (if no such point exists, let b be a_2). Then b will be the point in $I \cap I_2$ which can be reached equally well through I and through I_2. Remove a_2 from the event queue, set a_2 to b, and update c_2 if necessary. If $a \neq b$, go to (4); otherwise, stop (no interval needs to be inserted).

(4) (**Final updates**). Compute the frontier point, c_I, and access point for $I=[a, b]$. Push I onto the list ILIST and insert it into the interval list of edge e (between intervals I_1 and I_2). Put the points a, b, and c_I (if $c_I \neq a, b$) into the event queue, with labels $d+|\bar{r} a|$, $d+|\bar{r} b|$, and $d+|\bar{r} c_I|$, respectively.

The procedure *Delete* (I) simply deletes I from the ILIST, deletes I from the interval list of the corresponding edge-face pair, and deletes any entries of points of I in the event queue.

In step (1) of the above procedure, we must locate a point c in an access channel of an edge-face pair (e, f). This is easily done in time $O(\log n)$ by binary search as follows: We have for (e, f) an interval list which gives the segments $\overline{\beta_i c_i}$ in sorted order along edge e. The segments are nonintersecting, so to locate a point c on face f' in an access channel, we can do binary search in which each test determines on what side of a segment $\overline{\beta_i c_i}$ the point c lies (see Fig. 8).

6. Proof of correctness. We now begin a proof that the algorithm works. Our first objective is to prove the following proposition.

PROPOSITION 6.1. *After each iteration of the Main Loop of the Algorithm, the current list of candidate intervals (ILIST) correctly gives the best f-free path so far from s to points of these intervals, where "best so far" has the following meaning: If $x \in e=f \cap f'$ lies in interval I of root r with respect to (e, f), then an f-free path to x which has root r and length $d(r)+|\bar{r} x|$ is optimal among those f-free paths to x that enter face f' through an interval in the current ILIST.*

Proof. The proof proceeds by induction on the iteration count in the Main Loop. At the first iteration, the claim is true because the only candidate intervals are the trivial ones on the edges opposite to the source s. Assume now that the claim holds for the first k iterations (this is the Induction Hypothesis). The inductive step requires

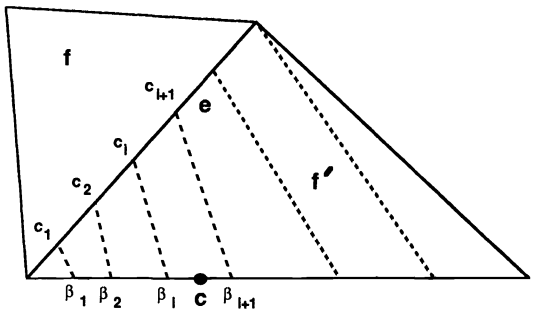


FIG. 8. *Locating point c in a channel.*

us to prove that if, at iteration $k+1$, we introduce interval I for root r with respect to (e,f), then the best f-free path so far to points of I is through root r. This will be shown in the following two lemmas. □

Assume now that the Induction Hypothesis holds.

A candidate interval will have one of two fates: Either its frontier point gets permanently labeled (by its becoming the top element in the priority queue), thereby assuring that some part of it will "survive" and become part of a final interval of optimality with the same attributes as the candidate; or, the candidate interval gets deleted in Step (2) of *Insert-Interval*, in which case we can guarantee that the interval of optimality corresponding to the candidate will be empty.

LEMMA 6.2. *If we delete interval I_1 in step* (2) *of procedure Insert-Interval, then the interval of optimality of root r_1 with respect to (e,f) has an empty interior. A similar statement holds if we delete interval I_2.*

Proof. Assume that $a_1 \in I$ and $d_1 + |\bar{r}_1 a_1| \geq d + |\bar{r} a_1|$. Then, the best f-free path so far to point a_1 is through root r. Let β be the intersection of $\bar{r}a_1$ with edge e'. Since $a_1 \in I$, we know that $\beta \in \bar{I}$, by the method used to construct I from its predecessor, $\bar{I}$. Let us assume that there exists a point y in the interior of the candidate interval of root r_1 with respect to (e,f) (so the path $p_f(y)$ has root r_1). We will then arrive at a contradiction.

First, note that y is interior to $[a_1, b_1]$ (otherwise, y is either inaccessible from r_1 or can be reached from some other root of (e,f) along a shorter f-free path). Next, note that interval I_1 was established when its predecessor, $\bar{I}_1$, was propagated. Since point $c \in \bar{I}$ was located in channel (I_1, I_2), we know that c lies to the *right* of the segment $\overline{\bar{r}_1 c_1}$, and therefore to the right of $\overline{\bar{r}_1 b_1}$ (otherwise, $c \in \bar{I}_1$). Since $\bar{I}$ is connected, then, *all* points of $\bar{I}$ lie to the right of $\overline{\bar{r}_1 z}$ for *any* $z \in [a_1, b_1]$, and lie *strictly* to the right of $\overline{\bar{r}_1 z}$ for z *interior* to $[a_1, b_1]$. In particular, β lies strictly to the right of the segment $\overline{\beta_1 y}$, where $\overline{\beta_1 y}$ is the intersection of $\overline{\bar{r}_1 y}$ with face f'. But, a_1 lies strictly to the *left* of segment $\overline{\beta_1 y}$, so segments $\overline{\beta a_1}$ and $\overline{\beta_1 y}$ must intersect in some point γ which is interior to both segments (see Fig. 9). Since γ lies on the path $p_f(y)$, the f-free path through r_1 to γ is no longer than the f-free path through r to γ. But this means that we can get a strict improvement to the path to a_1 by going through root r_1 directly to a_1 (since $|\bar{r}_1 \gamma| + |\gamma a_1| > |\bar{r}_1 a_1|$). This contradicts our assumption that $d_1 + |\bar{r}_1 a_1| \geq d + |\bar{r} a_1|$.

A similar proof holds if we delete interval I_2. □

Next, we must prove that the interval that is inserted by procedure *Insert-Interval* is "correct" in the following sense.

LEMMA 6.3. *When procedure Insert-Interval inserts an interval I between intervals I_1 and I_2, the resulting interval list of edge e correctly subdivides points of e according to the best f-free paths so far.*

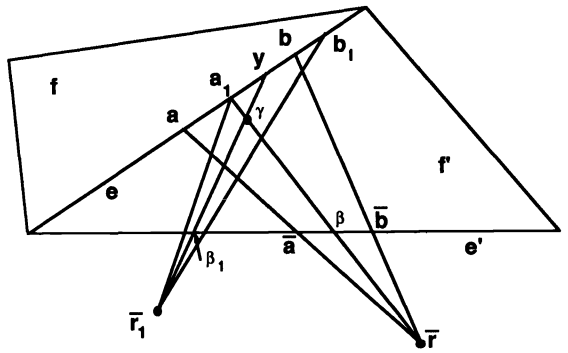


FIG. 9. *Illustration of proof of Lemma* 6.2.

Proof. Let $[a_1, b_1]$ and $[a_2, b_2]$ be the original extents of I_1 and I_2, and let $[a_1, b_1']$ and $[a_2', b_2]$ be the resulting intervals after the trimming is done by procedure *Insert-Interval.* Now, the best path so far to a point outside $[a_1, b_2]$ cannot go through r since it would have to cross some access channel boundary, $\overline{\beta_i c_i}$.

Now let $x \in I = [a, b]$, where $[a, b]$ is the extent of I *after* any trimming that may have been done. Since, by construction, $I \subseteq [a_1, b_2]$, the best path so far to x must be through r, r_1, or r_2. If x is not in $[a_1, b_1]$, then it is better to reach x through root r than through root r_1. If x is interior to $[a_1, b_1]$, then so is a (since $[a, b] \subseteq [a_1, b_2]$). But then the path to a through root r is no longer than the path to a through root r_1 and is, therefore, optimal so far. If it were better to reach x through root r_1 than through root r, then the path to x would intersect the path through r to a (since segments $\overline{ra}$ and $\overline{r_1 x}$ must cross), a contradiction. Likewise, we can show that it is better to reach x through r than through r_2.

If $x \in [a_1, b_1']$ (resp., $[a_2', b_2]$), a similar argument shows that it is better to reach x through r_1 (resp., r_2) than through r (and it is certainly better to reach x through r_1 (resp., r_2) than through r_2 (resp., r_1) since $[a_1, b_1'] \subseteq [a_1, b_1]$ and $[a_2', b_2] \subseteq [a_2, b_2]$). □

This completes the proof of Proposition 6.1.

One can show that during an execution of Dijkstra's algorithm, when a node is "permanently labeled" (i.e., put on the "CLOSED" list), it is labeled with the length of the shortest path from the source to the node. Analogously, we have the following lemma.

LEMMA 6.4. *When a point interior to an edge is permanently labeled as part of a candidate interval with respect to* (e, f), *it is labeled with the shortest f-free path length to it. When a vertex is permanently labeled, it is labeled with the shortest path length to it.*

Proof. Let x be interior to edge $e = f \cap f'$. Assume that the algorithm has just given x a permanent label of δ and that the path $p_f(x)$ is a shortest f-free path to x whose length is less than δ. Now, $p_f(x)$ intersects at least one candidate interval (since we initialized the ILIST to "surround" point s), so we can let $I_l = [a_l, b_l]$ be the *last* candidate interval intersected by $p_f(x)$, and assume that I_l is an interval with respect to (e_l, f_l). (Note that there will be more than one such interval if $p_f(x)$ intersects I_l at one of its endpoints; in that case, pick either interval. In the case that $p_f(x) \cap I_l = v$, a vertex, pick I_l such that $p_f(x) \cap f_l \neq v$.) Since the length of $p_f(x)$ is less than δ, we know that $p_f(x)$ enters face f' through some point which is in no current candidate interval (by Proposition 6.1, since x was labeled with the best path length so far). Thus, $f_l \neq f'$. Let $\overline{\beta x'} = f_l \cap p_f(x)$. Then, point x' lies on an edge $e_0 = f_l \cap f_0$. Let $p_{f_0}(x')$ be the subpath of $p_f(x)$ from s to x'. Clearly, $p_{f_0}(x')$ is a shortest f_0-free path to x'.

The interval I_l *must* have been propagated (since the distance to the frontier point of I_l is $\leqq d(\beta) < d(x) < \delta$, and the algorithm always permanently labels the closest event point). But then, by Lemma 6.3, there would have been a candidate interval created on edge e_0 which includes point x'. This contradicts the fact that I is the *last* candidate interval intersected by $p_f(x)$.

A similar argument shows that vertices are permanently labeled with the length of an optimal path to them. □

In our mental picture of the execution of the algorithm, we think of the "wavefront" moving out from s and imagine that, as the front encounters new edges or boundaries of candidate intervals, events occur. When an event occurs at a certain distance from s, then all points closer to s than that distance have already encountered the "wavefront," as Lemma 6.5 states.

LEMMA 6.5. *If the algorithm has just assigned a permanent label of δ to some point, then for any point x on an edge e such that $d(x) < \delta$, there is an interval I (for r with respect to (e,f)) in the current ILIST which contains x such that an optimal path to x has root r and length $d(r) + |\bar{r}x|$.*

Proof. Let $x \in e$ be an edge point such that $d(x) < \delta$. Assume that x does *not* lie in a candidate interval which correctly identifies an optimal path to x. Let $p(x)$ be an optimal path to x. As in the proof of Lemma 6.4, let $I_l = [a_l, b_l]$ be the *last* candidate interval of the current ILIST intersected by $p(x)$, and assume that I_l is an interval with respect to (e_l, f_l). Our assumption on x says that $x \notin I_l$. Let $\overline{\beta x'} = f_l \cap p(x)$. Then, point x' lies on an edge $e_0 = f_l \cap f_0$. Let $p_{f_0}(x')$ be the subpath of $p(x)$ from s to x'. Clearly, $p_{f_0}(x')$ is a shortest f_0-free path to x'. The interval I_l *must* have been propagated (since the distance to the frontier point of I_l is $\leqq d(\beta) < d(x) < \delta$, and the algorithm always permanently labels the closest event point). But then, by Lemma 6.3, there would have been a candidate interval created on edge e_0 which includes point x'. This contradicts the fact that I_l is the *last* candidate interval intersected by $p(x)$. □

At the conclusion of our algorithm, the wavefront has encountered *all* points of $\mathcal{S}$:

LEMMA 6.6. *At the conclusion of the algorithm, the interval list of (e,f) forms a covering of the edge e.*

Proof. Assume that there is some point $x \in e = f \cap f'$ which is *not* covered by a candidate interval with respect to (e,f). Let $p_f(x)$ be a shortest f-free path to x. As in the proof of Lemma 6.4, let $I_l = [a_l, b_l]$ be the *last* candidate interval intersected by $p_f(x)$, and assume that I_l is an interval with respect to (e_l, f_l). Since x is not covered, $e_l \neq e$. Let $\overline{\beta x'} = f_l \cap p_f(x)$. Then, point x' lies on an edge $e_0 = f_l \cap f_0$. Let $p_{f_0}(x')$ be the subpath of $p_f(x)$ from s to x'. Clearly, $p_{f_0}(x')$ is a shortest f_0-free path to x'. The interval I_l *must* have been propagated (otherwise, its frontier point would still be in the event queue, and the algorithm would not have concluded). But then, by Lemma 6.3, there would have been a candidate interval created on edge e_0 which includes point x'. This contradicts the fact that I_l is the *last* candidate interval intersected by $p_f(x)$. □

LEMMA 6.7. *The following are equivalent for a point x on edge e of face f.*

(1) *At the conclusion of the algorithm, x lies in the candidate interval of root r (and unfolded root $\bar{r}$) with respect to (e,f).*

(2) *There is a shortest f-free path p from s to x with root r and length $d(r) + |\bar{r}x|$; that is, x lies in the interval of optimality of $\bar{r}$ with respect to (e,f).*

Proof. The proof follows immediately from Proposition 6.1 and Lemma 6.6. □

7. Complexity analysis. To establish the running time of our algorithm, we first must bound the number of event points.

LEMMA 7.1. *There are at most $O(n^2)$ candidate intervals created by the algorithm.*

Proof. Let $I_1, I_2, \cdots, I_K$ be the interval list for some edge-face pair (e, f). Let x_i be a point interior to interval I_i, and let edge $e = \overline{x_0 x_{K+1}}$. Now draw the shortest f-free paths from s to each x_i $(0 \leqq i \leqq K+1)$. (Note that none of these paths cross edge e.) It is clear that this divides the polyhedral surface $\mathcal{S}$ into $K+1$ subsurfaces (subsurface i is the region to the right of the cycle from s to x_i (along $p_f(x_i)$), from x_i to x_{i+1} (along e), and then from x_{i+1} back to s (along the reverse of $p_f(x_{i+1})$). Note that paths $p_f(x_i)$ and $p_f(x_{i+1})$ may have some vertices in common. The interior of each such subsurface must contain a vertex. Thus, $K \leqq n-1$. Since there are at most $O(n)$ edge-face pairs, there can be at most $O(n^2)$ candidate intervals.

The above argument applies only to the case of a polyhedron of genus zero, as we are implicitly using the Jordan curve theorem to say that the surface is partitioned into $K+1$ subsurfaces. The case in which the genus may be higher can be handled by the following alternative proof. We trace back the paths from x_i and from x_{i+1}, noting where the paths first diverge by going through different edges (or where one path encounters a vertex, its root). Define the *fork point*, v_i, to be the vertex responsible for the divergence. It may be a root of one of the paths, or it may be a vertex between two adjacent edges whose interiors are crossed by the two paths. Let f_i be the face on which the divergence occurred; that is, both $p_f(x_i)$ and $p_f(x_{i+1})$ pass through the interior of f_i, and v_i is the vertex of f_i which "splits" paths $p_f(x_i)$ and $p_f(x_{i+1})$ (and it may lie on one of the paths). Then, we "charge" the interval I_i to the vertex-face pair, (v_i, f_i). It is not hard to see that each vertex-face pair can be charged at most twice: If v_i is a root of one path (say $p_f(x_i)$), then (v_i, f_i) can be charged at most twice (once for the pair $p_f(x_{i-1})$, $p_f(x_i)$, and once for the pair $p_f(x_i)$, $p_f(x_{i+1})$); otherwise, (v_i, f_i) is charged at most once. (The proofs of these claims simply use the fact that shortest f-free paths do not cross.) There is a one-to-one mapping between vertex-face pairs and edges, so we get that each edge can be charged at most twice. This gives us a linear bound for K in terms of the number of *edges* of $\mathcal{S}$ (which seems to be the proper measure of complexity for higher genus polyhedra). □

We can also bound the number of calls to *Delete* in procedure *Insert Interval.*

LEMMA 7.2. *There will be at most $O(n^2)$ calls to procedure Delete.*

Proof. By Lemma 6.2, we never delete an interval which has had some point permanently labeled. Once an interval is deleted, it can never be propagated. When an interval is propagated, its frontier point is permanently labeled and there are at most two new intervals created on the opposite edges. These two new intervals are candidate intervals which may end up being deleted. But there can be at most $O(n^2)$ such candidates, since, by Lemma 7.1, there can be at most $O(n^2)$ intervals with permanently labeled points. □

The size of the data structures needed is easily seen to be quadratic.

LEMMA 7.3. *The data structures require at most $O(n^2)$ space.*

Proof. Each of the $O(n^2)$ candidate intervals requires only a constant amount of storage. (Note that we do *not* keep the edge sequence corresponding to each interval, rather only the image of the root.) Since each edge is adjacent to only two faces, there are only $O(n)$ edge-face pairs, and each of these has associated with it a list of at most $O(n)$ candidate intervals (by the proof of Lemma 7.1). Thus, the data structure for the edge-face pairs requires only $O(n^2)$ space. The event queue can have at most three times the number of candidate intervals, so this structure is also bounded in size by $O(n^2)$. □

We are finally ready to assert the validity of our algorithm and its claimed efficiency.

THEOREM 7.4. *The algorithm above correctly computes the subdivision of each edge into intervals of optimality in time $O(n^2 \log n)$ and space $O(n^2)$.*

Proof. Correctness is based on Lemma 6.7. The space bound is shown in Lemma 7.3. For the time bound, we know from Lemma 7.1 that there are at most $O(n^2)$ events placed in the event queue. Since each event will be inserted and deleted only once, the total time needed for operations on the queue is $O(n^2 \log n)$. Also, by Lemma 7.1, there will be $O(n^2)$ calls to the procedure *Propagate*. If the frontier point being labeled is *not* a vertex, *Propagate* takes time $O(\log n)$ (just the time to locate c in an access channel) and then calls *Insert-Interval*. Each cell to *Insert-Interval* requires only constant time *except* for the work needed to call *Delete*. But each call to *Delete* requires at most $O(\log n)$ time (to delete an element from an interval list), so, by Lemma 7.2, the total time required is $O(n^2 \log n)$. In the case that the frontier point *is* a vertex, a single call to *Propagate* could potentially take time $O(n \log n)$ (since there could be $O(n)$ faces adjacent to c and each call to *Insert-Interval* takes $O(\log n)$ time). But the *total* number of calls to the procedure *Insert-Interval* cannot exceed the number of pairs (v, e), where v is a vertex, and e is the edge opposite v. Clearly, there can be at most $O(n)$ such pairs. Thus, the total time required in making calls to *Insert-Interval* is at most $O(n^2 \log n)$. This gives a total time bound of $O(n^2 \log n)$. □

8. Building the subdivision. The algorithm we presented allows us to recover the length, $d(x)$, of the shortest path from s to any point x on any edge $e = f \cap f'$. If x is a vertex, then $d(x)$ is just the label that was given it when x was permanently labeled. Otherwise, we locate x (in time $O(\log n)$) in interval of optimality I (resp., I') for root r (resp., r') with respect to (e, f) (resp., (e, f')). By Lemma 4.2, the smaller of the two sums, $d(r)+|x\bar{r}|$ and $d(r')+|x\bar{r}'|$ is the desired shortest path length.

We can extend our algorithm to find the shortest path length from s to any point of the polyhedral surface, as follows: Let e_1, e_2, and e_3 be the three edges of f. We build three subdivisions ($\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$) of face f. Subdivision $\mathcal{T}_i$ is the planar subdivision of f into cells associated with the intervals of optimality with respect to (e_i, f).

Let $I_1, I_2, \cdots, I_K$ be the interval list for edge-face pair (e_1, f). Associated with each interval of optimality I_j $(1 \leqq j \leqq K)$ is a cell C_j of $\mathcal{T}_1$ such that a point x lies in cell C_j if and only if there is a path (on $\mathcal{S}$) from s to x through interval I_j (that is, through the root of I_j and its edge sequence) which is shortest among all paths (on $\mathcal{S}$) from s to x which pass through edge e_1.

Let interval I_j have root r_j. Then a point x will be on the boundary of cells C_j and $C_{j'}$ if and only if $d(r_j)+|x\bar{r}_j| = d(r_{j'})+|x\bar{r}_{j'}|$, where $\bar{r}_j = U_{(\mathcal{E}_f, e)}(r_j)$ is the image of r_j unfolded into the plane of f (this is slightly different from the convention we have had so far that $\bar{r} = U_{\mathcal{E}}(r)$ is the unfolded image of r into the plane on the other side of e_1). Put another way, the *weighted* distance from x to $\bar{r}_j$ must equal that from x to $\bar{r}_{j'}$, where by weighted distance between a point x and another point y (which has an associated *weight* d) we mean the sum $d+|xy|$. From elementary analytic geometry, we know that the locus of points whose weighted distances to two points r and r' are equal is a hyperbola. Thus, the boundaries of cells C_j in the subdivision are hyperbolic arcs.

We now describe a procedure, *Build-Subdivision* (e, f), which builds the subdivision of f into cells C_j which give the best paths through e to points x on f. The procedure works by "propagating" the intervals of optimality $(I_1, I_2, \cdots, I_K)$ with respect to (e, f) into the interior of face f, keeping track of certain "event" points (which will be crossings of hyperbolic arcs of cell boundaries).

We let $h_{i,j}$ be the hyperbola containing the boundary between cell C_i (associated with interval $I_i = [a_i, b_i]$) and cell C_j (associated with interval $I_j = [a_j, b_j]$). Then $h_{i,j}$

is defined by the equation $d(r_i)+|x\bar{r}_i| = d(r_j)+|x\bar{r}_j|$ in x (in the coordinate system of face f). Note that $h_{i,j}$ is nonempty provided that $|d(r_i)-d(r_j)| \leqq |\bar{r}_i\bar{r}_j|$, where the case of equality yields a hyperbola which degenerates to a line. We truncate $h_{i,j}$ to be in the half-plane defined by the line through e and containing face f. We then think of $h_{i,j}$ being *oriented* in the direction of increasing distance from r_i and r_j. We define $h_{0,1}$ as the ray from a_1 along edge e_3 and $h_{K,K+1}$ as the ray from b_K along edge e_2. If cells C_i and C_j share a boundary, then we let $q_{i,j}$ $(0 \leqq i < j \leqq K+1)$ be the point of the boundary of C_i and C_j which is closest to point r_i (and r_j). See Fig. 10 for an example in which $K=6$. Finally, let u_i be the supremum of the distances from s to points of C_i (note that u_i may be $+\infty$).

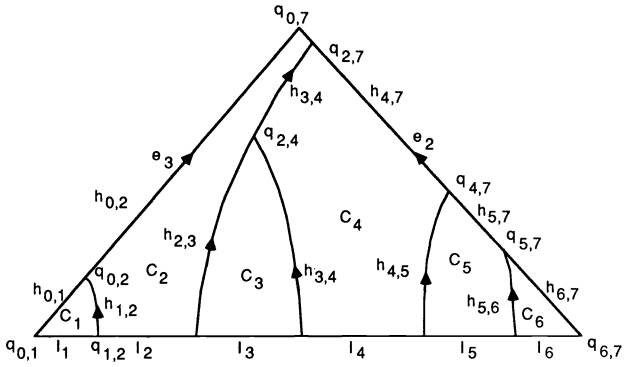


FIG. 10. *Extending the subdivision to the interior of f.*

PROCEDURE *Build-Subdivision* (e, f).

(0) Initialize INTLIST to be the list $(I_0, I_1, I_2, \cdots, I_K, I_{K+1})$. For each i $(1 \leqq i \leqq K)$, let $u_i = d_i + |\bar{r}_i w_i|$, where w_i is the point of intersection of $h_{i-1,i}$ and $h_{i,i+1}$ if they intersect, and let $u_i = +\infty$ if they do not intersect. Maintain the u_i's in a priority queue, ULIST.

(1) (**Determine Next Event**). If ULIST is empty (meaning that INTLIST equals (I_0, I_{K+1})), then stop. Otherwise, let the elements of INTLIST be indexed $(I_0, I_{i_1}, \cdots, I_{i_j}, I_{K+1})$. Let $u = u_{i_j}$ be the minimum value in ULIST (in case of ties, pick the one with smallest index i_j). Delete I_{i_j} from INTLIST, and delete u_{i_j} from ULIST.

(2) (**Update $u_{i_{j-1}}$ and $u_{i_{j+1}}$**). If $j \neq 1$, modify $u_{i_{j-1}}$ to be $d_{i_{j-1}} + |\bar{r}_{i_{j-1}} w_{j-1}|$, where w_{j-1} is the point of intersection of $h_{i_{j-2},i_{j-1}}$ and $h_{i_{j-1},i_{j+1}}$ if they intersect, and set $u_{i_{j-1}} = +\infty$ if they do not intersect. If $j \neq J$, modify $u_{i_{j+1}}$ to be $d_{i_{j+1}} + |\bar{r}_{i_{j+1}} w_{j+1}|$, where w_{j+1} is the point of intersection of $h_{i_{j-1},i_{j+1}}$ and $h_{i_{j+1},i_{j+2}}$ if they intersect, and set $u_{i_{j+1}} = +\infty$ if they do not intersect. Bisectors h_{i_{j-1},i_j} and $h_{i_j,i_{j+1}}$ become "inactive," and bisector $h_{i_{j-1},i_{j+1}}$ becomes "active" at this point.

(3) (**Update Subdivision**). Let $q_{i_{j-1},i_{j+1}}$ be the event point which corresponds to u_{i_j}. (Point $q_{i_{j-1},i_{j+1}}$ is the point of C_{i_j} which is furthest from r_{i_j}.) Establish $q_{i_{j-1},i_{j+1}}$ as a vertex of the subdivision, with incident arcs h_{i_{j-1},i_j} and $h_{i_j,i_{j+1}}$ (into $q_{i_{j-1},i_{j+1}}$) and $h_{i_{j-1},i_{j+1}}$ (out of $q_{i_{j-1},i_{j+1}}$). Go to step (1).

The procedure works by processing the vertices of $\mathcal{T}_1$ in the order of increasing distance from the source. Each event corresponds to "closing off" a cell C_{i_j}, at which time we delete I_{i_j} from the list INTLIST of intervals responsible for the wavefront, and we consider the effect of the bounding hyperbola between the cells corresponding

to the intervals on either side of I_{i_j} in list INTLIST. A typical step of the procedure is illustrated in Fig. 11. The lemma below shows that the procedure *Build-Subdivision* (e,f) does what it claims to do.

LEMMA 8.1. *The procedure Build-Subdivision* (e,f) *correctly constructs the cells* C_j *corresponding to the intervals of optimality of* (e,f).

Proof. The proof is by induction on the iteration count of the main loop. Assume that δ_k is the distance of the event point from s at iteration k. We first note that the subdivision induced by the initial set of hyperbolic bisectors is correct up to a distance δ_1 from s (since, up to distance δ_1, there are no intersections among the bisectors, by the definition of the event distance). Now assume that it is correct up to distance δ_k, and consider the effect of the event at distance δ_k. Event k corresponds to the closing off of a cell C_I and to the cells C_L and C_R becoming adjacent after distance δ_k. (C_L and C_R had been to the left and to the right, respectively, of C_I before the event.) If x is a point at distance δ from s, where $\delta_k \leqq \delta < \delta_{k+1}$, then we claim that x is properly categorized according to the current list of active hyperbolic bisectors, that is, according to the subdivision created by the procedure for points between event k and event $k+1$. First, x cannot be reached best through interval I, since it lies on the wrong side of the bisector between C_I and C_L or between C_I and C_R. The newly active bisector between C_L and C_R correctly categorizes x between the corresponding intervals. And the categorization of x with respect to all other intervals must be correct since all other bisectors are disjoint for distances from s which are between δ_k and δ_{k+1}, and by the induction all other bisectors are disjoint for distances from s which are between δ_k and δ_{k+1}, and by the induction hypothesis, we have assumed that the subdivision is correct so far. □

LEMMA 8.2. *The procedure Build-Subdivision* (e,f) *requires time* $O(K \log K)$ *and space* $O(K)$, *where* K *is the number of intervals of optimality with respect to* (e,f).

Proof. Since at each iteration of the procedure we delete one of the intervals from INTLIST, there can be at most K executions of steps (1) and (2). But each such

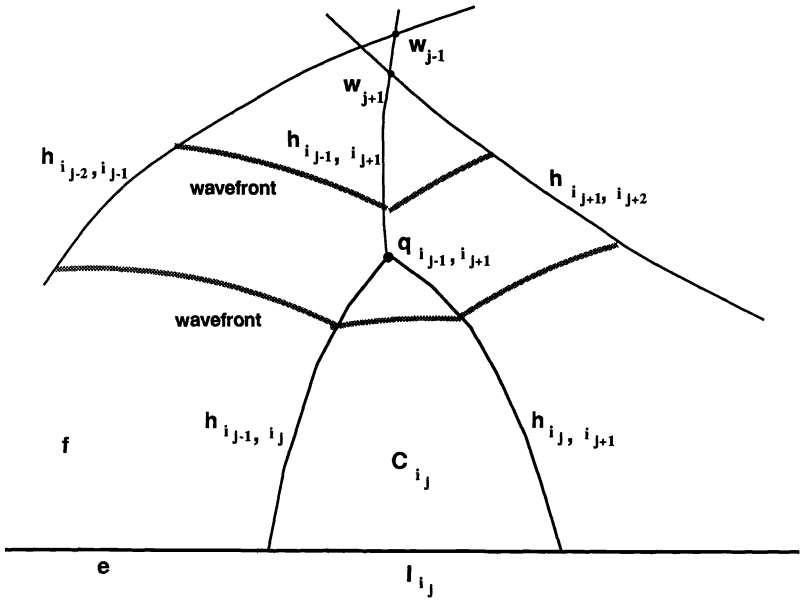


FIG. 11. *The event at point* $q_{i_{j-1},i_{j+1}}$.

execution involves a constant number of insertions and deletions from the list ULIST. Thus, the total time complexity of the procedure is clearly $O(K \log K)$. □

We should remark that it is not necessary to build independently each of the three subdivisions $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$. (This was done for simplicity of exposition.) Rather, we could have propagated the intervals for (e_1, f), (e_2, f), and (e_3, f) all at once into the interior of f. The procedure would be similar to that given above for a single edge, but now the INTLIST would be a cyclic list of intervals around the perimeter of f (eliminating the need for the "dummy" hyperbolas on the edges of f).

We execute the procedure *Build-Subdivision* (e, f) for every edge-face pair (e, f). This yields three subdivisions for each face and takes a total time of $O(n^2 \log n)$ (since there are a total of $O(n^2)$ intervals of optimality for all edge-face pairs). To find the length of the shortest path from s to any $x \in S$ we simply locate x in some face f, then we do three point locations of x within the three subdivisions of f. We now have the lengths of shortest paths to x through each of the three edges of f. Selecting the shortest of the three gives us the length of the shortest path to x. All four of the point locations can be done in time $O(\log n)$ by standard point location techniques ([15], [3]). (The only slight variation to the problem with which we must deal is the fact that the boundaries of cells are hyperbolic instead of straight. This, however, presents no real complication.) To determine an actual optimal path to x is straightforward by back-tracing through the predecessor pointers (from I to $\bar{I}$ to $\bar{\bar{I}}$, etc.). The back-tracing will, of course, take time $O(k)$, where k is the number of faces crossed by the optimal path. We therefore have the following theorem.

THEOREM 8.3. *After preprocessing, the length of the shortest path from s to any point $x \in \mathcal{S}$ on polyhedral surface $\mathcal{S}$ can be found in time $O(\log n)$ and a shortest path can be determined in time $O(k + \log n)$, where k is the number of faces crossed by the path. The preprocessing can be done in time $O(n^2 \log n)$ (and space $O(n^2)$).*

9. Conclusion. We have given an $O(n^2 \log n)$ algorithm for subdividing the surface of an arbitrary polyhedron so that the length of the shortest path from a given source s to any point t on the surface may be obtained simply by locating t in the subdivision. The algorithm requires $O(n^2)$ space to run. Point location can be done by standard methods in time $O(\log n)$, after which the actual path may be back-traced in time $O(k)$, where k is the number of faces traversed by the shortest path.

Our algorithm utilizes the continuous Dijkstra technique. The method is very analogous to Dijkstra's original shortest path algorithm, but includes the concepts of an expanding *wavefront* and discrete *events*. It is, in a sense, a cross between Dijkstra's algorithm and the plane sweep paradigm of computational geometry.

It is easy to extend our algorithm to the case of multiple sources and thereby to solve the Voronoi problem on the surface of a polyhedron [11]. We simply initialize the original event queue to be the events associated with the edges opposite to each source point, and then proceed exactly as before to propagate intervals. For m source points lying on the surface of a polyhedron, the algorithm builds the Voronoi diagram in time $O(N^2 \log N)$ and space $O(N^2)$, where $N = \max\{n, m\}$. One can then locate the closest source to a query point in time $O(\log N)$, and one can backtrace the path to the closest source in time proportional to the number of bends along the path.

It is also not hard to imagine how our algorithm might be generalized to other types of surfaces (instead of just those that are polyhedral). If we have a surface which is specified by giving the surface patches which make up its faces, then we can calculate for a fixed complexity of surface type (provided it is "well-behaved") how geodesic paths behave on that surface. For example, on planar patches we get the problem we

have covered here, and the geodesic paths are straight lines on each face. For a patch of spherical surface, geodesic paths will be arcs of great circles for the underlying sphere. We would then have to generalize the local optimality criterion that geodesic paths that cross patch boundaries unfold into straight lines: Geodesic paths will unfold such that *locally* (at the boundary of surface patches) they unfold to be straight. This is simply a continuity condition on the slope of geodesic paths. The continuous Dijkstra algorithm allows the propagation of a wavefront across the surface patch boundaries, while maintaining the local optimality criterion. Some work is needed to figure out the details of this generalization.

We would like to be able to improve the running time of our algorithm. It may be possible to reduce the time complexity to $O(n^2)$ (the current best time for planar obstacle avoidance algorithms), but we should not hope to improve it further until the planar case is solved more efficiently.

It should be possible, however, to reduce the space requirement from $O(n^2)$. Our algorithm is wasteful in that it maintains the points of intersection of the shortest path map with each edge, and there can be a quadratic number of such points. But the actual size of the graph whose embedding on the surface yields the subdivision is only linear (in the number of edges of the original polyhedron). It should be possible to keep track of bounding hyperbolas (or lines) instead of endpoints of intervals, and then to propagate the hyperbolas across new edges without creating new structures all the time. (This also suggests that we could have allowed the algorithm to build the entire subdivision (including the partitioning of the interiors of faces) "on the fly," keeping track of the events of crossing hyperbolic boundaries, just as we did in the procedure described in § 8.) We must be careful, though, that the subdivision we build is stored in such a way that point location queries can be answered efficiently (e.g., in time $O(\log n)$). Mount [12] has shown that a subdivision can be built for a polyhedral surface and stored in a data structure of size $O(n \log n)$, and that point location queries can then be answered in time $O(\log n)$. Can this construction be built "on the fly" during the execution of continuous Dijkstra? Note that for the special case of convex polyhedra, Mount [10] has been able to reduce the space requirements to $O(n \log n)$.

Another obvious open problem is to find an efficient algorithm for determining shortest paths in three dimensions in the presence of a general collection of polyhedral obstacles. The geodesic problem is one special case in which an efficient algorithm exists, but we conjecture certain that there are other special cases which may have enough structure to yield simple efficient algorithms.

Acknowledgment. We would like to thank the referees for a careful reading and helpful suggestions for the exposition.

REFERENCES

[1] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[2] M. R. GAREY, D. S. JOHNSON, F. P. PREPARATA AND R. E. TARJAN, *Triangulating a simple polygon*, Inform. Process. Lett., 7 (1978), pp. 175–179.

[3] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28–35.

[4] D. T. LEE, *Proximity and reachability in the plane*, Ph.D. thesis, Technical Report ACT-12, Coordinated Science Laboratory, Univ. of Illinois, Chicago, IL, November 1978.

[5] D. T. LEE AND F. P. PREPARATA, *Euclidean shortest paths in the presence of rectilinear boundaries*, Networks, 14 (1984), pp. 393–410.

[6] T. LOZANO-PEREZ AND M. A. WESLEY, *An algorithm for planning collision-free paths among polyhedral obstacles*, Comm. ACM, 22 (1979), pp. 560–570.

[7] L. A. LYUSTERNIK, *Shortest Paths: Variational Problems*, Macmillan, New York, 1964.

[8] J. S. B. MITCHELL, *Shortest paths in the plane in the presence of obstacles*, Manuscript, Dept. of Operations Research, Stanford Univ., Stanford, CA, 1984.

[9] ———, *Planning shortest paths*, Ph.D. thesis, Dept. of Operations Research, Stanford Univ., Stanford, CA, August, 1986. (Available as Research Report 561, Artificial Intelligence Series, No. 1, Hughes Research Laboratories, Malibu, CA.)

[10] D. M. MOUNT, *On finding shortest paths on convex polyhedra*, Technical Report 1495, Dept. of Computer Science, Univ. of Maryland, Baltimore, MD, 1985.

[11] ———, *Voronoi diagrams on the surface of a polyhedron*, Technical Report 1496, Dept. of Computer Science, Univ. of Maryland, Baltimore, MD, 1985.

[12] ———, *Storing the subdivision of a polyhedral surface*, Proc. 2nd ACM Symposium on Computational Geometry, Yorktown Heights, NY, June 2-4, 1986.

[13] J. O'ROURKE, S. SURI AND H. BOOTH, *Shortest paths on polyhedral surfaces*, Manuscript, The Johns Hopkins Univ., Baltimore, MD, 1984.

[14] C. H. PAPADIMITRIOU, *An algorithm for shortest-path motion in three dimensions*, Inform. Process. Lett., 20 (1985), pp. 259-263.

[15] F. P. PREPARATA, *A new approach to planar point location*, this Journal, 10 (1981), pp. 473-482.

[16] J. H. REIF AND J. A. STORER, *Shortest paths in Euclidean space with polyhedral obstacles*, Technical Report CS-85-121, Computer Science Dept., Brandeis Univ., Waltham, MA, April, 1985.

[17] M. SHARIR AND A. SCHORR, *On shortest paths in polyhedral spaces*, this Journal, 15 (1986), pp. 193-215.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

© 1987 Society for Industrial and Applied Mathematics
006

A 1.6 APPROXIMATION ALGORITHM FOR ROUTING MULTITERMINAL NETS*

TEOFILO F. GONZALEZ† AND SING-LING LEE‡

Abstract. The problem of connecting a set of n terminals belonging to m (signal) nets that lie on the sides of a rectangle to minimize the total area is discussed. We present an $O(n(m+\log n))$ approximation algorithm to solve this problem. Our algorithm generates a solution with area $\leqq 1.6 * \text{OPT}$, where OPT is the area of an optimal solution. The nets are routed according to the following greedy strategy: the wire connecting all points from a net is one whose path crosses the least number of corners of the rectangle. For some nets there are several routes that cross the least number of corners. A subset of these nets is connected by wires whose paths blend with the paths for other nets. The remaining nets are routed using several strategies and 2^6 layouts are obtained. The best of these layouts is the solution generated by our algorithm.

Key words. algorithms, wire routing, VLSI, minimize area, greedy methods

AMS(MOS) subject classification. 68

1. Introduction. Let G be a rectangle and S be a set of n points (terminals) that lie on the sides of G. Let $N_1, N_2, \cdots, N_m$ be any partition of set S. Each subset N_i is called a net. All the points in each net have to be made electrically common by interconnecting them with wires. The wires follow a path consisting of a finite number of horizontal and vertical line segments. These line segments are assigned to two different layers. All the horizontal line segments are assigned to one layer and all the vertical ones are assigned to the other layer. Line segments on different layers can be connected directly at any given point z by a wire perpendicular to the layers if both line segments cross point z in their respective layers. Every pair of distinct and parallel line segments must be at least $\lambda > 0$ units apart and every line segment must be at least λ units from each side of G, except in the region where the path joins a point in S it connects. Also, no path is allowed inside of G on any of the layers.

Problem R1M (routing around one module) consists of specifying the paths for all the wires in such a way that the total area is minimized, that is, placing G together with all the wires (that must satisfy the restrictions imposed above) inside a rectangle (with the same orientation as G) of least possible area. This problem has applications in the layout of integrated circuits ([L] and [R]) and conforms to a set of design rules for VLSI systems [MC].

The 2-R1M is defined similarly, except that all nets are restricted to be of size two. Hashimoto and Stevens present an $O(n \log n)$ algorithm to solve the R1M problem for the case when all the points in S lie on one side of G. An $\Omega(n \log n)$ lower bound on the worst case time complexity for this problem was established in [GLL]. Algorithms to solve the 2-R1M problem appear in [La] and [GL2]. The one in [GL2] is optimal with respect to the time complexity bound. If more than two layers are allowed and wire overlap is permitted, the problem becomes an NP-hard problem [SBR]. Other generalizations of the 2-R1M problem have been shown to be NP-hard [La]. Gonzalez

* Received by the editors July 1, 1986; accepted for publication December 18, 1986. A preliminary version of this paper was presented at the 22nd Allerton Conference on Communication, Control and Computing, October 1984.

† Department of Computer Science, University of California, Santa Barbara, California 93106. The work of this author was supported in part by the National Science Foundation under grant DCR-8503163.

‡ Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802.

and Lee [GL3] present an $O(n(m+\log n))$ approximation algorithm for the R1M problem that generates a solution with area $\leqq 1.69 * \text{OPT}$. In this paper we present an approximation algorithm with a worst case approximation bound of 1.6. This new algorithm is more complex than the previous one and the proof for the 1.6 bound is much more elaborate than the one for the 1.69 bound that appears in [GL3]. The introduction and definitions in this paper are similar to the ones in [GL3].

The first few steps of our procedure follow the initial steps of the algorithms in [La] and [GL2]. These steps reduce our problem to the one of selecting the starting point for the path connecting each global net (a net is called a global net if at least two of its terminals are located on opposite sides of G). The algorithm routes all nets by a path that crosses the least number of corners of G. For some nets there are several paths that cross the least number of corners. A subset of these nets is connected by paths that blend with the paths connecting other nets. The paths connecting the remaining nets are selected using several strategies and 2^6 layouts are obtained. The best of these layouts is the solution generated by our algorithm.

For $i=2, 3, 4$, let R1M-i denote an R1M problem in which all global nets contain terminals on exactly i sides of G. In § 2 we introduce our notation and present some basic results. In order to simplify the exposition of our results, we begin by presenting approximation algorithms for restricted versions of the R1M problem. In § 3 we present an approximation algorithm for the R1M-3 problem. An approximation algorithm for the R1M-4 problem is presented in § 4 and in § 5 we present an approximation algorithm for the R1M-2 problem. In § 6 we combine the results obtained in §§ 3–5 to obtain our 1.6 approximation algorithm for the R1M problem.

The algorithm for the R1M-3 problem is identical to the one in [GL3]. The analysis of this algorithm is a little bit more complex in this paper because one needs to prove the 1.6 bound. The analysis in [GL3] only proves the bound of 1.66. The algorithm for the R1M-2 is a generalization of the one in [GL3]. The main difference is that instead of constructing 2^3 layouts, one has to consider 2^6 layouts. In the former case we can only claim solutions within 69% of optimal, whereas in the latter case the approximation bound is 1.60. In both cases the analysis of the approximation bound is tight, i.e., it cannot be improved. The analysis in this paper for the 2^6 layouts algorithm is much more elaborate than the one in [GL3] for the 2^3 layouts algorithm. The algorithm for the R1M-4 problem is much more complex than the one in [GL3]. For brevity we did not include the postprocessing procedure for the R1M-2 problem in [GL3]. This procedure as well as its formal justification are included in this paper. A significant improvement over the 1.6 approximation bound seems impossible if one is restricted to our techniques. Perhaps a better approximation bound can be obtained by routing all nets concurrently; however, we doubt that the analysis could be simplified.

2. Basic results. We begin by defining the R1M problem and introducing notation similar to the one in [GL1]. Let G be a rectangular component of size h by w (height by width). There are n terminals $(T_1, T_2, \cdots, T_n)$ on the sides of G. These terminals are partitioned into m subsets denoted by $N_1, N_2, \cdots, N_m$. Each subset N_i is called a net and it is assumed that $|N_i|>1$ for all i. The problem depicted in Fig. 2.1 consists of five nets: $N_1=\{T_2, T_4, T_7\}$, $N_2=\{T_1, T_3, T_{10}, T_{14}\}$, $N_3=\{T_5, T_{12}\}$, $N_4=\{T_6, T_8, T_9\}$ and $N_5=\{T_{11}, T_{13}, T_{15}\}$. It is assumed that every pair of terminals is at least $\lambda>0$ units apart and every terminal is located at least λ units from each of the corners of G. All the terminals in each net must be made electrically common by connecting them with wires. The path followed by these wires can be partitioned into a finite number of straight line segments. Each of these line segments must lie on the same plane as

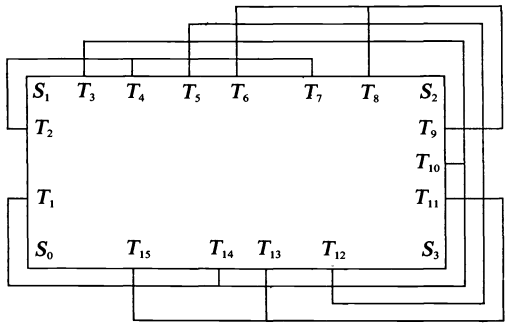


FIG. 2.1

G, be on the outside of G and be parallel to a side of G. Perpendicular line segments can intersect at any point, but parallel line segments must be at least λ units apart. Also, all line segments must be at least λ units away from every side of rectangle G except in the vicinity where a line segment connects a terminal. The *R1M problem* consists of specifying paths for all the interconnections subject to the rules mentioned above in such a way that the total area is minimized, i.e., place the component together with all the wires inside a rectangle (with the same orientation as G) of least possible area.

Label the sides of the component (in the obvious way) left, top, right and bottom. Starting in the bottom left-hand corner of G, traverse the sides of the rectangle clockwise. The ith corner to be visited is labeled S_{i-1}. Assume that the ith terminal visited is terminal T_i. The *close interval* $[x, y]$, where x and y are the corners of G or the terminals T_i, consists of all the points on the sides of G that are visited while traversing the sides of G in the clockwise direction starting at point x and ending at point y. Note that the interval $[x, x]$ consists of a single point. Parentheses are used instead of square brackets for open intervals. We use $[S_0, S_1]$, $[S_1, S_2]$, $[S_2, S_3]$ and $[S_3, S_0]$; 0, 1, 2 and 3; L, T, R and B; and L', T', R' and B' to represent the left, top, right and bottom sides of G, respectively. Terminal T_i is said to belong to side l, $S(i) = l$, if T_i is located in $[S_l, S_{(l+1) \bmod (4)}]$.

The function $I(j)$ indicates the index of the net to which terminal T_j belongs. The function $L(j)$ is defined in such a way that the interval $[T_j, T_{L(j)}]$ is the smallest interval that includes all the terminals from net $N_{I(j)}$. Set $D = \{d_1, d_2, \cdots, d_m\}$ is said to be an *assignment* if D contains exactly one index of a terminal from each net. Any subset of an assignment is said to be a *partial assignment.* An assignment D indicates the starting point for the path connecting all the points in each net. If $i \in D$ then the path connecting all the terninals in net $N_{I(i)}$ starts at terminal T_i moving perpendicular to side $S(i)$ and then it continues in the clockwise direction with respect to G until it reaches $T_{L(i)}$. Each terminal, T_k, in net $N_{I(i)}$ is joined to this path by a line segment perpendicular to side $S(T_k)$. The starting point for the paths connecting some nets might not be defined in a partial assignment. The assignment for the layout given by Fig. 2.1 is $\{2, 3, 5, 6, 11\}$. For any $l \in D$, we say that the path connecting net $N_{I(l)}$ given by D *crosses* point z if $z \in [T_l, T_{L(l)}]$. If $i \in D$, we say that the path connecting net $N_{I(i)}$ *begins* at point T_i and *ends* at point $T_{L(i)}$.

For any assignment (or partial assignment) D we define the *height function* H_D for $x, y \in \{T_1, T_2, \cdots, T_n\} \cup \{S_0, S_1, S_2, S_3\}$ as follows:

$$H_D(x, y) = \max \{\text{number of paths given by } D \text{ that cross point } z \mid z \in [x, y]\}.$$

For example, $H_D(S_0, S_1)$ is 1, $H_D(T_5, T_5)$ is 3 and $H_D(S_2, S_3)$ is 3 for the assignment,

D, whose layout appears in Fig. 2.1. We shall refer to $H_D(x, y)$ as the *height of assignment D for the interval $[x, y]$ of G*. The *height of assignment D for side X* ($X \in$ {*top, right, bottom. left*}) refers to the value of $H_D(x, y)$, where $[x, y]$ is the interval that represents side X. The *vertical* (*horizontal*) *height of assignment D* is the height on the top (left) plus the height on the bottom (right) side of G. The *total height of assignment D* is equal to the vertical plus the horizontal height of assignment D.

The next two lemmas establish that the R1M problem reduces to the problem of finding an assignment D with least

$$(h + (H_D(S_1, S_2) + H_D(S_3, S_0)) * \lambda) * (w + (H_D(S_0, S_1) + H_D(S_2, S_3)) * \lambda)$$

and then in $O(n \log n)$ time one can construct a layout of area h_Q by w_Q for it.

LEMMA 2.1. *For every assignment D, there is a rectangle Q of size h_Q by w_Q, where*

$$h_Q = h + (H_D(S_1, S_2) + H_D(S_3, S_0)) * \lambda,$$

$$w_Q = w + (H_D(S_0, S_1) + H_D(S_2, S_3)) * \lambda$$

with the property that rectangle G together with the interconnecting paths defined by D can be made to fit inside Q.

Proof. The proof is a direct generalization of the proof for the 2-R1M problem that appears in [La]. □

LEMMA 2.2. *For any assignment D a layout with the area given by Lemma* 2.1 *can be obtained in $O(n \log n)$ time.*

Proof. The proof of this lemma is a straightforward generalization of the proof for the 2-R1M problem that appears in [La]. The algorithm that constructs the final layout uses as a subalgorithm the procedure given in [GLL] and [HS]. □

Net N_i is said to be a *global net* if at least two of its terminals are located on opposite sides of G. Net N_i is said to be *local* otherwise, i.e., if all its terminals are located on the same side of G or on two adjacent sides of G. Nets $N_2 = \{T_1, T_3, T_{10}, T_{14}\}$ and $N_3 = \{T_5, T_{12}\}$ are the only global nets in the problem depicted in Fig. 2.1. For assignment D we define the *function $A(D)$* as

$$(h + (H_D(S_1, S_2) + H_D(S_3, S_0)) * \lambda) * (w + (H_D(S_0, S_1) + H_D(S_2, S_3)) * \lambda),$$

i.e., the total area required for a layout of G and all the interconnections given by D.

DEFINITION 2.1. Let D' be the partial assignment in which all the local nets are connected by paths crossing the least number of corners of G.

LEMMA 2.3. *Every assignment D can be transformed to an assignment M such that $D' \subseteq M$ and $A(M) \leqq A(D)$.*

Proof. The proof follows the same line as the one for the 2-R1M problem that appears in [La]. □

Lemma 2.3 shows that for any instance of the R1M problem there exists an optimal solution in which all local nets are connected by paths crossing at most one corner of G. The R1M problem has been reduced to the problem of finding the starting point (by default the direction is clockwise) for the paths connecting all the global nets in the presence of the partial assignment D'. At this point our algorithm abandons the procedures given in [La] and [GL2]. The main difficulty that we encounter in extending the results for the 2-R1M problem to this problem is that the divide-and-conquer step seems not applicable. The reason for this is that there seems to be no rule to separate the nets with terminals located in three or four sides of G into groups that can be optimally routed, independently of each other.

It should be clear that it is only required to specify the paths connecting all the global nets since we know that local nets can be routed optimally by routing them as

indicated in assignment D'. Also, once we have an assignment the proof of Lemma 2.2 (a constructive proof) can be used to find a layout of optimal area for it. In the next three sections we present approximation algorithms for the R1M-i problem, $2 \leqq i \leqq 4$. In § 6 we indicate how to combine these results to obtain our 1.6 approximation algorithm.

3. Approximation algorithm for the R1M-3 problem. In this section we present an approximation algorithm for the R1M-3 problem. Let M_3 be the set of global nets with terminals located on exactly three sides of G. Clearly, all global nets belong to set M_3. For the set of nets M_3 we construct assignment D_3 as follows:

$D_3 = \{$all nets in M_3 are connected by paths that cross the least number of corners of $G\}$.

Let $M_3^{TB}(M_3^{LR})$ be the set of all nets in M_3 without terminals located on either the left or right (top or bottom) side of G. Let m_3^{TB} and m_3^{LR} represent the number of elements in M_3^{TB} and M_3^{LR}, respectively. In Fig. 3.1(a) we give a layout for the assignment constructed by our algorithm for a net in M_3^{TB}. Suppose that this net is routed as in Fig. 3.1(b) in an optimal assignment D. Let M be D except for the path connecting the net that appears in Fig. 3.1(b) which is connected as indicated in Fig. 3.1(a). It is simple to show that

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + 1,$$
$$H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3).$$

A straightforward generalization of the above observation is given by Lemma 3.1.

LEMMA 3.1. *Let D be an optimal assignment such that $D' \subseteq D$. Let M be D except that all nets in $M_3^{TB} \cup M_3^{LR}$ which are assigned as in our algorithm. Then*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + x_3^{TB},$$
$$H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + x_3^{LR}$$

where x_3^{TB} (x_3^{LR}) is the number of nets in X_3^{TB} (X_3^{LR}), and X_3^{TB} (X_3^{LR}) is a set of all nets in M_3^{TB} (M_3^{LR}) that are connected differently in D and M.

Proof. For brevity the proof is not included. □

Before proving our main result on this section, we establish a lower bound on the area required by any optimal solution. Our lower bound is given in Table 3.1.

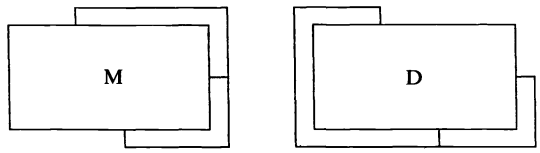


FIG. 3.1

TABLE 3.1
Lower bounds for the R1M-3 *problem.*

Set	Contribution to our lower bound for $h/(\lambda) + H_D(S_1, S_2) + H_D(S_3, S_0)$	Contribution to our lower bound for $w/(\lambda) + H_D(S_0, S_1) + H_D(S_2, S_3)$
M_3^{TB}	$1.5m_3^{TB} + 0.5x_3^{TB}$	$2m_3^{TB} + 0.5x_3^{TB}$
M_3^{LR}	$2m_3^{LR} + 0.5x_3^{LR}$	$1.5m_3^{LR} + 0.5x_3^{LR}$

LEMMA 3.2. *Let D be an optimal assignment such that $D' \subseteq D$. Assignment D and rectangle G satisfy the lower bounds given in Table* 3.1.

Proof. We only prove the lower bounds in the first column of Table 3.1 since the proof for the bounds in the second column is similar. The proof for these bounds can be obtained by adding the lower bounds given in Table 3.2. Therefore, we only need to prove the lower bounds given in Table 3.2.

Each net in M_3^{TB} has at least one terminal located on the left or right side of G and each net in M_3^{LR} has at least one terminal located on the left and on the right sides of G. Therefore, either on the left or the right side of G there are at least $0.5m_3^{TB}+m_3^{LR}$ terminals. Since every pair of these terminals is at least λ unit apart, we know that $h \geqq (0.5m_3^{TB}+m_3^{LR})\lambda$. This lower bound is given by the first column of Table 3.2.

TABLE 3.2

Set	Contribution to our lower bound for $h/(\lambda)$	Contribution to our lower bound for $H_D(S_1, S_2)+H_D(S_3, S_0)$
M_3^{TB}	$0.5m_3^{TB}$	$m_3^{TB}+0.5x_3^{TB}$
M_3^{LR}	m_3^{LR}	$m_3^{LR}+0.5x_3^{LR}$

Let us now establish a lower bound on the number of paths crossing the corners of G. Since in D every path connecting a net in $X_3^{TB} \cup X_3^{LR}$ crosses at least three corners of G and every path connecting a net in $(M_3^{TB}-X_3^{TB}) \cup (M_3^{LR}-X_3^{LR})$ crosses at least two corners of G, we know that

$$\sum_{i=0}^{3} H_D(S_i, S_i) \geqq 2(m_3^{LR}-x_3^{LR}+m_3^{TB}-x_3^{LR})+3(x_3^{LR}+x_3^{TB}).$$

Since the height on the top (bottom) side of G is at least as large as the average height of the two top (bottom) corners of G, we know that

$$H_D(S_1, S_2)+H_D(S_3, S_0) \geqq 0.5 * (H_D(S_1, S_1)+H_D(S_2, S_2)) + 0.5 * (H_D(S_3, S_3)+H_D(S_0, S_0)).$$

The lower bound for the second column of Table 3.2 follows from the above inequalities. The proof for the lower bounds for the first column of Table 3.1 is obtained by adding the bounds given by Table 3.2. This completes the proof of the lemma. □

THEOREM 3.1. *For the* R1M-3 *problem, let D be an optimal assignment such that $D' \subseteq D$ and let M be the assignment generated by our algorithm. Then, $A(M) \leqq 1.6 * A(D)$.*

Proof. From Lemma 3.1 and 3.2 we know that

$$\frac{A(M)}{A(D)} \leqq \left(1+\frac{a}{b}\right) * \left(1+\frac{c}{d}\right)^{1}$$

where

$$a = x_3^{TB},$$
$$b = 1.5m_3^{TB}+0.5x_3^{TB}+2m_3^{LR}+0.5x_3^{LR},$$
$$c = x_3^{LR},$$
$$d = 2m_3^{TB}+0.5x_3^{TB}+1.5m_3^{LR}+0.5x_3^{LR}.$$

[1] Note that a, b, c and d are nonnegative. If b or d is zero, the a, b, c and d will be zero and our assignment is optimal.

Since $m_3^{TB} \geqq x_3^{TB}$ and $m_3^{LR} \geqq x_3^{LR}$, we only need to prove that

$$(1+x/(2x+2.5y)) * (1+y/(2.5x+2y)) \leqq 1.5$$

where x and y are positive integers. A straightforward manipulation of this inequality results in the following inequality:

$$(7.5x^2+15.25xy+7.5y^2)/(5x^2+10.25xy+5y^2) \leqq 1.5.$$

Since the remaining part of the proof is simple, it will be omitted. This completes the proof for Theorem 3.1. □

4. Approximation algorithm for the R1M-4 problem. In this section we present an approximation algorithm for the R1M-4 problem. Let M_4 be the set of all global nets with terminals located on the four sides of G and let m_4 represent the number of nets in it. Clearly, all global nets belong to set M_4. In what follows we partition the set of nets M_4 into sets that will be routed independently of each other. There are several reasons for taking this approach. The main reason is that there are basically two types of nets: the ones with a large number of terminals and the ones with few terminals. For the first set of nets it is difficult to find routings with small area, but for the second type of nets such routing can be easily obtained. In our algorithm we do not route tightly the first type of nets. This routing does not increase our approximation bound significantly since these nets contribute large values to our lower bound function. The other type of nets have to be routed tightly, since their contribution to our lower bound function is small.

Nets N_i and N_j (both in set M_4) are said to be *agreeable* if on at least one side of G no terminal from net N_i is between any two terminals from net N_j and no terminal from net N_j is between any two terminals from net N_i. Procedure PARTITION, defined below, partitions the set M_4 into the sets M_4^A, M_4^N, $M_4^{T'}$, $M_4^{R'}$, $M_4^{B'}$, $M_4^{L'}$, M_4^T, M_4^R and M_4^B. Since some of these sets consist of 1-tuples, 2-tuples and 4-tuples, by partition we mean that every element in set M_4 is in one and only one of the tuples in these sets. After the algorithm terminates, let m_4^X represent the number of tuples in M_4^X, for $X \in \{A, N, T', R', B', L', T, R, B\}$. It is very important to keep in mind that sets M_4^A, $M_4^{T'}$, $M_4^{R'}$, $M_4^{B'}$ and $M_4^{L'}$ consist of 1-tuples, set M_4^N has only 4-tuples and the remaining sets contain only 2-tuples. When algorithm PARTITION terminates, the sets our procedure generates satisfy the following properties. Every net in set M_4^A has exactly four terminals (one on each side of G). Because of this, every pair of such nets is agreeable on each of the four sides of G. Every net in a 4-tuple of set M_4^N has at least two terminals on each side of G and the four nets in each 4-tuple are totally nonagreeable nets, i.e., every pair of these nets is not agreeable (on each of the four sides of G). Each net in set M_4^X, $X \in \{T', R', B', L'\}$, has at least two terminals located on side X and exactly one terminal located on each of the remaining sides of G. Every pair of nets in M_4^X, $X \in \{T', R', B', L'\}$, is agreeable on all sides of G, except possibly on side X. Each 2-tuple in sets M_4^X, for $X \in \{T, R, B\}$, contains a pair of nets that are agreeable on at least one side of G. Every net in set M_4^X, $X \in \{T, R, B\}$ satisfies one of the following conditions:

(i) it contains at least two terminals on at least two sides of G, or

(ii) it contains at least two terminals on side X and either exactly one terminal on the opposite side of X or at least two terminals on the opposite side of X when the two nets in the tuple are agreeable on an adjacent side of X (one net might violate this condition, i.e., it has only one terminal on the opposite side of X).

PROCEDURE PARTITION
//Partition all nets with exactly one terminal located on each side of G.//
$M_4^A \leftarrow \{N_i \mid N_i \in M_4$ and $|N_i| = 4\}$;
$W \leftarrow M_4 - M_4^A$;
//Partition all nets with at least two terminals located on each side of G.//
$Z \leftarrow \{N_i \mid N_i \in W$ has at least two terminals located on each side of $G\}$;
$W \leftarrow W - Z$;
$M_4^T \leftarrow \emptyset$; $M_4^R \leftarrow \emptyset$; $M_4^B \leftarrow \emptyset$; $M_4^N \leftarrow \emptyset$;
while $|Z| \geqq 4$ *do*
  let N_i, N_j, N_k and N_l be any four nets in Z;
  if any two of these four nets are agreeable
    then delete from Z two agreeable nets (two out of the four) and add them as a pair to M_4^T if these two nets are agreeable on the left or right side of G, otherwise add these two nets as a pair to M_4^R
    else delete all four nets from Z
      $M_4^N \leftarrow M_4^N \cup \{(N_i, N_j, N_k, N_l)\}$
  endif;
endwhile;
//Later on we explain what to do when $Z \neq \emptyset$. Note that $|Z| \leqq 3$.//
//Partition all nets with at least two terminals located on some side of G and exactly one terminal located on the remaining sides of G.//

$M_4^{T'} \leftarrow \emptyset$, $M_4^{R'} \leftarrow \emptyset$, $M_4^{B'} \leftarrow \emptyset$, $M_4^{L'} \leftarrow \emptyset$;

while there is a net in W with at least two terminals located on side X and exactly one terminal located on the remaining sides of G *do*.
  Let y be one of such nets and let $X \in \{T', R', B', L'\}$ be the side on which it has more than one terminal;
  $W \leftarrow W - \{y\}$;
  $M_4^X \leftarrow M_4^X \cup \{y\}$;
endwhile;
//Partition the remaining nets, i.e., all nets with at least two terminals located on two sides of G and exactly one terminal located on another side of G.//
$Z \leftarrow \{N_i \mid N_i \in W$ has at least two terminals located on the top side of $G\}$;
$W \leftarrow W - Z$;
while there are two nets in Z with exactly one
    terminal located on the bottom side of G *do*
  delete two of such nets from Z and add them as a pair to M_4^T;
endwhile;
while there are two nets in Z with exactly one
    terminal located on the same side of G *do*
  delete two of such nets from Z and add them as a pair to M_4^T
endwhile;
//Later on we explain what to do when $Z \neq \emptyset$. Note that $|Z| \leqq 3$.//
$Z \leftarrow \{N_i \mid N_i \in W$ has at least two terminals located on the right side of $G\}$;
$W \leftarrow W - Z$;
while there are two nets in Z with exactly one
    terminal located on the left side of G *do*
  delete two of such nets from Z and add them as a pair to M_4^R;
endwhile;

while $|Z| \geqq 2$ *do*
  delete any two nets from Z and add them as a pair to M_4^R
endwhile;
//Later on we explain what to do when $Z \neq \emptyset$. Note that $|Z| \leqq 1$//
$Z \leftarrow \{N_i \mid N_i \in W$ has at least two terminals located on the bottom side of $G\}$;
$W \leftarrow W - Z$;
while $|Z| \geqq 2$ *do*
  remove any two nets from Z and add them to M_4^B as a pair;
endwhile;
//Later on we explain what to do when $Z \neq \emptyset$. Note $|Z| \leqq 1$//
end of procedure

For some problem instances the algorithm fails to assign some nets to any of the sets M_4^X, $X \in \{A, N, T', R', B', L', T, R, B\}$. Let us assume that this is not the case, i.e., Z is always empty whenever we reach each of the "// later on $\cdots$ //" comments in our procedure. In § 6 we explain what to do when this is not the case. From procedure PARTITION it is simple to verify that no more than eight nets will be left out.

(a) *Construction of the assignment* D_A''. In what follows we explain how the assignment D_A'' for all the nets in set M_4^A is constructed. Figure 4.1 gives a layout for the assignment D_A'' constructed by our procedure for a set M_4^A with four nets. For any permutation, π, of the nets in set M_4^A and an integer i $(0 \leqq i \leqq 3)$ we define an assignment (denoted ASG (π, i)) as follows: the first net in π is connected by a path that begins on side i and ends on side $(i+3) \bmod (4)$; and the path connecting the kth net $(1 < k \leqq m_4^A)$ in π begins on side j and ends on side $(j+3) \bmod (4)$, where j is the side on which the path connecting the $(k-1)$st net in π ends. We claim that for every integer i $(0 \leqq i \leqq 3)$ there is a permutation, π (that depends on i) of the nets in set M_4^A such that there is a layout for assignment ASG(π, i) with the property that for any k $(1 < k \leqq m_4^A)$ the path connecting the $(k-1)$st net in π and the path connecting the kth net in π can share the same track on the side where the path connecting the $(k-1)$st net ends. In this case we say that π *is a valid permutation with respect to* i.

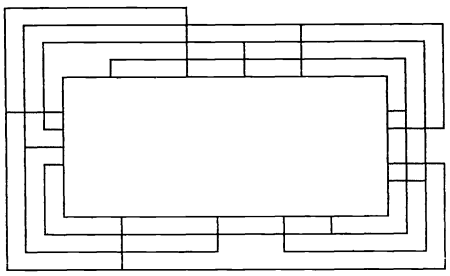


FIG. 4.1. *Assignment* D_A'' *for the case when* $m_4^A = 4$.

CLAIM. For every integer i, $0 \leqq i \leqq 3$, there is a valid permutation for the set of nets in M_4^A.

Proof. The proof of this claim is by induction on the number of nets in M_4^A. Clearly, the claim is true when set M_4^A has only one net. Assume that the claim holds when there are $k \geqq 1$ nets. We now show the claim holds when there are $k+1$ nets in M_4^A. Let $N_j \in M_4^A$ be the net with the topmost terminal on the left side of G. By the induction hypothesis we know that there is a valid permutation π' with respect to some i for the remaining k elements in M_4^A such that the path connecting the last net in the permutation ends on the left side of G. Since N_j was selected to be the net with

the topmost terminal located on the left side of G, we know permutation $\pi = (\pi', N_j)$ is a valid permutation for some value of i. To show the existence of the three other permutations (for the other values of i) we repeat the above procedure three times. The first time we let N_j be the net with the rightmost terminal located on the top side of G, the second time N_j is the net with the bottommost terminal located on the right side of G, and the last time N_j is the net with the leftmost terminal located on the bottom side of G. This completes the proof of the claim.

A valid permutation, π, for $i = 1$ can be constructed by a simple recursive procedure. Once π is obtained, the assignment $D''_A = \text{ASG}(\pi, i)$ can be easily constructed. Figure 4.1 gives a layout for the assignment D''_A constructed by our procedure for set M_4^A with four nets. Note that the contribution to the total height of the assignment is at most $3 * m_4^A + 1$.

(b) *Construction of the assignment* D''_N. The assignment D''_N is constructed by applying the following rule to each 4-tuple in M_4^N. Let (N_i, N_j, N_k, N_l) be any tuple in set M_4^N. Let us assume that these nets have been ordered in such a way that N_i is the net (amongst these four nets) whose bottommost terminal located on the left side of G is closest to the top side of G; net N_j is the net (amongst N_j, N_k and N_l) whose rightmost terminal located on the bottom side of G is closest to the left side of G; and net N_l is the net (amongst N_k and N_l) whose leftmost terminal located on the top side of G is closest to the right side of G. A layout for the assignment constructed for these nets is given in Fig. 4.2.

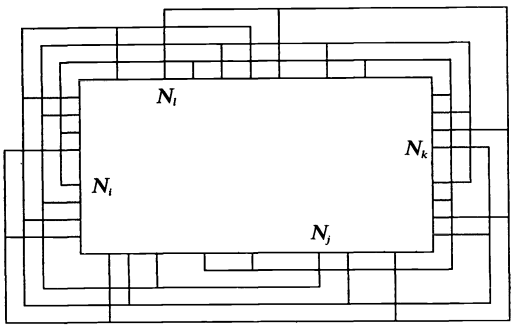


FIG. 4.2. *Assignment for a tuple in* M_4^N.

(c) *Construction of assignment* D''_X *for* $X \in \{T', R', B', L'\}$. Since all the assignments D''_X, $X \in \{T', R', B', L'\}$, are constructed by similar procedures, we only explain the one for assignment $D''_{T'}$. Assignment $D''_{T'}$ for $M_4^{T'}$ is constructed by applying a rule similar to the one used in part (a). The main difference is that we are only interested in valid permutations with respect to $i = 1$ and in a valid permutation it is not required for the $(k-1)$st net in π and the kth net in π to share the same track on the side where the path connecting the $(k-1)$st net ends (for $k = 5, 9, 13, \cdots$). Once a valid permutation is obtained the assignment $D''_{T'} = \text{ASG}(\pi, 1)$ can be easily constructed. Note that the total height of the assignment is at most $3m_4^{T'} + [m_4^{T'}/4]$.

(d) *Construction of assignment* D''_X *for* $X \in \{T, R, B\}$. Assignment D''_X, $X \in \{T, R, B\}$, for the set M_4^X is constructed by applying the following rule to each 2-tuple in M_4^X. Let (N_i, N_j) be any 2-tuple in set M_4^X. From algorithm PARTITION we know that both of these nets are agreeable on at least one side of G because either both of these nets have exactly one pin located on some side of G, or both of the nets have at least two terminals on each side of G and are agreeable on a side adjacent to X.

The pairs that have exactly one terminal on some side of G are assigned in such a way that there is a layout in which these nets share the same track on side Z of G and both nets have exactly one terminal located on side Z. Side Z is selected using the following priorities: the opposite side of X has the highest priority; and the adjacent sides of G have the lowest priorities. Note that side Z cannot be the same as side X. Figure 4.3 shows a layout for the assignment for a pair of nets in M_4^B with Z being the top side of G. The remaining pairs are assigned in such a way that they share the same track on a side adjacent to X.

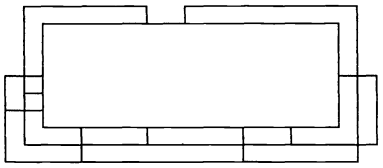


FIG. 4.3. *Layout for a 2-tuple in M_4^B with Z being the top side of G.*

(e) *Final assignment and proof of approximation bound.* Let $D_4 = D''_A \cup D''_N \cup D''_{T'} \cup D''_{R'} \cup D''_{B'} \cup D''_{L'} \cup D''_T \cup D''_R \cup D''_B$. The output of our approximation algorithm is $D_4 \cup D'$. Before showing that our algorithm generates a solution within 60% of the optimal solution value, we need to establish upper bounds for the area of the layouts obtained from assignment D_4 and prove lower bounds for the area of an optimal solution.

In what follows we assume that m_4^A is a multiple of four. In § 6 we indicate what to do when this is not the case.

LEMMA 4.1. *Let D be an optimal assignment such that $D' \subseteq D$. Let M be D except that all the nets in M_4^A are routed as in our approximation algorithm. Then*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + 0.5m_4^A + 1,$$

$$H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + 0.5m_4^A.$$

Proof. From the construction rule for assignment D''_A (a), we know that the contribution of the nets in M_4^A to the total height of any assignment that includes D''_A is at most $3m_4^A + 1$. Also, for any assignment that includes D''_A the maximum contribution to the vertical height by the paths connecting the nets in M_4^A is at most $1.5m_4^A + 1$ and the maximum contribution by these nets to the horizontal height is at most $1.5m_4^A$. In an optimal assignment each path connecting a net in M_4^A must contribute to the vertical and horizontal height at least one. Therefore, when transforming D to M the total increase in vertical and horizontal height is given by the inequalities in the statement of the lemma. This completes the proof of the lemma. □

In what follows we assume that m_4^X, $X \in \{T', R', B', L'\}$, is a multiple of four. In § 6 we indicate what to do when this is not the case.

LEMMA 4.2. *Let D be an optimal assignment such that $D' \subseteq D$. Let M be D except for all the nets in M_4^X, $X \in \{T', R', B', L'\}$, are routed as in our approximation algorithm. Then*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + x * m_4^X,$$

$$H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + y * m_4^X$$

where

$$x = \begin{cases} 0.75 & \text{if } X \in \{T', B'\}, \\ 0.5 & \text{otherwise,} \end{cases} \qquad y = \begin{cases} 0.5 & \text{if } X \in \{T', B'\}, \\ 0.75 & \text{otherwise.} \end{cases}$$

Proof. Since the proof of all cases is similar, we only prove the case for $X = T'$. From the construction rule for assignment $D''_{T'}$ (c), we know that the contribution from the nets in set $M_4^{T'}$ to the total vertical height of an assignment that include $D''_{T'}$ is at most $(1.75) * m_4^{T'}$ and with respect to the total horizontal height is at most $(1.5) * m_4^{T'}$. On any optimal assignment, D, the contribution to the vertical and horizontal height by these nets is at least $m_4^{T'}$. Hence, when transforming D to our assignment, the difference in heights is given by the inequalities in the statement of the lemma. This completes the proof of the lemma. □

For $X \in \{T, R, B\}$ let p_4^X be the faaction of pairs in M_4^X that are connected in D''_X by paths that do not overlap on the top or bottom side of G, and $q_4^X = 1 - p_4^X$. Since all pairs of nets in M_4^B are agreeable on the top side of G, we know from our routing algorithm that $q_4^B = 0$.

LEMMA 4.3. *Let D be an optimal assignment such that $D' \subseteq D$. Let M be D except that all nets in M_4^X, $X \in \{T, R, B\}$, are routed as in our approximation algorithm. Then*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + p_4^X * m_4^X + 2q_4^X * m_4^X,$$

$$H_M(S_2, S_3) + H_M(S_0, S_1) \leqq H_D(S_2, S_3) + H_D(S_0, S_1) + 2p_4^X * m_4^X + q_4^X * m_4^X.$$

Proof. Since the proof of all three cases is similar we only prove the case for $X = T$. In assignment D each net with terminals located on the four sides of G contributes to the vertical height and to the horizontal height at least one unit. Therefore, the contribution to the vertical and horizontal heights of D for a pair of nets in M_4^T is at least two. For any assignment a pair of nets routed in such a way that they can share the same track on the top or bottom side of G contributes to the vertical height at most three and their contribution to the horizontal height is at most four. For the nets routed following this sharing rule, we know that transforming D to M increases the vertical and horizontal height as indicated by the inequalities in the statement of the lemma. A similar result can be obtained for pairs of nets that are routed in such a way that they share a track on the left or right side of G. This completes the proof of the lemma. □

LEMMA 4.4. *Let D be an optimal assignment such that $D' \subseteq D$. Let M be D except that all nets in M_4^N are routed as in our approximation algorithm. Then*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + 7x * m_4^N,$$

$$H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + 7y * m_4^N,$$

where x, y are nonnegative reals such that $x + y = 1$.

Proof. One can easily show that we only need to prove that when assigment D is transformed to assignment M the total height increases by at most seven for each tuple in M_4^N. Let us now prove that this statement holds for any tuple, H, in M_4^N. There are two cases.

Case 1. At least one of the paths connecting a net in H begins and ends on the same side of G in D.

For this case we know that the contribution to the total height in D is at least three for the nets connected in D by paths that begin and end on the same side of G. The remaining connecting paths contribute at least two units to the total height.

Therefore the contribution to the total height of D by these nets is at least nine. On assignment, D''_N the contribution to the total height for the same set of nets is at most sixteen. Hence, the total height increases by at most seven after the transformation.

Case 2. In assignment D, no path connecting a net in H begins and ends on the same side of G.

There are four subcases depending on how the nets in H are connected.

Subcase 2.1. In assignment D, two of the paths connecting nets in H begin on adjacent sides of G.

The paths connecting these two nets are depicted in Fig. 4.4. It is simple to see that the contribution to the total height of assignment D by these two nets is at least five. The contribution to the total height of D by the remaining nets is at least four. The proof now follows similar steps to the ones in Case 1.

Subcase 2.2. In assignment D all the paths connecting nets in H begin on the same side of G.

Since the paths connecting all the nets in H begin on the same side of G, we know that only three nets from H are connected differently in D and D''_N. Each time the path connecting one of these nets is changed, the maximum increase of the total height is at most two. Hence, the total height increases by at most six.

Subcase 2.3. In assignment D the paths connecting all the nets in H begin on the left or the right side of G and at least one of these paths begins on each of these two sides.

Assume that no net in H is connected identically in D''_N and D, as otherwise the proof of Subcase 2.2 can be used to complete the proof for this case. The total height increases every time a path is interchanged by at most two, except a path that begins on the left side of G in D''_N. This is because x is always greater than y (see x and y in Fig. 4.5). Remember that such a net was selected to be the one whose bottommost element on the left side of G is closest to the top side. Hence the total height increases by at most seven units.

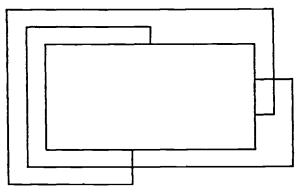


FIG. 4.4

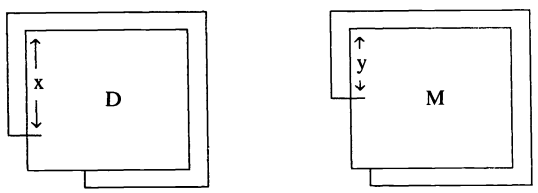


FIG. 4.5

Subcase 2.4. In assignment D the paths connecting all nets in H begin on the top or bottom side of G and at least one of these paths begins on each of these two sides.

Assume that no net in H is connected identically in D''_N and D, as otherwise the proof of Subcase 2.2 can be used to complete the proof for this case. The remaining part of the proof uses similar arguments to the ones used in the proof for Subcase 2.3 because either at least two of these paths begin on the bottom side of G or at least three of these paths begin on the top side of G. Since the connecting path that begins on the bottom side of G in our solution is the second "best" and the one that begins on the top side of G in our solution is the third "best," similar arguments to the ones used in Subcase 2.3 can be used to complete the proof for this subcase.

This completes the proof for Case 2 and the lemma. □

Before proving our main result in this section we establish a lower bound on the area required by any optimal solution. The lower bound is given in Table 4.1.

TABLE 4.1
Lower bounds for the R1M-4 *problem.*

Set	Contribution to our lower bound for $h/(\lambda)+H_D(S_1,S_2)+H_D(S_3,S_0)$	Contribution to our lower bound for $w/(\lambda)+H_D(S_0,S_1)+H_D(S_2,S_3)$
M_4^A	$2.5m_4^A$	$2.5m_4^A$
M_4^N	$14m_4^N$	$14m_4^N$
$M_4^{T'} \cup M_4^{B'}$	$2.5m_4^{T'}+2.5m_4^{B'}$	$3m_4^{T'}+3m_4^{B'}$
$M_4^{R'} \cup M_4^{L'}$	$3m_4^{R'}+3m_4^{L'}$	$2.5m_4^{R'}+2.5m_4^{L'}$
$M_4^T \cup M_4^B$	$p_4^T * m_4^T+5m_4^T+6m_4^B$	$q_4^T * m_4^T+6m_4^T+6m_4^B$
M_4^R	$p_4^R * m_4^R+6m_4^R$	$q_4^R * m_4^R+5m_4^R$

LEMMA 4.5. *Let D be an optimal assignment such that $D' \subseteq D$. Assignment D and rectangle G satisfy the lower bounds given in Table* 4.1.

Proof. We only prove the lower bounds in the first column of Table 4.1 since the proof for the bounds in the second column is similar. Before establishing this bound, we prove some intermediate lower bounds. It is simple to prove the following lower bounds:

(a) Each net in M_4^A has exactly one terminal on the left and right sides of G.

(b) The four nets in each tuple in M_4^N have at least eight terminals (two per net) on the left and right sides of G.

(c) Each net in M_4^X, $X \in \{T', B'\}$, has exactly one terminal on the left and right side of G. Each net in M_4^X, $X \in \{R', L'\}$ has at least two terminals on side X and exactly one terminal on the opposite side of X.

(d) Each net in each pair of nets in M_4^T routed in D''_T by sharing a track on the bottom side of T has at least one terminal on the left and right sides of G and at least two terminals on either the left or the right side of G. Each remaining pair in M_4^T has at least two terminals (one per net) on the left and right sides of T. Each pair of nets in M_4^B has at least two terminals on the right side (one per net) and at least four terminals on the left side (two per net). Each pair of nets in M_4^R routed in D''_R by sharing a track on the left side of T has at least two terminals (one per net) on the left side of G and at least four terminals (two per net) on the right side of T. The nets

in the remaining pairs in M_4^R have at least four terminals (two per net) on the left and right sides of G.

Using the above observations we obtain the bounds given by the first column of Table 4.2.

Let us now establish a lower bound on the number of paths crossing the corners of G. Since the path connecting every net in $M_4^A \cup M_4^{T'} \cup M_4^{R'} \cup M_4^{B'} \cup M_4^{L'}$ crosses at least three corners of G; the paths connecting every pair of nets M_4^X, $X \in \{T, B, L\}$ cross at least six corners (each path crosses three corners) of G and the paths connecting each four nets in a tuple of M_4^N cross at least twelve corners (each path crosses at least three corners) of G, we know that

$$\sum_{i=0}^{3} H_D(S_i, S_i) \geqq 3m_4^A + 12m_4^N + 3m_4^{T'} + 3m_4^{R'} + 3m_4^{B'} + 3m_4^{L'} + 6m_4^T + 6m_4^R + 6m_4^B.$$

It is simple to show that

$$H_D(S_1, S_2) + H_D(S_3, S_0) \geqq 0.5 * (H_D(S_1, S_1) + H_D(S_2, S_2)) + 0.5 * (H_D(S_3, S_3) + H_D(S_0, S_0)).$$

The lower bound for the second column of Table 4.2 follows from the above inequalities. The proof for the lower bounds for the first column of Table 4.1 can be obtained by adding the bounds given by Table 4.2. This completes the proof of the lemma. □

THEOREM 4.1. *For the* R1M-4 *problem, let D be an optimal assignment such that $D' \subseteq D$ and let M be the assignment generated by our algorithm. Then, $A(M) \leqq 1.6 * A(D)$.*

Proof. Using the Lemmas 4.1 to 4.5, we know that

$$\frac{A(M)}{A(D)} \leqq \left(1 + \sum_{i=1}^{6} a_i \Big/ \sum_{i=1}^{6} b_i\right) * \left(1 + \sum_{i=1}^{6} c_i \Big/ \sum_{i=1}^{6} d_i\right),^2$$

where a_i, b_i, c_i and d_i are defined in Table 4.3 and x, y are positive integers such that $x + y = 1$.

The proof of the theorem can be obtained by multiplying all terms after replacing 1 by $m_4^A/4$ (remember that $m_4^A \geqq 4$) and obtaining an expression of the form $(ax + by + \cdots)/(a'x + b'y + \cdots)$, where $a, b, \cdots, a', b', \cdots$ are constants and $x, y, \cdots$ are products of variables. The final step consists of showing that $a/a' \leqq 1.6$, $b/b' \leqq 1.6, \cdots$.

This completes the proof of the theorem. □

TABLE 4.2

Set	Contribution to our lower bound for $h/(\lambda)$	Contribution to our lower bound for $H_D(S_1, S_2) + H_D(S_3, S_0)$
M_4^A	m_4^A	$1.5m_4^A$
M_4^N	$8m_4^N$	$6m_4^N$
$M_4^{T'} \cup M_4^{B'}$	$m_4^{T'} + m_4^{B'}$	$1.5m_4^{T'} + 1.5m_4^{B'}$
$M_4^{R'} \cup M_4^{L'}$	$1.5m_4^{R'} + 1.5m_4^{L'}$	$1.5m_4^{R'} + 1.5m_4^{L'}$
$M_4^T \cup M_4^B$	$p_4^T * m_4^T + 2m_4^T + 3m_4^B$	$3m_4^T + 3m_4^B$
M_4^R	$p_4^R * m_4^R + 3m_4^R$	$3m_4^R$

[2] Note that a_i, b_i, c_i and d_i are nonnegative. If b_i or d_i is zero, then a_i, b_i, c_i and d_i and these elements may be deleted from the summation.

TABLE 4.3(a)

i	a_i	b_i
1	$0.5m_4^A+1$	$2.5m_4^A$
2	$7x * m_4^N$	$14m_4^N$
3	$0.75m_4^{T'}+0.75m_4^{B'}$	$2.5m_4^{T'}+2.5m_4^{B'}$
4	$0.5m_4^{R'}+0.5m_4^{L'}$	$3m_4^{R'}+3m_4^{L'}$
5	$q_4^T * m_4^T+m_4^T+m_4^B$	$p_4^T * m_4^T+5m_4^T+6m_4^B$
6	$q_4^R * m_4^R+m_4^R$	$p_4^R * m_4^R+6m_4^R$

TABLE 4.3(b)

i	c_i	d_i
1	$0.5m_4^A$	$2.5m_4^A$
2	$7y * m_4^N$	$14m_4^N$
3	$0.5m_4^{T'}+0.5m_4^{B'}$	$3m_4^{T'}+3m_4^{B'}$
4	$0.75m_4^{R'}+0.75m_4^{L'}$	$2.5m_4^{R'}+2.5m_4^{L'}$
5	$p_4^T * m_4^T+m_4^T+2m_4^B$	$q_4^T * m_4^T+6m_4^T+6m_4^B$
6	$p_4^R * m_4^R+m_4^R$	$q_4^R * m_4^R+5m_4^R$

5. Approximation algorithm for the R1M-2 problem. In this section we present an approximation algorithm for the R1M-2 problem. Clearly, all global nets contain terminals located on only two opposite sides of G. Let $M_2^{TB}(M_2^{LR})$ represent the set of global nets with terminals located only on the top and bottom (left and right) sides of G. Let m_2^{TB} and m_2^{LR} represent the number of nets in M_2^{TB} and M_2^{LR}, respectively. Assume that m_2^{TB} and m_2^{LR} are multiples of 6. In § 6 we indicate the modifications that need to be made to the algorithm when this is not the case. First, let us explain (informally) how the nets in these sets are routed. We will explain only the procedure for M_2^{TB}, since the one for M_2^{LR} is similar. The set M_2^{TB} is partitioned into six equally sized groups. Later on we explain precisely how this partition is obtained. A group is said to be routed by a path type $L(R)$ if all the nets in this group are routed by a path that does not cross the right (left) side of G. We will construct 2^6 assignments. Each of these assignments corresponds to a string of six elements from the alphabet $\{L, R\}$. The ith element in a string specifies the type of path used to connect all nets in the ith group. Let us now define formally this construction process. For each net N_j in M_2^{TB} (M_2^{LR}), let $p(j)$ be the index of the leftmost (bottommost) terminal of N_j located on the top (left) side of G. Let $P^{TB}=\{p(j)\,|\,N_j \in M_2^{TB}\}$, and $P^{LR}=\{p(j)\,|\,N_j \in M_2^{LR}\}$. Each of these sets is partitioned into the sets N_i^{TB} and N_i^{LR} for $1 \leqq i \leqq 6$ as follows

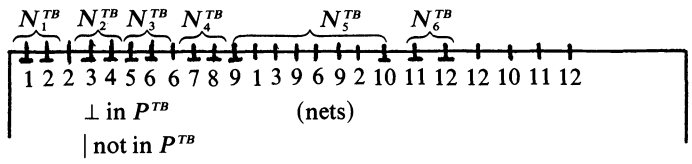


FIG. 5.0

(see Fig. 5.0). (Recall that function $I(j)$ was defined as the index of the net to which terminal T_j belongs.)

$N_i^{TB} = \{I(j) | j$ is the kth smallest value in the set P^{TB} and $((i-1)/6) * |P^{TB}| < k \leqq (i/6) * |P^{TB}|\}$ and

$N_i^{LR} = \{I(j) | j$ is the kth smallest value in the set P^{LR} and $((i-1)/6 * |P^{LR}| < k \leqq (i/6) * |P^{LR}|\}$.

We define the following sets for $0 \leqq i \leqq 63$:

$D_i^{TB} = \{$if in the binary representation of i the lth least significant bit is one then the path connecting each net in set N_l^{TB} does not cross corner S_2, otherwise the path connecting such nets does not cross corner $S_1 | 1 \leqq l \leqq 6\}$, and

$D_i^{LR} = \{$if in the binary representation of i the lth least significant bit is one then the path connecting each net in set N_l^{LR} does not cross corner S_1, otherwise the path connecting such nets does not cross corner $S_0 | 1 \leqq l \leqq 6\}$.

For $0 \leqq i, j \leqq 63$, define assignment $D_{i,j}$ as follows:

$$D_{i,j} = D' \cup D_i^{TB} \cup D_j^{LR},$$

and let P be one of the $D_{i,j}$'s of least area.

For simplicity, let us assume that in all optimal assignments the nets in $M_2^{TB} \cup M_2^{LR}$ are routed by paths that cross exactly two corners of G. In general this is not true; however our results also apply to these problem instances. Just before Lemma 5.4 we will indicate how to handle these problem instances.

Before proving our last upper bound we need to introduce additional notation. First of all let us concentrate on the set of nets in M_2^{TB}. Remember that in our algorithm we partitioned this set of nets into six groups and 2^6 assignments were obtained by routing all nets in each of these sets by paths that either do not cross the left or right side of G. Let us refer to each of these groups by G_i, for $1 \leqq i \leqq 6$ (these sets correspond to the sets N_i^{TB}, for $1 \leqq i \leqq 6$, defined above). Let D be an optimal assignment such that $D' \subseteq D$. Assignment D will be transformed into a nonemtpy set of assignments by an i-string. An i-string consists of a sequence of six elements from the alphabet {"r", "l", "|", "*", "."}. If the number of "*" symbols in the i-string is j, 2^j assignments are generated. The kth element in an i-string indicates how the paths connecting the nets G_k in D are changed. Let us now explain how these changes are performed. When the kth element in an i-string is "l", the paths connecting all the nets in G_k that cross the left side of G in assignment D are changed. These nets will be connected by paths that do not cross the left side of G. If the kth element is "r", the paths connecting all the nets in G_k that cross the right side of G in assignment D are changed. These nets will be connected by paths that do not cross the right side of G. When the kth symbol is "|", it means "l" if the number of paths connecting the nets in G_k that cross the left side of G in D is less than or equal to $(1/2) * |G_k|$; and it means "r" otherwise. When the element is a period "." it means that none of the paths connecting the nets in G_k are changed. The interpretation of "*" is more complex. An i-string with one or more "*" symbols denotes the set of all i-strings *obtainable* from it by replacing each of the "*" symbols with an "l" or an "r." When we apply an i-string with no "*" symbols to an optimal assignment, D, we obtain an assignment identical to D except for the paths connecting some of the nets in the groups G_i. The specific paths

to be changed are indicated by the i-string. Applying an i-string with "*" symbols, I, to an assignment, D, generates a (nonempty) set of assignments. This set of assignments consists of all assignments obtained from D by applying an i-string obtainable from I.

In Lemma 5.1 and the lemmas appearing in the appendices, we assume that when interchanging the paths connecting two nets in M_2^{TB} that cross on the top side of G will increase by at most two the vertical height of the assignment. The assumption is not always true. In Fig. 5.1 we show the only counterexample. We call this interchange a *type I interchange.* The two nets involved in this interchange form a *type I pair.* Interchanging the path connecting a net in M_2^{TB} to one that does not cross the left or right sides of G increases the vertical height of an assignment by at most two. This statement always holds.

Sometimes Lemma 5.1 holds even when type I interchanges occur when transforming an optimal assignment to the ones generated in our algorithm. Since there are cases when this will not be true we need to apply some postprocessing improvement to eliminate the effects of the type I interchanges. We will explain how to do this after Lemma 5.1. We should point out that after this postprocessing procedure the 1.6 approximation bound will hold for all problem instances.

LEMMA 5.1. *Let D be an optimal assignment such that $D' \subseteq D$. There is an assignment $D_{i,j}$ (constructed by our algorithm) such that if M is defined as D except for all the nets in $M_2^{TB} \cup M_2^{LR}$ which are routed as in the assignment $D_{i,j}$, then*

(a) $H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (0.6) * m_2^{TB}$,

(b) $H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + (0.6) * m_2^{LR}$.

(It is assumed that when D is transformed to any of the assignments D_{ij} and two paths that cross on the top side of G are interchanged, such an interchange is not a type I interchange.)

Proof. The proof is given in Appendix A. □

Let L be an optimal solution that includes D'. The existence of at least one of these assignments is guaranteed by Lemma 2.3. Let S be any of the $D_{i,j}$ assignments. The difference between S and L is the way in which some nets in $M_2^{TB} \cup M_2^{LR}$ are routed. Lemma 5.1 shows that at least one of our assignments differs in vertical height from L by at most $0.6 * m_2^{TB}$ and in honrizontal height by at most $0.6 * m_2^{LR}$. As noted in the text appearing immediately before Lemma 5.1, we cannot yet claim that the 0.6 bound holds when type I interchanges occur.

When there is a type I interchange, the height on the top side of G could increase by at most one and the height of the bottom side of G could increase by at most two. If instead of increasing the total vertical height by at most three it would have been by at most two, then the previous lemma would be correct even with interchanges of type I. This is true sometimes, but in general we cannot guarantee that it will always happen. In order for our 0.6 bound to hold we transform each assignment S (each of the $D_{i,j}$'s) to another assignment R such that if the total increase in vertical height when transforming L to S by interchanging some of the nets in M_2^{LR} was $\leqq w+3\alpha$,

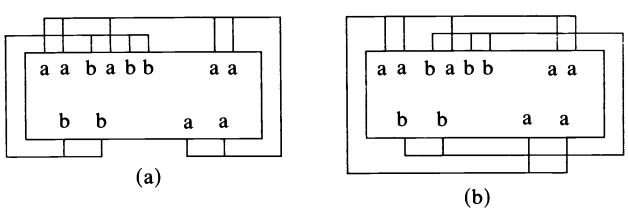


FIG. 5.1. *Type I interchange.* (a) *Before the interchange*; (b) *After the interchange.*

where α is the number of type I interchanges made and w is the contribution from all other interchanges, then the actual difference in vertical height between L and R is at most $w+2\alpha$. Before presenting our transformation algorithm we make a couple of definitions.

DEFINITION 5.1. A pair of nets, (a, b), in M_2^{TB} is said to form a *crossing pair* in an assignment if the following conditions are satisfied (see Fig. 5.2);

(i) Net "a" is connected by a path that crosses the left side of G.

(ii) Net "b" is connected by a path that crosses the right side of G.

(iii) On the bottom side of G all the terminals from net "b" appear to the left of all the terminals from net "a".

(iv) On the top side of G there is at least one terminal from net "a" located to the right of the rightmost terminal from net "b" and at least one terminal from net "a" is located to the left of the leftmost terminal from net "b".

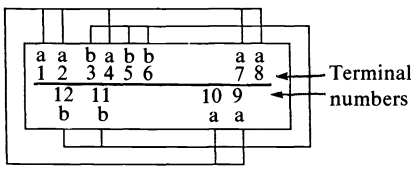


FIG. 5.2. *Crossing pair* (a, b).

One can easily prove that any two nets involved in a type I interchange will, after the interchange, form a crossing pair. However, the converse is not true. Later on we show that such a pair has harming effects only when it is a conflicting pair.

DEFINITION 5.2. A crossing pair (a, b) *includes* point x if either of the two following conditions is satisfied:

(i) If x is located on the top side of G, then all terminals from net "b" located on the top side of G appear to the left of x and the rightmost terminal from net "a" is located to the right of x (interval (6, 8) in Fig. 5.2). Or

(ii) If x is located on the bottom side of G, then all terminals from net "b" located on the bottom side of G appear to the left of x and all the terminals from net "a" are located to the right of x (interval (10, 11) in Fig. 5.2).

DEFINITION 5.3. A crossing pair (a, b) *partially includes* point x if x is located on the bottom side of G and there is a terminal from net b at point x or to the left of x, and there is a terminal from net a at point x or to the right of x (interval [9, 12] in Fig. 5.2).

Note that if point x located on the bottom side of G is included in a crossing pair then it is also partially included in it, but the converse is not always true.

DEFINITION 5.4. A *conflicting pair*, (a, b), is a crossing pair that includes the leftmost point with maximum height located on the top side of G, partially includes all the points with maximum height located on the bottom side of G and includes either the leftmost or the rightmost point with maximum height located on the bottom side of G.

Our postprocessing procedure will find conflicting pairs and interchange their connecting paths. In Fig. 5.3 we show a conflicting pair and in Fig. 5.4 we indicate how these paths are interchanged.

This transformation reverses the effects of type I interchanges.

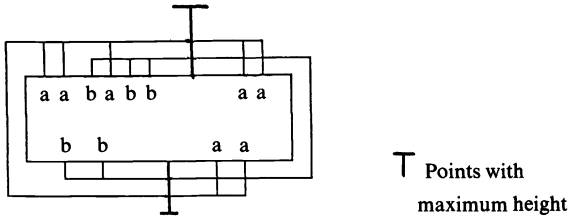


FIG. 5.3. *(a, b) forms a conflicting pair.*

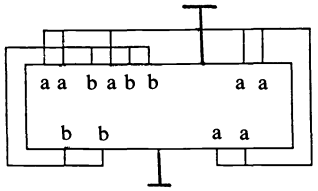


FIG. 5.4 *Interchange of the conflicting pair given in Fig. 5.3.*

ALGORITHM MODIFY
$R \leftarrow S$;
//The statement in this comment is not executed by the algorithm; its presence is to simplify the proof of Lemma 5.2.
Mark all nets involved in type I interchanges when L was transformed into S.//
while there is a conflicting pair in R *do*
interchange a conflicting pair in R
//The statement in this comment is not executed by the algorithm; its presence is to simplify the proof of Lemma 5.2.
Unmark all marked nets involved in the interchange just performed as well as all nets involved in a type I interchange with the nets in the conflicting pair just interchanged.//
endwhile
end of algorithm

During each iteration of the algorithm the maximum height on the top side of G is not increased; however the maximum height on the bottom side of G is decreased by at least one. Note that the maximum height on the top side of G is not increased because a conflicting pair includes the leftmost point with maximum height on the top side of G.

Let α be the number of type I interchanges made when transforming L to S and let $w+3\alpha$ be the total increase in vertical height because of such transformation. Let $R_0 = S$ and $\alpha_0 = \alpha$. For $i \geqq 1$ let R_i be assignment R at the end of the ith iteration of algorithm MODIFY and let α_i be the number of type I pairs that remain marked at the end of the ith iteration of algorithm MODIFY (using the "imaginary" computations described inside the comments).

LEMMA 5.2. *The vertical height of assignment R_i minus the vertical height of assignment L is at most $w+2\alpha+\alpha_i$.*

Proof. We prove this lemma by induction on i. By definition, the statement of the lemma is true for $i=0$. Assume it is true for $i-1 \geqq 0$ and let us now show it is true for i. There are three cases depending on the number of nets unmarked during the ith iteration of the algorithm.

Case 1. No nets are unmarked during the ith iteration.

Since at each iteration of the algorithm we decrease the vertical height by at least one and since no terminal is unmarked at this step,

$$\begin{aligned}(H_{R_i}(S_1, S_2)+H_{R_i}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))&\\ <(H_{R_{i-1}}(S_1, S_2)+H_{R_{i-1}}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))&\\ \leqq w+2\alpha+\alpha_{i-1}&\\ =w+2\alpha+\alpha_i&\end{aligned}$$

and the induction hypothesis holds for i.

Case 2. A type I pair is unmarked during the ith iteration.

Since at each iteration of the algorithm we decrease the vertical height by at least one and since $\alpha_i=\alpha_{i-1}-1$,

$$\begin{aligned}(H_{R_i}(S_1, S_2)+H_{R_i}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))&\\ \leqq(H_{R_{i-1}}(S_1, S_2)+H_{R_{i-1}}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))-1&\\ =w+2\alpha+\alpha_{i-1}-1&\\ =w+2\alpha+\alpha_i,&\end{aligned}$$

and the induction hypothesis holds for i.

Case 3. Two type I pairs are unmarked during the ith iteration.

Each of the nets in the two pairs contributed two to the 2α term in $(H_{R_{i-1}}(S_1, S_2)+H_{R_{i-1}}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))$ and one to the α_{i-1} term in the same expression. Now, we know that two of these nets will be routed as in assignment L, so the total increase of the vertical height by each of the two remaining nets is at most two. Hence the total contribution of the interchange is at most four.

$$\begin{aligned}(H_{R_i}(S_1, S_2)+H_{R_i}(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))&\\ \leqq\text{contribution of the nets not involved in this interchange}&\\ \quad\text{(which is the same as in the previous iteration)}+&\\ \quad\text{contribution of the two pairs involved in the interchange}&\\ \leqq(w+2(\alpha-2)+(\alpha_{i-1}-2))+4&\\ =w+2\alpha+\alpha_{i-1}-2.&\end{aligned}$$

Since $\alpha_i=\alpha_{i-1}-2$, the induction hypothesis holds for i and the proof of the lemma follows by induction. □

DEFINITION 5.5. When algorithm MODIFY terminates, let x be the leftmost point located on the top side of G whose height in assignment R is maximum.

LEMMA 5.3. *The vertical height of assignment R minus the vertical height of assignment L is at most* $w+2\alpha$.

Proof. It is simple to see that

$$(H_R(S_1, S_2)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))\leqq w+2\alpha+\alpha'$$

where α' is the value of α_i at the end of the last iteration of the algorithm. If α' is zero, there is nothing to prove. So let us assume it is not the case. Let k be the number of type I pairs that do not include point x. Each of these k pairs will only contribute

two to

$$(H_R(x, x)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))$$
$$=(H_R(S_1, S_2)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0)).$$

Hence, if $k=\alpha'$, there is nothing to prove. So, assume that $k<\alpha'$ and let $\alpha''=\alpha'-k$. That is α'' is the number of type I pairs that include point x. Hence,

$$(H_R(S_1, S_2)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))\leqq w+2\alpha+\alpha''.$$

Let y_l (y_r) be the leftmost (rightmost) point on the bottom side of G whose height in assignment R is maximum. Since the algorithm terminates when there remains no conflicting pairs, there can be no type I pair that includes x, y_l and partially includes (or includes) y_r; or includes x, y_r and partially includes (or includes) y_l. Let z_l (z_r) be the number of type I pairs that include x and y_l (y_r). There are two cases depending on the values of z_l and z_r.

Case (i): $z_l\leqq z_r$. It is simple to verify that each type I pair is counted (included) in either z_l, z_r or in $\alpha-z_r-z_l$. Every type I pair counted in z_l contributes to

$$(H_R(x, x)+H_R(y_l, y_l))-(H_L(S_1, S_2)+H_L(S_3, S_0))$$

exactly three, each of the type I pairs counted in z_r contributes one, and each of the remaining type I pairs (those not counted in z_r and z_l) that include x contributes at most two. Hence,

$$(H_R(x, x)+H_R(y_l, y_l))-(H_L(S_1, S_2)+H_L(S_3, S_0))\leqq w+2(\alpha-z_l-z_r)+3*z_l+z_r.$$

Since $z_l\leqq z_r$, $z_l+z_r\leqq\alpha''$ and

$$(H_R(x, x)+H_R(y_l, y_l))-(H_L(S_1, S_2)+H_L(S_3, S_0))$$
$$=(H_R(S_1, S_2)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0)).$$

Hence, $(H_R(S_1, S_2)+H_R(S_3, S_0))-(H_L(S_1, S_2)+H_L(S_3, S_0))\leqq w+2\alpha$. This completes the proof for case (i).

Case (ii): $z_l>z_r$. The proof for this case is similar to case (i), so it will be omitted. This completes the proof of the lemma. □

An algorithm similar to MODIFY has to be applied to the $D_{i,j}$ in order to be able to claim the bound $0.6*m_2^{LR}$. Let us refer to this new procedure MODIFY-LR. Our algorithm works as follows:

Obtain assignments $D_{i,j}$;
Use MODIFY and MODIFY-LR to transform each $D_{i,j}$ into a new assignment, $M_{i,j}$;
Output the assignment $M_{i,j}$ of least area;

We have also been assuming that in all optimal assignments the nets in $M_2^{TB}\cup M_2^{LR}$ are connected by paths that cross exactly two corners of G. In general this is not true. Let D be an optimal assignment and let $m_2^{TB'}$ ($m_2^{LR'}$) be the number of nets in M_2^{TB} (M_2^{LR}) that are connected in D by paths that cross two corners of G. Let $m_2^{TB''}=m_2^{TB}-m_2^{TB'}$ and $m_2^{LR''}=m_2^{LR}-m_2^{LR'}$. When transforming an optimal assignment D to one of the assignments generated by our algorithm, we will first modify the assignment to one in which all nets in $M_2^{TB}\cup M_2^{LR}$ are connected by paths that cross exactly two of the corners of G. This new assignment is obtained by connecting all nets in M_2^{TB} (M_2^{LR}) that are connected by paths that cross the four corners of G, by paths that do not cross the right (top) side of G. The new assignment satisfies the restrictions imposed

before. The cost of this modification must be accounted for (see Lemma 5.4). Note that each time an interchange is performed we will increase by at most one the height of one side of G, but we will decrease the height of another side (adjacent to the previous one) of G by one. From the above discussion and lemmas, we obtain the following lemma which we state without a proof.

LEMMA 5.4. *Let D be an optimal assignment such that $D' \subseteq D$. There is an assignment $M_{i,j}$ (constructed by our algorithm) such that if M is defined as D except for all the nets in $M_2^{TB} \cup M_2^{LR}$ which are routed as in the assignment $M_{i,j}$, then*

$$\text{(a)}\quad H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (0.6) * m_2^{TB} + m_2^{TB''} - m_2^{LR''},$$

$$\text{(b)}\quad H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + (0.6) * m_2^{LR} + m_2^{LR''} - m_2^{TB''}.$$

Before proving our main result in this section we establish a lower bound on the area required by any optimal solution. The lower bound is given in Table 5.1.

LEMMA 5.5. *Let D be an optimal assignment such that $D' \subseteq D$. Assignment D and rectangle G satisfy the lower bounds given in Table* 5.1.

Proof. We only prove the lower bounds in the first column of Table 5.1 since the proof for the bounds in the second column is similar. Clearly, each net in M_2^{LR} has at least one terminal on the left and right side of G.

Using the above observation, we obtain the bounds given by the first column of Table 5.2.

Let us now establish a lower bound on the number of paths crossing the corners of G. Since every path for a net in $M_2^{TB'} \cup M_2^{LR'}$ crosses two corners of G, and every path for a net in $M_2^{TB''} \cup M_2^{LR''}$ crosses four corners of G, we know that

$$\sum_{i=0}^{3} H_D(S_i, S_i) \geqq 2(m_2^{TB'} + m_2^{LR'}) + 4(m_2^{TB''} + m_2^{LR''}).$$

It is simple to show that

$$H_D(S_1, S_2) + H_D(S_3, S_0) \geqq 0.5 * (H_D(S_1, S_1) + H_D(S_2, S_2)) + 0.5 * (H_D(S_3, S_3) + H_D(S_0, S_0)).$$

TABLE 5.1
Lower bounds for the R1M-2 *problem.*

Set	Contribution to our lower bound for $h/(\lambda) + H_D(S_1, S_2) + H_D(S_3, S_0)$	Contribution to our lower bound for $w/(\lambda) + H_D(S_0, S_1) + H_D(S_2, S_3)$
M_2^{TB}	$m_2^{TB'} + 2m_2^{TB''}$	$2m_2^{TB'} + 3m_2^{TB''}$
M_2^{LR}	$2m_2^{LR'} + 3m_2^{LR''}$	$m_2^{LR'} + 2m_2^{LR''}$

TABLE 5.2

Set	Contribution to our lower bound for $h/(\lambda)$	Contribution to our lower bound for $H_D(S_1, S_2) + H_D(S_3, S_0)$
M_2^{TB}	—	$m_2^{TB'} + 2m_2^{TB''}$
M_2^{LR}	$m_2^{LR'} + m_2^{LR''}$	$m_2^{LR'} + 2m_2^{LR''}$

The lower bound for the second column of Table 5.2 follows from the above inequalities. The proof for the lower bounds for the first column of Table 5.1 can be obtained by adding the bounds given by Table 5.2. This completes the proof of the lemma. □

THEOREM 5.1. *Let D be an optimal assignment such that $D' \subseteq D$ and let P be the assignment produced by our algorithm. Then, $A(P) \leqq 1.6 * A(D)$.*

Proof. Let M be any of the assignments constructed by the algorithm such that $M = M_{i,j}$ and $M_{i,j}$ is an assignment that satisfies Lemma 5.4. We now prove that $A(M) \leqq 1.6 * A(D)$. Since $A(P) \leqq A(M)$ our bound holds. Using Lemmas 5.4 and 5.5 we know that:

$$A(M)/A(D) \leqq (1 + a_1/b_1) * (1 + c_1/d_1),^3$$

where

$$a_1 = 0.6m_2^{TB'} + 1.6m_2^{TB''} - m_2^{LR''},$$
$$b_1 = m_2^{TB'} + 2m_2^{TB''} + 2m_2^{LR'} + 3m_2^{LR''},$$
$$c_1 = -m_2^{TB''} + 0.6m_2^{LR'} + 1.6m_2^{LR''},$$
$$d_1 = 2m_2^{TB'} + 3m_2^{TB''} + m_2^{LR'} + 2m_2^{LR''}.$$

The proof of the theorem can be obtained by multiplying all terms and obtaining an expression of the form $(ax + by + \cdots)/(a'x + b'y + \cdots)$, where $a, b, \cdots, a', b', \cdots$ are constants and $x, y, \cdots$ are products of variables. The final step consists of showing that $a/a' \leqq 1.6$, $b/b' \leqq 1.6, \cdots$.

This completes the proof of the theorem. □

6. Approximation algorithm for the R1M problem. In this section we show that our algorithm takes $O(n(m + \log n))$ time and generates a solution with area $\leqq 1.6 * \text{OPT}$, where OPT is the area of an optimal solution.

ALGORITHM FOR THE R1M PROBLEM

Construct assignments D', D_3 and D_4;
$D_{i,j} \leftarrow D' \cup D_3 \cup D_4 \cup D_i^{TB} \cup D_j^{LR}$ for $0 \leqq i, j \leqq 63$;
Apply algorithm MODIFY and MODIFY-LR to each $D_{i,j}$ to obtain $M_{i,j}$;
Let P be one of the $M_{i,j}$'s of least area;
Construct and output a layout with area $A(P)$ for P;
end of algorithm

THEOREM 6.1. *The time complexity of our algorithm is $O(n(m + \log n))$.*

Proof. It is simple to verify that all the steps in our algorithm take $O(n)$ time except for the initial sorting of the terminals (if not initially sorted), the construction of assignment D''_A, D''_X ($X \in \{T', R', B', L'\}$), and procedure MODIFY and MODIFY-LR. Clearly, sorting can be performed in $O(n \log n)$ time. Assignment D''_A (as well as assignments D''_X, $X \in \{T', R', B', L'\}$) can be obtained in $O(n \log n)$ time by a simple recursive procedure that manipulates four priority queues and uses a simple marking scheme. The priority queues can be implemented by heap trees [AHU] and the marking scheme can be implemented using a one-dimensional array. A slightly complex marking scheme can be used to reduce the time complexity of this part of the algorithm to $O(n)$. For brevity such a scheme will be omitted. The total number of iterations performed by procedure MODIFY and MODIFY-LR is at most $O(m)$ since at each

[3] See footnote in Theorem 4.1.

iteration the height of assignment R on the bottom side of G is decreased by one and the height can be at most $O(m)$. The statements inside the "while loop" of both procedures can be easily implemented to take $O(n)$ time. Hence, the total time complexity for both these procedures is $O(n(m+\log n))$. □

Before proving our main result we establish a lower bound on the area required by any optimal solution. The lower bound is given in Table 6.1.

TABLE 6.1
Lower bounds for the R1M *problem.*

Set	Contribution to our lower bound for $h/(\lambda)+H_D(S_1,S_2)+H_D(S_3,S_0)$	Contribution to our lower bound for $w/(\lambda)+H_D(S_0,S_1)+H_D(S_2,S_3)$
M_2^{TB}	$m_2^{TB'}+2m_2^{TB''}$	$2m_2^{TB'}+3m_2^{TB''}$
M_2^{LR}	$2m_2^{LR'}+3m_2^{LR''}$	$m_2^{LR'}+2m_2^{LR''}$
M_3^{TB}	$1.5m_3^{TB}+0.5x_3^{TB}$	$2m_3^{TB}+0.5x_3^{TB}$
M_3^{LR}	$2m_3^{LR}+0.5x_3^{LR}$	$1.5m_3^{LR}+0.5x_3^{LR}$
M_4^{A}	$2.5m_4^{A}$	$2.5m_4^{A}$
M_4^{N}	$14m_4^{N}$	$14m_4^{N}$
$M_4^{T'}\cup M_4^{B'}$	$2.5m_4^{T'}+2.5m_4^{B'}$	$3m_4^{T'}+3m_4^{B'}$
$M_4^{R'}\cup M_4^{L'}$	$3m_4^{R'}+3m_4^{L'}$	$2.5m_4^{R'}+2.5m_4^{L'}$
$M_4^{T}\cup M_4^{B}$	$p_4^{T}*m_4^{T}+5m_4^{T}+6m_4^{B}$	$q_4^{T}*m_4^{T}+6m_4^{T}+6m_4^{B}$
M_4^{R}	$p_4^{R}*m_4^{R}+6m_4^{R}$	$q_4^{R}*m_4^{R}+5m_4^{R}$

LEMMA 6.1. *Let D be an optimal assignment such that $D'\subseteq D$. Then assignment D and rectangle G satisfy the lower bounds given in Table* 6.1.

Proof. The proof for this bound follows from Lemmas 3.2, 4.5 and 5.5. □

THEOREM 6.2. *Let D be an optimal assignment such that $D'\subseteq D$ and let P be the assignment generated by our algorithm. Then, $A(P)\leqq 1.6*A(D)$.*

Proof. Let M be any of the assignments constructed by the algorithm such that $M=M_{i,j}$ and $M_{i,j}$ is an assignment that satisfies Lemma 5.4. We now prove that $A(M)\leqq 1.6*A(D)$. Since $A(P)\leqq A(M)$ our bound holds. Using arguments similar to those used in the proofs of Lemmas 3.1, 4.1, 4.2, 4.3, 4.4, and 5.4, and the lower bound given in Lemma 6.1, we know that:

$$\frac{A(M)}{A(D)}\leqq\left(1+\sum_{i=1}^{8}a_i\Big/\sum_{i=1}^{8}b_i\right)*\left(1+\sum_{i=1}^{8}c_i\Big/\sum_{i=1}^{8}d_i\right),^{4}$$

where a_i, b_i, c_i and d_i are defined in Table 6.2 and x, y are positive integers such that $x+y=1$.

Substituting $m_3^{TB}\geqq x_3^{TB}$, $m_3^{LR}\geqq x_3^{LR}$ and $a\geqq m_4^{A}/4$ (remember that $m_4^{A}\geqq 4$) and multiplying all terms, we obtain the expression of the form $(ax+by+\cdots)/(a'x+b'y+\cdots)$, where $a, b, \cdots, a', b', \cdots$ are constants and $x, y, \cdots$ are products of variables. The final step consists of showing that $a/a'\leqq 1.6$, $b/b'\leqq 1.6, \cdots$. This completes the proof of the theorem. □

Our algorithm generates 2^{12} assignments and it outputs one that requires the least layout area. An algorithm that only generates 2^6 assignments can be easily obtained by only taking the best of the modified $D'\cup D_3\cup D_4\cup D_i^{TB}$ together with the best of the modified $D'\cup D_3\cup D_4\cup D_i^{LR}$. Note that a transformation applied to the first assignment does not modify the horizontal height on the second assignment, and a

[4] The footnote in Theorem 4.1 applies for $i=1, 8$.

TABLE 6.2(a)

i	a_i	b_i
1	$0.6m_2^{TB'}+1.6m_2^{TB''}-m_2^{LR''}$	$m_2^{TB'}+2m_2^{TB''}+2m_2^{LR'}+3m_2^{LR''}$
2	x_3^{TB}	$1.5m_3^{TB}+0.5x_3^{TB}+2m_3^{LR}+0.5x_3^{LR}$
3	$0.5m_4^A+1$	$2.5m_4^A$
4	$7x * m_4^N$	$14m_4^N$
5	$0.75m_4^{T'}+0.75m_4^{B'}$	$2.5m_4^{T'}+2.5m_4^{B'}$
6	$0.5m_4^{R'}+0.5m_4^{L'}$	$3m_4^{R'}+3m_4^{L'}$
7	$q_4^T * m_4^T+m_4^T+m_4^B$	$p_4^T * m_4^T+5m_4^T+6m_4^B$
8	$m_4^R+q_4^R * m_4^R$	$p_4^R * m_4^R+6m_4^R$

TABLE 6.2(b)

i	c_i	d_i
1	$-m_2^{TB''}+0.6m_2^{LR'}+1.6m_2^{LR''}$	$2m_2^{TB'}+3m_2^{TB''}+m_2^{LR'}+2m_2^{LR''}$
2	x_3^{LR}	$2m_3^{TB}+0.5x_3^{TB}+1.5m_3^{LR}+0.5x_3^{LR}$
3	$0.5m_4^A$	$2.5m_4^A$
4	$7y * m_4^N$	$14m_4^N$
5	$0.5m_4^{T'}+0.5m_4^{B'}$	$3m_4^{T'}+3m_4^{B'}$
6	$0.75m_4^{R'}+0.75m_4^{L'}$	$2.5m_4^{R'}+2.5m_4^{L'}$
7	$p_4^T * m_4^T+m_4^T+2m_4^B$	$q_4^T * m_4^T+6m_4^T+6m_4^B$
8	$p_4^R * m_4^R+m_4^R$	$q_4^R * m_4^R+5m_4^R$

transformation applied to the second assignment does not modify the vertical height on the first assignment. For brevity we will not prove that this solution also satisfies our approximation bounds. In §§ 4 and 5 we assumed that the number of nets in some sets was a multiple of some fixed constant. More specifically, we might have to delete at most five nets from sets M_2^{TB} and M_2^{LR}; at most three nets from sets M_4^A, $M_4^{T'}$, $M_4^{R'}$, $M_4^{B'}$ and $M_4^{L'}$; and procedure PARTITION deletes at most eight nets (those never added to M_4^N, M_4^T, M_4^R and M_4^B). All of these nets with a "small" number of terminals can be routed optimally by trying all possible routing paths and then selecting the best of the solutions generated. If some of these nets do not have a "small" number of terminals then select any routing paths for them. Note that it will not make too much difference which routing path is selected since their contribution to the lower bound in an optimal solution is large (contribution from the number of terminals). There are better ways of dealing with the remaining 33 nets; however, for brevity these other methods will not be discussed in this paper.

7. Discussion We have shown that there is an efficient approximation algorithm that generates a solution within 60% of optimal for the R1M problem. The algorithm takes $O(n(m+\log n))$ time and the constant associated with this bound is small. Our algorithm generates 2^6 assignments (see comment at the end of § 6) and it outputs one with least layout area. We conjecture that the solutions generated by our algorithm are usually closer to optimal than the bound of 60%. The reason for this is that our bounds are rough and are based on analyzing groups of nets separately. We took this approach in order to simplify the analysis, which in spite of this became very complex. One of the interesting aspects of our algorithms is that the proof of Lemma 5.1 can be obtained by solving a set of Linear Programming problems. Before we proved this lemma we programmed a personal computer (IBM with INTEL 8088 and 8087 processors) to prove the lemma for us. The computer solved several hundred linear

programming problems using the simplex method. In each of these problems there were around eighty equations and about one hundred variables. The simplex method that we programmed took about 30 iterations to generate the optimal solution to each of the LP problems. The total execution time was about 50 CPU hours.

A better solution (not a better approximation bound) can be obtained by selecting more groups for the nets in set M_2^{TB} and M_2^{LR}, as well as some postprocessing improvements similar to those performed by procedure MODIFY. For brevity we will not discuss such extensions. If less than six groups are selected for M_2^{TB} or M_2^{LR}, one cannot prove the approximation bound 1.6. When three groups are selected (as in [GL3]), the approximation bound is exactly 1.69.

Appendix A.

LEMMA A.1. *Let D be an optimal assignment such that $D' \subseteq D$. There is an assignment $D_{i,j}$ (constructed by our algorithm) such that if M is defined as D except for all the nets in $M_2^{TB} \cup M_2^{LR}$ which are routed as in the assignment $D_{i,j}$, then*

(a) $H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (0.6) * m_2^{TB}$,

(b) $H_M(S_0, S_1) + H_M(S_2, S_3) \leqq H_D(S_0, S_1) + H_D(S_2, S_3) + (0.6) * m_2^{LR}$.

It is assumed that when D is transformed to any of the assignments D_{ij} and two paths that cross on the top side of G are interchanged, such an interchange is not a type I interchange.

Proof. Since the proof for part (b) is similar to the proof of part (a), we only prove part (a). Before proving the lemma we make the following observations (note that we are assuming that there are no type I interchanges):

(i) The interchange of a path connecting a net in M_2^{TB} in assignment D to one that crosses the least number of corners in G will increase by at most two the vertical height of D.

(ii) The interchange of two paths that cross on the top side of G in assignment D will increase by at most two the vertical height of D.

Let G_i, for $1 \leqq i \leqq 6$, be the partition of the nets generated by our algorithm (these sets were called N_i^{TB}, for $1 \leqq i \leqq 6$, in our algorithm). For assignment D, let l_i be the number of nets in G_i that are connected by a path that crosses the left side of G and let r_i be the number of nets in G_i that are connected by a path that crosses the right side of G.

In what follows we apply several i-strings to assignment D. When an i-string contains three "*" symbols and three "." symbols, for example ".*..**", eight assignments are obtained. In Appendix B we prove that at least one of these assignments, M, has the property that

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (1/3) * m_2^{TB}.$$

When the i-string contains five "*" symbols and a "." symbol, for example "**.***", 32 assignments are generated. In Appendix B we show that at least one of these assignments, M, has the property that

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (8/15) * m_2^{TB}.$$

In what follows we will use i-strings that contain three "*" symbols and three letters, for example "*r*rl**." Using the lemma in Appendix B we know that the i-string ".*..**" when applied to D produces an assignment with some given property. If such assignment is modified by the i-string "*r.rl..*" we obtain an assignment whose vertical height differs from the one in D by at most $2 \max\{r_1 + r_3, l_4\} + (1/3) * m_2^{TB}$. A similar interpretation is given to i-strings with five "*" symbols and one "|" symbol. In this

case the bound on the increase of the vertical height is given by the lemma in Appendix C.

We prove part (a) of the lemma by contradiction. Suppose that all assignments M obtained from D by applying the i-string "******" do not satisfy the lemma. There are three cases that need to be considered.

Case 1: $\max\{l_i, r_i\} \geqq (2/15) * m_2^{TB}$, for some $1 \leqq i \leqq 6$. Let j be such that $\max\{l_j, r_j\} \geqq (2/15) * m_2^{TB}$. Since $r_j + l_j = m_2^{TB}/6$, we know that $\min\{l_j, r_j\} \leqq m_2^{TB}/30$. Applying the i-string that consists of all "*" symbols except for a "|" symbol in the jth position to D generates an assignment whose vertical height differs from the one in D by at most

$$2 \min\{l_j, r_j\} + (8/15) * m_2^{TB},$$

which we know must be greater than $(0.6) * m_2^{TB}$. Substituting the upper bound for "min" obtained above we know that $(9/15) > 0.6$, a contradiction. Hence, the conditions of Case 1 do not hold.

Case 2.

(A.2.0) $\max\{l_i, r_i\} < (2/15) * m_2^{TB}$ for $1 \leqq i \leqq 6$, and

(A.2.1) $r_2 + r_4 \geqq l_3 + l_5$.

First of all we prove a pair of inequalities (A.2.2 and A.2.3) hold. These inequalities will be useful later on. Applying the i-string "*rrl****" to D generates an assignment whose vertical height differs from the one in D by at most $2 \max\{r_1 + r_2, l_3\} + m_2^{TB}/3$, which we know must be greater than $(0.6) * m_2^{TB}$. This is equivalent to

$$\max\{r_1 + r_2, l_3\} > (2/15) * m_2^{TB}.$$

If $l_3 \geqq r_1 + r_2$ then from the above inequality we know that $l_3 > (2/15) * m_2^{TB}$. This contradicts (A.2.0). So it must be that

(A.2.2) $r_1 + r_2 > l_3$.

Applying the i-string "***rll*" to D and using similar arguments, we know that

(A.2.3) $l_5 + l_6 > r_4$.

Hence, it must be that inequalities (A.2.2) and (A.2.3) hold as otherwise there is a contradiction.

We now show that for this case there is a contradiction. There are two subcases.

Subcase 2.1.

(A.2.1.1) $l_4 + l_5 + l_6 \geqq r_1 + r_2 + r_3$.

CLAIM. *If one of the following inequalities does not hold then there is a contradiction*:

(A.2.1.2) $l_3 + l_4 + l_6 \geqq r_1 + r_2 + r_5$,

(A.2.1.3) $l_2 + l_4 + l_6 \geqq r_1 + r_3 + r_5$,

(A.2.1.4) $r_5 < l_6$,

(A.2.1.5) $r_4 < l_5$,

(A.2.1.6) $r_3 < l_4$,

(A.2.1.7) $r_2 > l_3$.

Proof of Claim. In what follows we show that (A.2.1.2)-(A.2.1.7) hold as otherwise there is a contradiction.

Proof of (A.2.1.2)-(A.2.1.3). If any of these inequalities does not hold then replace in it r_j by $m_2^{TB}/6 - l_j$ and l_k by $m_2^{TB}/6 - r_k$ for appropriate values of k and j, and we obtain a contradiction to inequality (A.2.1.1).

Proof for (A.2.1.4). Since inequality (A.2.1.1) holds we know that applying the i-string "*rrrlll*" to D produces an assignment with a vertical height that differs from the one in D by at most

$$2(l_4 + l_5 + l_6),$$

which we know must be greater than $(0.6) * m_2^{TB}$. If $r_5 \geqq l_6$ then replacing l_6 by r_5 and replacing $r_5 + l_5$ by $m_2^{TB}/6$ in the above inequality we know that $l_4 > (2/15) * m_2^{TB}$. This contradicts inequality (A.2.0). Hence, it must be that $r_5 < l_6$. This completes the proof for (A.2.1.4).

Proof for (A.2.1.5). Since the proof for this part is similar to one for the previous one, it will be omitted.

Proof for (A.2.1.6). Same as the proof for (A.2.1.4) but applying the i-string "*rrllrl*" and using inequalities (A.2.1.4) and (A.2.1.2).

Proof for (A.2.1.7). The proof follows from (A.2.1.5) and (A.2.1).

End of Claim. □

We now show that when inequalities (A.2.1.2)-(A.2.1.7) and (A.2.0) hold then there is a contradiction. Applying the i-string "*rlrlrl*" to D and using inequalities (A.2.1.4), (A.2.1.6) and (A.2.1.3), we know that

$$l_2 + l_4 + l_6 > (0.3) * m_2^{TB}.$$

Similarly, applying the i-string "$|$*rlrlr*" and using inequalities (A.2.1.7) and (A.2.1), we know that

$$\min\{r_1, l_1\} + r_2 + r_4 + r_6 > (0.3) * m_2^{TB}.$$

Adding these two inequalities and replcing $r_j + l_j$ by $(1/6) * m_2^{TB}$, we know that

$$\min\{r_1, l_1\} > (0.1) * m_2^{TB},$$

a contradiction (since $r_1 + l_1 = (1/6) * m_2^{TB}$). This completes the proof for Subcase 2.1.

Subcase 2.2.

(A.2.2.1) $$l_4 + l_5 + l_6 \leqq r_1 + r_2 + r_3.$$

CLAIM. *If one of the following inequalities does not hold then there is a contradiction.*

(A.2.2.2) $$r_1 + r_2 + r_4 \geqq l_3 + l_5 + l_6,$$

(A.2.2.3) $$r_1 + r_3 + r_4 \geqq l_2 + l_5 + l_6,$$

(A.2.2.4) $$r_1 + r_3 + r_5 \geqq l_2 + l_4 + l_6,$$

(A.2.2.5) $$r_1 + r_2 + r_5 \geqq l_3 + l_4 + l_6,$$

(A.2.2.6) $$r_1 > l_2,$$

(A.2.2.7) $$r_2 > l_4,$$

(A.2.2.8) $$r_1 > l_3,$$

(A.2.2.9) $$r_2 > l_3,$$

(A.2.2.10) $$r_3 > l_4,$$

(A.2.2.11) $$l_3 + l_5 + l_6 \geqq r_2 + r_4,$$

(A.2.2.12) $$l_1 + l_5 < (2/15) * m_2^{TB}.$$

Proof of Claim. In what follows we show that (A.2.2.2)-(A.2.2.12) hold as otherwise there is a contradiction.

Proof of (A.2.2.2)-(A.2.2.5). If any of these inequalities does not hold then replace in it r_j by $m_2^{TB}/6 - l_j$ and l_k by $m_2^{TB}/6 - r_k$ for appropriate values of j and k, and we obtain a contradiction to inequality (A.2.2.1).

Proof of (A.2.2.6). The proof for this part uses arguments similar to the ones in the proof of (A.2.1.4) in Subcase 2.1. The i-string applied in this case is "*rrlrll*" and inequalities (A.2.2) and (A.2.2.2) are used to establish the bound on the difference in vertical heights.

Proof of (A.2.2.7). Similar to the proof for (A.2.2.6).

Proof of (A.2.2.8). Same as above but applying the i-string "*rrrlll*" and using inequality (A.2.2.1).

Proof of (A.2.2.9). Similar to the proof for (A.2.2.8).

Proof of (A.2.2.10). Same as above but applying the i-string "*rlrrll*" and using inequalities (A.2.2.6) and (A.2.2.3).

Proof of (A.2.2.11). If $l_3 + l_5 + l_6 < r_2 + r_4$, then since (A.2.2.9) holds we know that applying the i-string "|*rlrll*" to D produces an assignment with a vertical height that differs from the one in D by at most

$$2 \min \{l_1, r_1\} + 2r_2 + 2r_4,$$

which we know must be greater than $(0.6) * m_2^{TB}$. Since $\min \{r_1, l_1\} \leqq m_2^{TB}/12$ and $r_j + l_j = m_2^{TB}/6$ for all j, the above inequality becomes

$$l_2 + l_4 < (7/60) * m_2^{TB}.$$

Also, applying the i-string "*rl*l***" to D generates an assignment whose vertical height differs from the one in D by at most

$$2 \max \{r_1, l_2 + l_4\} + m_2^{TB}/3,$$

which must be greater than $(0.6) * m_2^{TB}$. Substituting $l_2 + l_4 < (7/60) * m_2^{TB}$ and inequality (A.2.0) in the above expression results in the inequality $0.6 > 0.6$, a contradiction. So it must be that

$$l_3 + l_5 + l_6 \geqq r_2 + r_4.$$

Proof of (A.2.2.12). Applying the i-string "*rrllrl*" and using inequalities (A.2.2.8), (A.2.2.7) and (A.2.2.5), we know that

(eq.A.1) $$2(r_1 + r_2 + r_5) > (0.6) * m_2^{TB}.$$

Similarly, using "*rlrlrl*," (A.2.2.6), (A.2.2.10) and (A.2.2.4), we know that

(eq.A.2) $$2(r_1 + r_3 + r_5) > (0.6) * m_2^{TB}.$$

Substituting r_i by $1/6 - l_i$ in the inequality obtained by adding (eq.A.1), (eq.A.2) and four times $(l_1 + l_5 \geqq (2/15) * m_2^{TB})$, we know that

$$l_2 + l_3 < 2/15 * m_2^{TB}.$$

Now, applying the i-string "*rll****" to D generates an assignment whose vertical height differs from the one in D by at most

$$2 \max \{r_1, l_2 + l_3\} + m_2^{TB}/3,$$

which must be greater than $(0.6) * m_2^{TB}$. Since $\max\{r_1, l_2+l_3\} \leqq (2/15) * m_2^{TB}$, we know that $0.6 > 0.6$. A contradiction. So it must be that

$$l_1 + l_5 < (2/15) * m_2^{TB}.$$

End of Claim. □

We will now show that at least one of the above inequalities does not hold. There are two subcases:

Subcase A.2.2.1: $r_1 + r_3 \geqq l_2 + l_4 + l_5$. Applying the *i*-string "*rlrllr*" and using inequalities (A.2.2.6) and $r_1 + r_3 \geqq l_2 + l_4 + l_5$, we know that

$$2(r_1 + r_3 + r_6) > (0.6) * m_2^{TB}.$$

Similarly, using "*lrlrll*", (A.2.3) and (A.2.2.11), we know that

$$2l_1 + 2(l_3 + l_5 + l_6) > (0.6) * m_2^{TB}.$$

Adding these last two inequalities, we know that

$$l_5 > m_2^{TB}/10.$$

Since r_1 is less than $(2/15) * m_2^{TB}$, it must be that

$$l_1 > m_2^{TB}/30.$$

Adding these two inequalities, we know that $l_1 + l_5 > (2/15) * m_2^{TB}$, which contradicts (A.2.2.12).

Subcase A.2.2.2: $r_1 + r_3 < l_2 + l_4 + l_5$. Applying the *i*-string "*rlrllr*", and using inequalities (A.2.2.6) and $r_1 + r_3 < l_2 + l_4 + l_5$, we know that

(eq.A.3) $$2(l_2 + l_4 + l_5) + 2r_6 > (0.6) * m_2^{TB}.$$

Similarly, using "*lrlrll*," (A.2.3) and (A.2.2.11), we know that

(eq.A.4) $$2l_1 + 2(l_3 + l_5 + l_6) > (0.6) * m_2^{TB}.$$

Using "*lrlrlr*," (A.2.2.9) and (A.2.1), we know that

(eq.A.5) $$2l_1 + 2(r_2 + r_4 + r_6) > (0.6) * m_2^{TB}.$$

Replacing $l_i + r_i$ by $m_2^{TB}/6$ in $2 * (\text{eq.A.4}) + 2 * (\text{eq.A.2}) + (\text{eq.A.3}) + (\text{eq.A.5})$, we know that

$$l_1 + l_5 > (2/15) * m_2^{TB},$$

which contradicts (A.2.2.12). This completes the proof of this subcase and Case 2.

Case 3: $\max\{l_i, r_i\} < (2/15) * m_2^{TB}$, and $r_2 + r_4 \leqq l_3 + l_5$.

The proof for this case is symmetric to the one for Case 2.

This completes the proof of the lemma. □

Appendix B.

LEMMA B.1. *Let D be an optimal assignment such that $D' \subseteq D$ and let I be an i-string with three "*" symbols and three "." symbols. Applying the i-string I to D generates an assignment M with*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (1/3) * m_2^{TB}.$$

It is assumed that when D is transformed to any of the assignments obtainable from D and I, all the interchanges of pairs of paths that cross on the top side of G are not type I interchanges.

Proof. It is simple to show that the observations (i) and (ii) given in the proof of Lemma 5.1 hold for this case.

Let G_i, for $1 \leqq i \leqq 3$, represent the ith group of nets (from left to right) for which there is a "*" symbol in I. For assignment D, let l_i be the number of nets in G_i that are connected by a path that crosses the left side of G and let r_i be the number of nets in G_i that are connected by a path that crosses the right side of G. In what follows we will use i-strings of size three. Their meaning is obvious.

We prove our lemma by contradiction. Suppose that all assignments M obtained from D by using the i-string "***" do not satisfy the lemma.

Applying the i-string "*rll*" to D generates an assignment whose vertical height differs from the one in D by at most

$$2 \max \{r_1, l_2 + l_3\},$$

which we know must be greater than $(1/3) * m_2^{TB}$. If $r_1 \geqq l_2 + l_3$ then it must be that $r_1 > (1/6) * m_2^{TB}$, a contradiction. Therefore, $l_2 + l_3 > (1/6) * m_2^{TB}$, which is equivalent to

$$l_3 > r_2. \tag{B.1}$$

Applying the i-string "*lrl*" to D generates an assignment whose vertical height differs from the one in D by at most

$$2 * l_1 + 2 * \max \{r_2, l_3\},$$

which we know must be $> (1/3) * m_2^{TB}$. Substituting inequality (B.1) in this inequality, we know that

$$l_1 + l_3 > (1/6) * m_2^{TB}. \tag{B.2}$$

Using similar arguments with the i-strings "*rrl*" and "*rlr*," we know that

$$r_1 + r_3 > (1/6) * m_2^{TB}.$$

Adding this inequality to inequality (B.2) gives that the number of elements in G_1 and G_3 is greater than $(1/3) * m_2^{TB}$. This is a contradiction. Hence, at least one of the assignments obtained from the above i-strings satisfies the lemma. This completes the proof of the lemma. □

Appendix C.

LEMMA C.1. *Let D be an optimal assignment such that $D' \subseteq D$ and let I be an i-string with five "*" symbols and one "." symbol. Applying the i-string I to D generates an assignment such that*

$$H_M(S_1, S_2) + H_M(S_3, S_0) \leqq H_D(S_1, S_2) + H_D(S_3, S_0) + (8/15) * m_2^{TB}.$$

It is assumed that when D is transformed to any of the assignments obtainable from D and I, all the interchanges of pairs of paths that cross on the top side of G are not type I interchanges.

Proof. It is simple to show that the observations (i) and (ii) given in the proof of Lemma 5.1 hold for this case.

Let G_i, for $1 \leqq i \leqq 5$, represent the ith group of nets (from left to right) for which there is a "*" symbol in I. For assignment D, let l_i be the number of nets in G_i that are connected by a path that crosses the left side of G and let r_i be the number of nets in G_i that are connected by a path that crosses the right side of G. In what follows we will use i-strings of size five. Their meaning is obvious.

We prove our lemma by contradiction. Suppose that all assignments M obtained from D by applying the i-string "*****" do not satisfy the lemma. There are two cases:

Case 1.

(C.0) $$r_1 + r_3 > l_2 + l_4.$$

CLAIM.

(C.1) $$r_1 + r_2 > l_3 + l_4,$$

(C.2) $$r_1 > l_2.$$

Proof of Claim. We now show that the above inequalities hold as otherwise there is a contradiction. We prove each part separately.

Proof of (C.1). The proof can be easily obtained by replacing r_j by $(1/6) * m_2^{TB} - l_j$ and l_k by $(1/6) * m_2^{TB} - r_k$, for appropriate values of j and k, in inequality (C.0).

Proof of (C.2). Applying the i-string "*rrll*|" to D and using inequality (C.1) gives that the increase of the vertical height is at most

$$2(r_1 + r_2) + 2 \min \{l_5, r_5\},$$

which we know must be greater than $(8/15) * m_2^{TB}$. Since $\min \{l_5, r_5\} \leqq (1/12) * m_2^{TB}$, we know that

$$r_1 + r_2 > (11/60) * m_2^{TB}.$$

If $r_1 \leqq l_2$ then substituting in the above inequality, we know that

$$l_2 + r_2 > (11/60) * m_2^{TB}.$$

But $l_2 + r_2 = (1/6) * m_2^{TB}$, a contradiction. Hence, r_1 must be greater than l_2.

End of Claim. □

We now show that when (C.1) and (C.2) hold there is a contradiction. There are two subcases.

Subcase 1.1.

(C.1.0) $$r_2 + r_4 < l_3 + l_5.$$

CLAIM.

(C.1.1) $$r_2 + r_3 < l_4 + l_5,$$

(C.1.2) $$r_4 < l_5.$$

Proof of Claim. We now prove if any of the above inequalities does not hold there is a contradiction. We prove each part separately.

Proof of (C.1.1). Same as the proof for (C.1) but uses inequality (C.1.0).

Proof of (C.1.2). Same as the proof for (C.2) but uses the i-string "|*rrll*" and inequality (C.1.1).

End of Claim. □

We now show that when the above inequalities hold there is a contradiction. Applying the i-string "*rlrlr*" to D and using inequalities (C.2) and (C.0) we know that the increase in vertical height is

$$2(r_1 + r_3) + 2r_5$$

which we know must be greater than $(8/15) * m_2^{TB}$.

Using similar arguments with i-string "*lrlrl*", and inequalities (C.1.2) and (C.1.0), we know that

$$2l_1 + 2(l_3 + l_5) > (8/15) * m_2^{TB}.$$

Adding these last two inequalities and replacing $r_j + l_j$ by 1/6, we know that

$$1 > 16/15,$$

a contradiction. This completes the proof of this subcase.

Subcase 1.2.

(C.2.0) $$r_2 + r_4 \geqq l_3 + l_5.$$

CLAIM.

(C.2.1) $$r_2 + r_3 \geqq l_4 + l_5,$$

(C.2.2) $$r_2 > l_3,$$

(C.2.3) $$r_1 + r_2 < l_3 + l_4 + l_5,$$

(C.2.4) $$r_1 + r_3 < l_2 + l_4 + l_5,$$

(C.2.5) $$r_2 + r_3 > (4/15) * m_2^{TB}.$$

Proof of Claim. We now show that if any of the above inequalities does not hold then there is a contradiction. We prove each part separately.

Proof of (C.2.1). The proof for this part is similar to the proof for (C.1). The main difference is that we use inequality (C.2.0).

Proof of (C.2.2). The proof for this case follows the same arguments as the ones in the proof for (C.2) but uses the *i*-string "|*rrll*" and inequality (C.2.1).

Proof of (C.2.3). If $r_1 + r_2 \geqq l_3 + l_4 + l_5$ then applying the *i*-string "*rrlll*" to D generates an assignment whose vertical height differs from the one in D by at most

$$2(r_1 + r_2),$$

which we know must be greater than $(8/15) * m_2^{TB}$. Replacing r_j by $(1/6) * m_2^{TB} - l_j$, we know that

$$l_1 + l_2 < (1/15) * m_2^{TB}.$$

Applying the *i*-string "*ll****" to D, we know that

$$2(l_1 + l_2) + (1/3) * m_2^{TB} > (8/15) * m_2^{TB}.$$

Substituting the first of these inequalities into the second, we know that

$$7/15 > 8/15,$$

a contradiction.

Proof of (C.2.4). The proof for this case is by replacing r_2 by $(1/6) * m_2^{TB} - l_2$ and l_3 by $(1/6) * m_3^{TB} - r_3$ in (C.2.3).

Proof of (C.2.5). Applying the *i*-string "*rlrlr*" to D and using inequalities (C.2) and (C.0), we know that

(eq.C1) $$2(r_1 + r_3) + 2r_5 > (8/15) * m_2^{TB}.$$

Applying the *i*-string "*lrlrl*" to D and using inequalities (C.2.2) and (C.2.0), we know that

(eq.C2) $$2l_1 + 2(r_2 + r_4) > (8/15) * m_2^{TB}.$$

Applying the *i*-string "*rrlll*" to D and using inequality (C.2.3), we know that

(eq.C3) $$2(l_3 + l_4 + l_5) > (8/15) * m_2^{TB}.$$

Applying the i-string "$rlrll$" to D and using inequalities (C.2) and (C.2.4), we know that

(eq.C4) $$2(l_2+l_4+l_5) > (8/15) * m_2^{TB}.$$

Replacing $r_j + l_j$ by $(1/6) * m_2^{TB}$ in the expression

$$2(\text{eq.C1}) + 2(\text{eq.C2}) + (\text{eq.C3}) + (\text{eq.C4}),$$

we know that

$$r_2 + r_3 > (4/15) * m_2^{TB}.$$

End of Claim. □

We now show that if the above inequalities hold then there is a contradiction.

Applying the i-string "$*ll**$" to D generates an assignment D whose vertical height differs from the one in D by at most

$$2(l_2+l_3) + (1/3) * m_2^{TB},$$

which we know must be greater than $(8/15) * m_2^{TB}$. Replacing l_j by $(1/6) * m_2^{TB} - r_j$, we know that

$$r_2 + r_3 < (7/30) * m_2^{TB}.$$

But this contradicts (C.2.5). This completes the proof of this subcase and Case 1.

Case 2.

(C.3) $$r_1 + r_3 \leqq l_2 + l_4.$$

Subcase 2.1. $r_2 + r_4 < l_3 + l_5$.

The proof of this case is symmetric to the one for Subcase 1.2 in Case 1.

Subcase 2.2.

(C.4) $$r_2 + r_4 \geqq l_3 + l_5.$$

CLAIM.

(C.4.1) $$r_1 + r_2 \leqq l_3 + l_4,$$

(C.4.2) $$r_2 + r_3 \geqq l_4 + l_5,$$

(C.4.3) $$r_3 < l_4,$$

(C.4.4) $$r_2 > l_3.$$

Proof of Claim. We now show that if any of the above inequalities does not hold then there is a contradiction. We prove each part separately.

Proof of (C.4.1). The proof for this part is similar to the proof for (C.1). The main difference is that we use inequality (C.3).

Proof of (C.4.2). The proof for this part is similar to the proof for (C.1). The main difference is that we use inequality (C.4).

Proof of (C.4.3). The proof for this case follows the same arguments as the ones in the proof for (C.2) but uses inequality (C.4.1).

Proof of (C.4.4). The proof for this case follows the same arguments as the ones in the proof for (C.2) but uses the i-string "$|rrll$" and inequality (C.4.2).

End of Claim. □

We now show that when the above inequalities hold there is a contradiction. Applying the i-string "$rlrl|$" to D and using inequalities (C.4.3) and (C.3) we know that the increase in vertical height is at most

$$2(l_2+l_4)+2\min\{l_5, r_5\}$$

which we know must be greater than $(8/15) * m_2^{TB}$.

Using similar arguments with i-string "$|rlrl$" and inequalities (C.4.4) and (C.4), we know that

$$2r_2+2r_4+2\min\{l_1, r_1\} > (8/15) * m_2^{TB}.$$

Adding these last two inequalities and replacing r_j+l_j by $1/6$ and $\min\{l_k, r_k\} \leqq (1/12) * m_2^{TB}$ results in the inequality

$$1 > 16/15,$$

a contradiction. This completes the proof of this subcase and Case 2.

This completes the proof of the lemma. □

REFERENCES

[AHU] A. AHO, J. HOPCROFT AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1975.

[GL1] T. GONZALEZ AND S. LEE, *An optimal algorithm for optimal routing around a rectangle*, Proc. 20th Annual Allerton Conference on Comm. Control and Computing, Univ. of Illinois, Urbana, IL, October, 1982, pp. 636-645.

[GL2] ———, *An $O(n \log n)$ algorithm for optimal routing around a rectangle*, Technical Report #116, Programs in Computer Science, The University of Texas at Dallas, Dallas, TX, November 1982. (Revised May 1985.)

[GL3] ———, *Routing multiterminal nets around a rectangle*, IEEE Trans. Comput., C-35 (1986), pp. 543-549.

[GLL] U. I. GUPTA, D. T. LEE AND J. LEUNG, *An optimal solution for the channel-assignment problem*, IEEE Trans. Comput., C-28 (1979), pp. 807-810.

[HS] A. HASHIMOTO AND J. E. STEVENS, *Wire routing by optimizing channel assignment without large apertures*, Proc. 8th IEEE Design Automation Conference, 1971, pp. 155-169.

[L] A. S. LAPAUGH, *A polynomial time algorithm for optimal routing around a rectangle*, Proc. 21st IEEE Foundations of Computer Science, 1980, pp. 282-293.

[La] ———, *Algorithms for integrated circuit layout, an analytic approach*, Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1980

[MC] C. MEAD AND L. CONWAY, *Introduction to* VLSI *Systems*, Addison-Wesley, Reading, MA, 1980.

[R] R.L. RIVEST, *The* PI (*Placement and Interconnect*) *System*, Proc. 19th IEEE Design Automation Conference, pp. 475-481.

[SBR] S. SAHNI, A. BHATT and R. RAGHAVAN, *The complexity of design automation problems*, Proc. 17th Design Automation Conference, June 1980, pp. 402-411.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

© 1987 Society for Industrial and Applied Mathematics
007

LIST ORGANIZING STRATEGIES USING STOCHASTIC MOVE-TO-FRONT AND STOCHASTIC MOVE-TO-REAR OPERATIONS*

B. JOHN OOMMEN† AND E. R. HANSEN‡

Abstract. Consider a list of elements $\{R_1, \cdots, R_N\}$ in which the element R_i is accessed with an (unknown) probability s_i. If the cost of accessing R_i is proportional to i (as in sequential search) then it is advantageous if each access is accompanied by a simple reordering operation. This operation is chosen so that ultimately the list will be sorted in the descending order of the access probabilities.

In this paper we present two list organizing schemes, the first of which uses bounded memory and the second of which uses memory proportional to the number of elements in the list. Both of the schemes reorder the list by moving only the accessed element. However, as opposed to the schemes discussed in the literature the move operation is performed stochastically in such a way that ultimately no more move operations are performed. When this occurs we say that the scheme has converged. We shall show that:

(i) The bounded memory stochastic move-to-front algorithm is expedient, but is always worse than the deterministic move-to-front algorithm.

(ii) The linear-memory stochastic move-to-rear scheme is optimal, independent of the distribution of the access probabilities. By this we mean that although the list could converge to one of its $N!$ configurations, by suitably updating the probability of performing the move-to-rear operation, the probability of converging to the right arrangement can be made as close to unity as desired.

Key words. dynamic list ordering, move-to-front rule, adaptive leaning, self-organizing lists, stochastic list operations

AMS(MOS) subject classifications. 68P05, 68P10, 68P20, 68Q20

1. Introduction. Suppose we are given a set of elements $\{R_1, \cdots, R_N\}$. At every time instant one of these elements is accessed. Further, the element R_j is accessed with an unknown access probability s_j. We assume that the accesses are made independently. Whenever an element R_j is accessed, a sequential search is performed on the list. To minimize the cost of accessing, it is desirable that the records are ordered in the descending order of their access probabilities. We shall refer to a file ordered in this way as a completely organized file.

McCabe [11, pp. 398–399], [12] was the first to propose a solution to this problem. His solution rendered the list dynamically self-organizing and it involved moving an element to the front of the list every time it was accessed. Using this rule, the move-to-front (MTF) rule, the limiting average value of the number of probes done per access has the value C_{MTF}, where

$$C_{\mathrm{MTF}} = 0.5 + \sum_{i,j} (s_i s_j)/(s_i + s_j).$$

Many other researchers [3], [6]–[9], [14] have also extensively studied the MTF rule and various properties of its limiting convergence characteristics are available in the literature.

McCabe [11, pp. 398–399], [12] also introduced a scheme which is called the transposition rule. This rule requires that an accessed element is interchanged with its preceding element in the list, unless, of course, it is at the front of the list. Much literature is available on the transposition rule [1], [2], [4], [11]–[13] but particularly important is the work of Rivest [13] and Tenenbaum et al. [1], [15] who extensively

* Received by the editors May 20, 1985; accepted for publication (in revised form) November 11, 1986. This research was partially supported by the Natural Sciences and Engineering Research Council of Canada. A preliminary version was presented at the 1986 ICDT Conference at Rome, Italy.

† School of Computer Science, Carleton University, Ottawa, K1S 5B6, Canada.

‡ Lockheed Missiles and Space Co. Inc., Sunnyvale, California 94086.

studied this rule and suggested its generalizations—the Move-k-Ahead rule and the POS (k) rule. In the former, the accessed element is moved k-positions forward to the front of the list unless it is found in the first k positions—in which case it is moved to the front. The POS (k) rule moves the accessed element to position k of the list if it is in positions $k+1$ through n. It transposes it with the preceding element if it is in positions 2 through k. If it is the first element it is left unchanged.

Rivest [13] showed that the limiting behaviour of the transposition rule (quantified in terms of the average number of probes) was never worse than that of the MTF rule. He conjectured that the transposition rule has lower expected cost than any other reorganization scheme. He also conjectured that the Move-k-Ahead rule was superior to the Move-$k+1$-Ahead rule; but as yet this is unproven. Tenenbaum and Nemes [15] proved various results for the POS (k) rule primarily involving a distribution in which $s_2 = s_3 = \cdots = s_N = (1-s_1)/(N-1)$. Their results seem to strengthen Rivest's conjecture.

All of the schemes discussed in the literature are represented by Markov chains which are ergodic. By virtue of this fact, the list can be in any one of its $N!$ configurations, even in the limit. Thus, for example, a list which was completely organized can be thoroughly disorganized by the MTF rule by a *single* request for the element which is accessed most infrequently. Observe that after this unfortunate occurrence, it will take a long time for the list to be organized again, i.e., for this element to dribble its way to the tail of the list.

In this paper we propose two learning algorithms in which the elements of the list adaptively learn to find their place. The first algorithm is essentially a move-to-the-front algorithm with the exception that on the nth access, the accessed element is moved to the front of the list with a probability $f(n)$. This probability is systematically decreased every time an element is accessed. Ultimately on being accessed, each element tends to stay in the place where it is (as opposed to moving to the front of the list). In other words, the Markovian representation of the procedure is absorbing, as opposed to ergodic. The organization of the list gets "absorbed" into one of the $N!$ orderings. But, we shall show that the scheme is expedient, i.e., if $s_i > s_j$, then the probability of absorption into an arrangement in which R_i precedes R_j is always greater than 0.5.

The second algorithm which we present is far more powerful. In this case, the algorithm is a Move-to-Rear scheme in which the accessed element R_j is moved to the *rear* of the list with a probability q_j. This quantity q_j is progressively decremented every time the *element* is accessed. As before, ultimately there is no move operation performed on the list. Thus this scheme too is absorbing in its Markovian representation and so could converge into any one of its $N!$ arrangements. However, in this case, we shall show that the probability of converging to the optimal arrangement can be made as close to unity as desired.

The techniques introduced here require more workspace than the traditionally used methods. However, apart from the latter algorithm being much more accurate than all the algorithms reported in the literature, it is also computationally more efficient. This is because the access of an element requires the update of *exactly* one probability (namely the probability q_j associated with the accessed element R_j). Further, unlike the contemporary algorithms, since the list operations are essentially stochastic, a list operation is *not* necessarily performed on every access. Finally, since the Markovian representation of the scheme is absorbing, the number of list operations performed asymptotically decreases to zero.

As in the literature concerning the theory of adaptive learning, we shall use the terms algorithm, rule and scheme interchangeably. Further, for the sake of simplicity

we shall assume that the access probabilities of the records are distinct. The cases when they are nondistinct are those when there are *many* optimal configurations. A remark about these cases will be made appropriately in the body of the paper.

2. Bounded memory probabilistic move-to-front operations. The concept of performing probabilistic move operations on an accessed element is not entirely new. Kan and Ross [9] suggested a probabilistic transposition scheme and showed that no advantage was obtained by rendering the scheme probabilistic. Their scheme, however, required that the probability of performing the operation, be *time invariant.* As opposed to this, we shall define move operations which are essentially probabilistic, but the probabilities associated with the move operations are dynamically varied.

Let $f(n)$ be the probability (at time "n") of any element being moved to the front of the list on being accessed. Observe that this implies that an element, on being accessed, stays where it is with probability $(1-f(n))$. For an initial condition we define

$$f(0)=a. \tag{2.1}$$

The probability $f(n)$ is updated every time any record is accessed. The updating scheme is given by (2.2) below for $0<a<1$:

$$f(n+1)=af(n) \quad \text{every time a record is accessed.} \tag{2.2}$$

The quantity "a" is defined as the updating constant.

Let ${}_iP_j(n)$ be the expected probability of record R_i succeeding R_j at the nth time instant. Clearly ${}_jP_i(n)=1-{}_iP_j(n)$, for all n. We shall derive the transient and asymptotic properties of ${}_iP_j(n)$. To do this we need the following lemma.

LEMMA 1. *Let A be any $n\times n$ matrix with distinct eigenvalues. Let K be the matrix which diagonalizes A. Let*

$$B(n)=I+a^nA.$$

Then, $B(n)$ is diagonalizable by the same matrix K for all n.

Proof. Since A has distinct eigenvalues, and K diagonalizes A

$$K^{-1}AK=\text{Diag}\,(\theta_1,\cdots,\theta_N)$$

where $\text{Diag}\,(\theta_1,\cdots,\theta_N)$ is the diagonal matrix with the eigenvalues of A on its diagonal. Let $B(n)=I+a^nA$. Then

$$\begin{aligned}K^{-1}B(n)K&=K^{-1}(I+a^nA)K\\&=I+a^nK^{-1}AK\\&=I+a^n\cdot\text{Diag}\,(\theta_1,\cdots,\theta_N)\end{aligned}$$

and the lemma is proved.

Using the above lemma and the theory of Markov chains, we prove the following theorems.

THEOREM 1. *Let ${}_iP_j(n)$ be the expected probability of R_i succeeding R_j at the nth time instant. Then, ${}_iP_j(n)$ and ${}_jP_i(n)$ obey the following time varying Markov equation*:

$$\begin{bmatrix}{}_iP_j(n+1)\\{}_jP_i(n+1)\end{bmatrix}=[B(n)]\begin{bmatrix}{}_iP_j(n)\\{}_jP_i(n)\end{bmatrix}$$

where

$$B(n)=\begin{bmatrix}1-a^ns_i & a^ns_i\\ a^ns_j & 1-a^ns_j\end{bmatrix}T.$$

Proof. R_i succeeds R_j at time instant "$n+1$" if and only if:

(a) R_i succeeded R_j at time instant "n" and no list operation was performed, or

(b) R_j was accessed and it was moved to the front of the list. Observe that R_i cannot succeed R_j if R_i was accessed and moved to the front.

Let ${}_ip_j(n) = \text{Prob}[R_i \text{ succeeds } R_j \text{ at time "}n\text{"}]$. Clearly, $E[{}_ip_j(n)] = {}_iP_j(n)$. The above leads to the following recursive definition of ${}_ip_j(n)$:

$$\begin{aligned} {}_ip_j(n+1) &= 0 && \text{if } R_i \text{ accessed and MTF performed,} \\ &= 1 && \text{if } R_j \text{ accessed and MTF performed,} \\ &= {}_ip_j(n) && \text{otherwise.} \end{aligned}$$

Observe that the probabilities of the events defined above are readily available in terms of the unknown access probabilities. Further, an MTF operation is performed at "n" with a probability a^n. Thus,

$$\begin{aligned} {}_ip_j(n+1) &= 0 && \text{with probability } s_i \cdot a^n, \\ &= 1 && \text{with probability } s_j a^n, \\ &= {}_ip_j(n) && \text{with probability } 1-(s_i+s_j)a^n. \end{aligned} \tag{2.3}$$

Taking conditional expectations, we have

$$E[{}_ip_j(n+1)|{}_ip_j(n)] = a^n s_j + {}_ip_j(n) - {}_ip_j(n) \cdot a^n(s_i+s_j).$$

Taking expectations again and observing that $E[{}_ip_j(n)] = {}_iP_j(n)$, we get

$${}_iP_j(n+1) = [1-a^n(s_i+s_j)]\,{}_iP_j(n) + s_j \cdot a^n.$$

Since ${}_iP_j(n) + {}_jP_i(n) = 1$, we expand the constant term as

$$s_j a^n = s_j a^n[{}_iP_j(n) + {}_jP_i(n)].$$

Thus

$${}_iP_j(n+1) = [1-a^n s_i]\,{}_iP_j(n) + [a^n s_j]\,{}_jP_i(n).$$

This leads to the following matrix equation:

$$\begin{bmatrix} {}_iP_j(n+1) \\ {}_jP_i(n+1) \end{bmatrix} = \begin{bmatrix} 1-a^n s_i & a^n s_j \\ a^n s_i & 1-a^n s_j \end{bmatrix} \begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix},$$

and the theorem is proved.

THEOREM 2. *The constant matrix K where*

$$K = \begin{bmatrix} 1 & 1 \\ s_i/s_j & -1 \end{bmatrix}$$

diagonalizes $B(n)$, for all values of n.

Proof. Observe that $B(n)$ is of the form

$$B(n) = I + a^n A$$

where

$$A = \begin{bmatrix} -s_i & s_j \\ s_i & -s_j \end{bmatrix}.$$

Since $B(n)$ is a stochastic matrix, we know that one of its eigenvalues is unity. Further, since the sum of the eigenvalues is equal to the trace of the matrix, the second eigenvalue is $1-a^n(s_i+s_j)$. Using Lemma 1, we know that the matrix which diagonalizes $B(0)$ also diagonalizes $B(n)$ for all n. In this case, it is easy to see that the eigenvectors for $B(0)$ are as follows:

(i) $[1 \;\; s_i/s_j]^T$ for the eigenvalue unity, and

(ii) $[1 \;\; -1]^T$ for the eigenvalue $1-(s_i+s_j)$.

Thus, the constant matrix K where

$$K=\begin{bmatrix} 1 & 1 \\ s_i/s_j & -1 \end{bmatrix}$$

diagonalizes $B(n)$ for all n. This proves the theorem.

Remark. By performing elementary operations, it is easy to see that K^{-1} has the form:

$$K^{-1}=\frac{1}{s_i+s_j}\begin{bmatrix} s_j & s_j \\ s_i & -s_j \end{bmatrix}. \tag{2.4}$$

One can trivially verify that

$$K^{-1}B(n)K=\text{Diag}\,(1,\,1-a^n(s_i+s_j))$$

where

$$\text{Diag}\,(1,\,1-a^n(s_i+s_j))=\begin{bmatrix} 1 & 0 \\ 0 & 1-a^n(s_i+s_j) \end{bmatrix}.$$

Similarly, $K\cdot\text{Diag}\,(1,\,1-a^n(s_i+s_j))\cdot K^{-1}$ is exactly $B(n)$.

THEOREM 3. *The value of ${}_iP_j(n)$ for an updating constant "a" obtained by solving the Markov equation given by Theorem* 1, *has the form*

$${}_iP_j(n)=\frac{s_i}{s_i+s_j}Q_{a,n}+\frac{s_j}{s_i+s_j}(1-Q_{a,n})$$

where

$$Q_{a,n}=0.5\left[\prod_{k=0}^{n-1}(1-a^k(s_i+s_j))\right].$$

Proof. From the results of Theorem 1, we can see that

$$\begin{bmatrix} {}_iP_j(n+1) \\ {}_jP_i(n+1) \end{bmatrix}=B(n)\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix}.$$

Thus, the solution of the matrix difference equation yields

$$\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix}=B(n-1)B(n-2)\cdots B(0)\begin{bmatrix} {}_iP_j(0) \\ {}_jP_i(0) \end{bmatrix}=\prod_{k=0}^{n-1}B(k)\begin{bmatrix} {}_iP_j(0) \\ {}_jP_i(0) \end{bmatrix}.$$

Rewriting each $B(k)$ in terms of the diagonal matrix $\text{Diag}\,(1,\,1-a^k(s_i+s_j))$, we have

$$\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix}=\left(\prod_{k=0}^{n-1}K\cdot[\text{Diag}\,(1,\,1-a^k(s_i+s_j))]\cdot K^{-1}\right)\cdot\begin{bmatrix} {}_iP_j(0) \\ {}_jP_i(0) \end{bmatrix}.$$

Since the product of each consecutive pair $K \cdot K^{-1}$ yields the identity matrix, we write

$$\begin{aligned}\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix} &= K\left[\prod_{k=0}^{n-1} \text{Diag}\,(1, 1-a^k(s_i+s_j))\right] \cdot K^{-1} \cdot \begin{bmatrix} {}_iP_j(0) \\ {}_jP_i(0) \end{bmatrix} \\ &= K \cdot \begin{bmatrix} 1 & 0 \\ 0 & \prod_{k=0}^{n-1}(1-a^k(s_i+s_j)) \end{bmatrix} K^{-1} \begin{bmatrix} {}_iP_j(0) \\ {}_jP_i(0) \end{bmatrix}.\end{aligned} \tag{2.5}$$

Let $Q_{a,n} = \frac{1}{2}\prod_{k=0}^{n-1}(1-a^k(s_i+s_j))$. Since, with no loss of generality, ${}_iP_j(0) = {}_jP_i(0) = 0.5$, we expand (2.5) above to yield

$$\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix} = \frac{1}{s_i+s_j}\begin{bmatrix} 1 & 1 \\ s_i/s_j & -1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 2\cdot Q_{a,n} \end{bmatrix}\begin{bmatrix} s_j & s_j \\ s_i & -s_j \end{bmatrix}\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

After considerable simplification this results in

$$\begin{bmatrix} {}_iP_j(n) \\ {}_jP_i(n) \end{bmatrix} = \frac{1}{s_i+s_j}\begin{bmatrix} s_j(1-Q_{a,n}) + s_iQ_{a,n} \\ s_i(1-Q_{a,n}) + s_jQ_{a,n} \end{bmatrix} \tag{2.6}$$

and the theorem is proved.

Remark. Observe that ${}_iP_j(n) = {}_jP_i(n) = 0.5$ if $s_i = s_j$. This is intuitively satisfying.

To prove the asymptotic value of ${}_iP_j(n)$ we need the following lemma.

LEMMA 2. *The infinite product*

$$\prod_{k=1}^{\infty}(1+b_k) \qquad (\textit{where } b_k \neq -1 \textit{ for all } k)$$

tends to a nonzero finite limit if and only if the infinite sum $\sum_{k=1}^{\infty} b_k$ *is convergent.*

Proof. The lemma is proved in Titchmarsh [17, pp. 13–15].

THEOREM 4. *The stochastic bounded memory Move-to-Front Algorithm is asymptotically always less accurate than the deterministic Move-to-Front Algorithm.*

Proof. Consider the term for ${}_iP_j(n)$ as

$${}_iP_j(n) = \frac{s_j}{s_i+s_j}(1-Q_{a,n}) + \frac{s_i}{s_i+s_j}Q_{a,n}$$

where $Q_{a,n}$ is defined in Theorem 3.

Using Lemma 2, it is clear that $Q_{a,n} > 0$ as $0 < a < 1$, but tends to zero as a tends to unity. Differentiating with respect to a, we obtain

$$\frac{\delta_i P_j(n)}{\delta a} = (s_j - s_i) \cdot Q_{a,n} \cdot \sum_{k=1}^{n-1} \frac{k \cdot a^{k-1}}{1-a^k(s_i+s_j)}.$$

Since $0 < a < 1$ and $0 < s_i + s_j < 1$, we have

$$0 < 1 - a^k(s_i+s_j) < 1 \quad \text{for all } k \geqq 1.$$

Assume that with no loss of generality that $s_i < s_j$. This tells us that

$$\frac{\delta_i P_j(n)}{\delta a} > 0. \tag{2.7}$$

In other words, ${}_iP_j(n)$ has no stationary point with respect to a in the interval $0 < a < 1$. Further, due to (2.7), the largest value of ${}_iP_j(n)$ occurs when $a = 1$. Since (2.7) is true for finite and infinite values of n, the value of ${}_iP_j(\infty)$ is maximized at the largest acceptable value of a and the theorem is proved.

COROLLARY 4.1. *The stochastic bounded memory Move-to-Front Algorithm is expedient independent of the access distribution of the records.*

Proof. Due to the multiplying factor of 0.5, and the previous theorem, it is easy to see that for all a, $0 < Q_{a,n} < 0.5$. The result is now obvious since ${}_iP_j(n)$ is merely a convex combination of $s_j/(s_i+s_j)$ and $s_i/(s_i+s_j)$ weighted by $(1-Q_{a,n})$ and $Q_{a,n}$, respectively.

Remark. Throughout this discussion it was assumed that the single memory location that stores $f(n)$ can contain an arbitrarily small positive real number. In practice, however, all that we need to store is an index, n, of the time that has lapsed since the file reorganization scheme was initiated. From this index, $f(n)$ can be computed trivially, since

$$f(n) = a^n.$$

It is also appropriate to observe that the maximum number that this index should attain is governed by the uniform random number generator accessible to the system. If the smallest positive number yielded by the random number generator is $x_{\min}$, then the memory location which stores n need not store numbers larger than $n_{\max}$, where

$$n_{\max} = \lceil \log_a(x_{\min}) \rceil.$$

We now proceed to study the linear memory stochastic move-to-rear scheme.

3. Linear memory probabilistic move-to-rear operations. In this section we shall show how a probabilistic scheme with an absorbing Markovian representation can indeed be made asymptotically optimal. The idea is essentially one of moving the accessed element R_j to the rear of the list with a probability q_j which is systematically updated so that, as in the previous case, in the limit, no list operations are performed. In this case, we shall show that the list converges to one of the $N!$ possible arrangements, and the probability of converging to the optimal one can be made arbitrarily close to unity. Let $q_j(n)$ be the probability (at time "n") of moving the element R_j to the *rear* of the list on *being* accessed. This implies that on being accessed, the element stays where it is with probability $(1-q_j(n))$.

Initially, we set $q_j(0) = 1$ for all $j = 1, \cdots, N$. After this, the quantity $q_j(n)$ is decremented every time R_j is accessed as below:

$$\begin{aligned} q_j(n+1) &= aq_j(n) \quad \text{if } R_j \text{ is accessed,} \\ &= q_j(n) \quad \text{otherwise.} \end{aligned} \tag{3.1}$$

Observe that the updating scheme is reminiscent of the Linear Reward-Inaction scheme studied extensively in the area of adaptive learning.

We now derive the properties of $q_j(n)$. To simplify notation, unless explicitly stated q_j will refer to the quantity $q_j(n)$.

THEOREM 5. *$E[q_i(n)]$ decreases monotonically with time. Further, for all $j \neq k$,*

$$s_j > s_k \quad \textit{if and only if } E[q_j(n)] < E[q_k(n)]. \tag{3.2}$$

Proof. Consider the random variable $q_i(n+1)$. By virtue of (3.1), the latter has the following distribution:

$$\begin{aligned} q_i(n+1) &= aq_i \quad \text{with probability } s_i, \\ &= q_i \quad \text{with probability } (1-s_i). \end{aligned} \tag{3.3}$$

Thus

$$\begin{aligned} E[q_i(n+1)|q_i] &= aq_is_i + q_i(1-s_i) \\ &= q_i(1-(1-a)s_i) \\ &= e_iq_i \quad \text{where } e_i = 1-(1-a)s_i. \end{aligned} \tag{3.4}$$

Note that since $0<a<1$ for all i, e_i obeys $0<e_i<1$. Taking expectations again, we obtain

$$E[q_i(n+1)] = e_iE[q_i(n)]. \tag{3.5}$$

The difference equation (5) subject to the initial condition $q_i(0)=1$, yields the solution

$$E[q_i(n)] = e_i^n.$$

Clearly, $E[q_i(n)]$ is monotonically decreasing with n. Further,

$$s_j > s_k \Leftrightarrow e_j < e_k \Leftrightarrow e_j^n < e_k^n,$$

and the theorem is proved.

COROLLARY 5.1. *For all i,*

$$\lim_{n\to\infty} q_i(n) = 0 \quad \textit{with probability } 1.$$

The result follows since the Markov process $\{q_i(n)\}$ has only one absorbing barrier, namely the probability zero [10].

THEOREM 6. *For $n>0$, $\mathrm{Var}[q_i(n)]$ decreases monotonically with n and its limiting value is zero.*

Proof. From the distribution of $q_i(n)$ given by (3.3), we obtain

$$\begin{aligned} q_i^2(n+1) &= a^2q_i^2 \quad \text{with probability } s_i, \\ &= q_i^2 \quad \text{with probability } (1-s_i). \end{aligned}$$

Thus,

$$E[q_i^2(n+1)|q_i] = [1-(1-a^2)s_i]q_i^2. \tag{3.6}$$

Taking expectations again, we obtain the difference equation

$$E[q_i^2(n+1)] = [1-(1-a^2)s_i]E[q_i^2(n)].$$

Since $E[q_i^2(0)]=1$, we have

$$E[q_i^2(n)] = [1-(1-a^2)s_i]^n. \tag{3.7}$$

Consider now the expression for $\mathrm{Var}[q_i(n+1)|q_i]$. Using the form of $E[q_i(n+1)|q_i]$ from Theorem 5, we obtain

$$\begin{aligned} \mathrm{Var}[q_i(n+1)|q_i] &= E[q_i^2(n+1)|q_i] - E^2[q_i(n+1)|q_i] \\ &= [1-(1-a^2)s_i]q_i^2 - [1-(1-a)]^2q_i^2 \\ &= (1-a)^2s_i(1-s_i)q_i^2. \end{aligned}$$

Taking expectations again and using (3.7), we get

$$\mathrm{Var}[q_i(n+1)] = (1-a)^2s_i(1-s_i)[1-(1-a^2)s_i]^n$$

which monotonically decreases with n and has a limiting value of zero.

Remark. The fact that the limiting value of $\mathrm{Var}[q_i(n)]$ is zero is obvious from Theorem 5, since $q_i(n)$ is a nonnegative random variable whose limiting mean is zero. However, the fact that it *monotonically decreases* to this limiting value is not necessarily obvious, and this is the main contribution of Theorem 6.

Summarizing the results of the above theorems, we note that given two elements R_j and R_k, the probabilities q_j and q_k converge with probability 1 to zero. Further, at any instant,

$$E[q_j(n)] > E[q_k(n)] \quad \text{if and only if } s_j < s_k.$$

Observe that this is true for all $0 < a < 1$. By appropriately choosing the value of the updating constant "a", we shall now show that a stochastically *stronger* inequality exists, which not merely relates the *expected values* of q_j and q_k but the probabilities themselves.

THEOREM 7. *For all j, k where $j \neq k$, if $s_j > s_k$, then the quantity* $\Pr[q_j(n) < q_k(n)]$ *can be made as close to unity as desired.*

Proof. Assume with no loss of generality that $s_j > s_k$. Let

$$x_{j,k}(n) = q_j(n)/q_k(n).$$

Clearly, $x_{j,k}(0) = 1$. Further, $x_{j,k}(n)$ can only assume nonnegative values. Consider the distribution of $x_{j,k}(n+1)$ given the values of $q_j(n)$ and $q_k(n)$. By virtue of (3.1),

$$\begin{aligned} x_{j,k}(n+1) &= (aq_j)/q_k && \text{with probability } s_j, \\ &= q_j/(aq_k) && \text{with probability } s_k, \\ &= q_j/q_k && \text{with probability } (1-s_j-s_k). \end{aligned}$$

Thus,

$$\begin{aligned} x_{j,k}(n+1) &= ax_{j,k}(n) && \text{with probability } s_j, \\ &= x_{j,k}(n)/a && \text{with probability } s_k, \\ &= x_{j,k}(n) && \text{with probability } (1-s_j-s_k). \end{aligned} \tag{3.8}$$

Taking conditional expectations yields

$$E[x_{j,k}(n+1)|x_{j,k}(n)] = 1/a[a^2 s_j + a(1-s_j-s_k) + s_k]x_{j,k}(n).$$

Whence on taking expectation again and observing that $x_{j,k}(0) = 1$, we get

$$E[x_{j,k}(n)] = (h_{j,k})^n \quad \text{where } h_{j,k} = 1/a[a^2 s_j + a(1-s_j-s_k) + s_k]. \tag{3.9}$$

Let a be any real number satisfying $s_k < as_j$. Then

$$\begin{aligned} s_k/a < s_j &\Rightarrow s_k(1-a)/a < s_j(1-a) \\ &\Rightarrow s_k(1/a-1) < s_j(1-a) \\ &\Rightarrow as_j - s_j - s_k + (s_k/a) < 0 \\ &\Rightarrow h_{j,k} < 1. \end{aligned}$$

Thus, $h_{j,k}$ can be made strictly less than unity by appropriately choosing "a." From (3.9), this implies that $E[x_{j,k}(n)]$ can be rendered monotonically *decreasing* with n, and further,

$$\lim_{n\to\infty} E[x_{j,k}(n)] = 0.$$

But $x_{j,k}(n)$ is always nonnegative. This implies that

$$\lim_{n\to\infty} x_{j,k}(n) \to 0 \quad \text{with probability 1.}$$

Thus,

$$\lim \Pr[x_{j,k}(n) > 0] \to 0 \quad \text{with probability 1,}$$

and the result follows.

Remark. Observe that although the values of $\{s_i\}$ are unknown, the above theorem says that by making "a" sufficiently close to unity the asymptotic value $\Pr[q_j(n) < q_k(n)]$ can be made as close to unity as desired.

We now prove the optimality of the scheme.

Let ${}_iY_j(n)$ be the expected probability that R_i succeeds R_j at the nth time instants. Clearly,

$${}_jY_i(n) = 1 - {}_iY_j(n) \quad \text{for all } i, j = 1, \cdots, N, \quad i \neq j.$$

The properties of ${}_iY_j(n)$ are summarized by the following theorem.

THEOREM 8. *The asymptotic value of ${}_iY_j(n)$ is unity if and only if $s_i < s_j$.*

Proof. The theorem follows in a manner analogous to that of Theorem 1.

Note that R_i succeeds R_j at time "$n+1$" if and only if:

(a) R_i succeeded R_j at time n and no list operation was performed, or

(b) R_i was accessed at time n and it was moved to the *rear* of the list.

Let ${}_iy_j = \text{Prob}[R_i \text{ succeeds } R_j \text{ at time "}n\text{"}]$. Clearly, $E[{}_iy_j(n)] = {}_iY_j(n)$. We thus have the following recursive definition of ${}_iy_j(n)$:

$$\begin{aligned} {}_iy_j(n+1) &= 0 && \text{if } R_j \text{ was accessed and MTR is performed,} \\ &= 1 && \text{if } R_i \text{ was accessed and MTR is performed,} \\ &= {}_iy_j(n) && \text{otherwise.} \end{aligned}$$

Let Z_i be the number of times R_i was accessed prior to and including the time instant "n." Clearly, Z_i is a random variable, with $\sum_{i=1}^{N} Z_i = n$. Further, the probability of moving R_i at the nth time instant (on being accessed) to the rear of the list is exactly $a^{Z_i}i^{-1}$. Thus, we rewrite ${}_iy_j(n)$ to obey the following equations:

$$\begin{aligned} {}_iy_j(n+1) &= 0 && \text{with probability } s_j \cdot a^{Z_j - 1}, \\ &= 1 && \text{with probability } s_i a^{Z_i - 1}, \\ &= {}_iy_j && \text{otherwise.} \end{aligned}$$

Taking expectations twice and observing that $E[{}_iy_j(n)] = {}_iY_j$, we obtain

$$\begin{bmatrix} {}_iY_j(n+1) \\ {}_jY_i(n+1) \end{bmatrix} = \begin{bmatrix} 1 - a^{Z_j - 1}s_j & a^{Z_j - 1}s_j \\ a^{Z_i - 1}s_i & 1 - a^{Z_i - 1}s_i \end{bmatrix}^T \begin{bmatrix} {}_iY_j(n) \\ {}_jY_i(n) \end{bmatrix}.$$

Unlike the case of Theorem 2, this time varying difference equation cannot be solved directly using the theory of diagonalization. This is because there is no single constant matrix which diagonalizes the Markov matrix for every time instant. However, we consider the asymptotic value of ${}_iY_j(n)$. On converging, we have

$${}_iY_j(\infty) = a^{Z_i - 1}s_i + (1 - a^{Z_i - 1}s_i - a^{Z_j - 1}s_j)\,{}_iY_j(\infty)$$

which yields

$${}_iY_j(\infty) = \lim_{Z_i, Z_j \to \infty} [a^{Z_i - 1} \cdot s_i / (a^{Z_i - 1} \cdot s_i + a^{Z_j - 1} \cdot s_j)] = \lim_{Z_i, Z_j \to \infty} \left[\frac{a^{Z_i} s_i}{a^{Z_i} s_i + a^{Z_j} s_j} \right].$$

Let $t = s_j / s_i$. Further, using the law of large numbers,

$$Z_j = \frac{s_j}{s_i} \cdot Z_i = tZ_i.$$

Thus,

$$ {}_iY_j(\infty) = \lim_{Z_i \to \infty} \left[\frac{1}{1 + ta^{(t-1)Z_i}} \right]. $$

Now, since we assume that $s_i < s_j$, t is strictly greater than unity. Therefore,

$$ \lim_{Z_i \to \infty} a^{(t-1)Z_i} = 0 $$

and hence, $\lim_{Z_i \to \infty} {}_iY_j(\infty) = 1$. Hence the theorem!

Remark. Observe that in this case too, ${}_iY_j(\infty)$ has a value of 0.5 if $s_i = s_j$, which is what we would expect.

A natural consequence of the above theorem is the corollary stated below. The corollary is obvious inasmuch as the above theorems are valid for every arbitrary pair of records R_i and R_j.

COROLLARY 8.1. *The probability of the list converging to the optimal arrangement (out of the $N!$ possible arrangements) can be made as close to unity as desired.*

Remarks. (1) Just as in the case of the stochastic MTF scheme, observe that we have assumed that the quantities $q_j(n)$ can be arbitrarily small positive real numbers. However, in practice, just as in the above case, it is sufficient to retain indices which remember the number of times a record is accessed. From these indices the quantities $q_j(n)$ can be trivially computed. Observe too that the maximum integer a memory location should contain need not exceed $n_{\max}$, where

$$ n_{\max} = \lceil \log_a x_{\min} \rceil + 1 $$

where $x_{\min}$ is the smallest positive real number yielded by the random number generator accessible to the user.

(2) In this connection, it is beneficial to compare the scheme we have proposed with the scheme which keeps the records *sorted* in the order of the number of times they have been accessed. First of all, observe that our scheme does not require any "global" comparisons, as for example, comparing $q_i(n)$ with $q_j(n)$. Further consider the case when two records R_i and R_j have exactly the same access probabilities. If Z_i and Z_j, respectively, are the number of times these records are accessed, then notice that if one keeps the list sorted on the basis of counters,

$$ \Pr[\text{Move Operation Performed}][Z_i = Z_j;\ R_j \text{ succeeds } R_i;\ R_j \text{ accessed}] = 1. $$

Notice, too, that this probability will be unity *even in the limit.* However, the latter probability will tend to *zero* in the stochastic MTR scheme which we have proposed essentially because of the absorbing Markovian representation of our scheme. One can easily extrapolate this situation to the case when many records have been accessed exactly the same number of times.

Conclusions. We have considered a list of elements $\{R_1, \cdots, R_N\}$ in which the element R_i is accessed with a probability s_i, which is unknown a priori. Two stochastic list organizing schemes have been proposed both of which have absorbing Markovian representations. The first of these schemes requires bounded memory and is of a move-to-front flavour. It is expedient, but is always less optimal than the deterministic move-to-front algorithm. The second scheme performs a move-to-rear operation on the accessed element R_j with a probability q_j which is systematically decreased. Ultimately, the list gets absorbed into one of the $N!$ possible arrangements. We have shown that the asymptotic probability of converging to the optimal arrangement can be made as close to unity as desired.

Acknowledgments. I am greatly indebted to my colleague Nicola Santoro who introduced me to the problem and to the literature in the field. I am grateful to him and my colleague Mike Atkinson for their encouragement and comments during the course of the study. I am also grateful to Professor Ian Munroe, from the University of Waterloo, who first conjectured, while in a personal conversation, the result that the stochastic MTF scheme was at its best expedient.

REFERENCES

[1] D. M. ARNOW AND A. M. TENENBAUM, *An investigation of the Move-Ahead-k rules*, Congressus Numerantium, Proc. 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida, February 1982, pp. 47-65.

[2] J. R. BITNER, *Heuristics that dynamically organize data structures*, this Journal, 8 (1979), pp. 82-110.

[3] P. J. BURVILLE AND J. F. C. KINGMAN, *On a model for storage and search*, J. Appl. Probab., 10 (1973), pp. 697-701.

[4] C. R. COOK AND D. J. KIM, *Best sorting algorithm for nearly sorted lists*, Comm. ACM, 23 (1980), pp. 620-624.

[5] E. W. DIJKSTRA, *Smoothsort, an algorithm for sorting in* SITU, in Science of Computer Programming, North-Holland, Amsterdam-New York, 1982, pp. 223-233.

[6] G. H. GONNET, J. I. MUNRO AND H. SUWANDA, *Exegesis of self organizing linear search*, this Journal, 10 (1981), pp. 613-637.

[7] W. J. HENDRICKS, *The stationary distribution of an interesting Markov chain*, J. Appl. Probab., 9 (1972), pp. 231-233.

[8] ———, *An extension of a theorem concerning an interesting Markov chain*, J. Appl. Probab., 10 (1973), pp. 231-233.

[9] Y. C. KAN AND S. M. ROSS, *Optimal list order under partial memory constraints*, J. Appl. Probab., 17 (1980), pp. 1004-1015.

[10] KARLIN AND H. M. TAYLOR, *A First Course in Stochastic Processes*, Academic Press, New York, 1975.

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

[12] J. MCCABE, *On serial files with relocatable records*, Oper. Res., 12 (1965), pp. 609-618.

[13] R. L. RIVEST, *On self-organizing sequential search heuristics*, Comm. ACM, 19 (1976), pp. 63-67.

[14] D. SLEATOR AND R. TARJAN, *Amortized efficiency of list update rules*, Proc. 16th Annual ACM Symposium on the Theory of Computing, April 1984, pp. 488-492.

[15] A. M. TENENBAUM AND R. M. NEMES, *Two spectra of self-organizing sequential search algorithms*, this Journal, 11 (1982), pp. 557-566.

[16] B. J. OOMMEN, *On the use of Smoothsort and stochastic move-to-front operations for optimal list organization*, Proc. 22nd Allerton Conference on Communication, Control and Computing, October 1984, pp. 243-252.

[17] E. C. TITCHMARSH, *The Theory of Functions*, Oxford University Press, London, 1964.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

© 1987 Society for Industrial and Applied Mathematics
008

GENERATING RANDOM UNLABELLED GRAPHS*

NICHOLAS C. WORMALD†

Abstract. Unlabelled graphs on n vertices can be generated uniformly at random, without calculating the total numbers of such graphs, but by using asymptotic enumeration results. For large n, the process is very efficient, taking $O(n^2)$ steps on average. In fact, for $n \geqq 50$, virtually all practical applications will require only the first step of the algorithm to be implemented. This step is almost as simple as, and has the appearance of, the random generation of labelled graphs, yet the net effect is the uniform generation of unlabelled graphs at random.

By similar methods one can generate random unlabelled graphs with n vertices and m edges, in expected time $O(n^2\sqrt{m})$ for most values of m. Also, random unlabelled r-regular graphs can be generated in expected time $O(n)$ when $r \geqq 3$ is fixed, but this is only (so far) practicable for a few small values of r.

Key words. random graph, uniform generation, enumeration, generation algorithm, unlabelled graph

AMS(MOS) subject classifications. 68R10, 05C30

1. Introduction. To study typical configurations, or to evaluate the performance of an algorithm on such configurations, it is often desirable to be able to generate such configurations uniformly at random (u.a.r.). For some configurations, such as labelled graphs on n vertices, this is simple to do: one chooses the edges independently with probability $\frac{1}{2}$ for each. This paper is concerned primarily with unlabelled graphs, but the general method is developed explicitly in terms of generating a random orbit of an arbitrary permutation group. The central idea is the adaptation of asymptotic enumeration methods and results to random generation problems. This will be shown to be fruitful for several graph generation problems.

Whenever a counting formula has been found for a set of configurations, there is a reasonable chance that the method of deriving the formula will lead to an algorithm for generating the configurations u.a.r. For example, this was done for unlabelled graphs by Dixon and Wilf [3] and for labelled regular graphs in [8]. The resulting algorithm can be rather complex. However, if there is a suitable method of counting the configurations asymptotically, this can also lead to a generation procedure. Typically, the latter procedure consists of a trial which may fail or succeed, and needs to be repeated until successful. This was also done for labelled regular graphs [8].

The method of converting an asymptotic enumeration method into a generation method will vary, but it appears that the following idea will apply in many cases. The following discussion is phrased in terms of graphs but applies equally well to other configurations. Suppose that the asymptotic result is obtained by proving that almost all graphs of type A are of type B, where graphs of type B are easy to count and easy to generate u.a.r. (For instance, almost all triangle-free graphs are bipartite [4], and almost all chordal graphs are split graphs [1], although it is not claimed that in either of these cases, the random generation algorithm is easy or even feasible to find. Note that if it is graphs of type A which are easy to generate u.a.r., then rejection of the graph if it is not of type B produces type B graphs u.a.r. in asymptotically the same expected time. For instance, this applies to generating k-connected graphs u.a.r. since almost all graphs are k-connected.) The method of the proof is often equivalent to an algorithm for generating all graphs of type A which are not of type B, such that the number of possible outcomes of the algorithm is far less than the number, say b, of

* Received by the editors October 21, 1985; accepted for publication (in revised form) October 15, 1986.
† Department of Mathematics and Statistics, University of Auckland, Auckland, New Zealand.

graphs of type B. Such a generation algorithm will lead naturally to a random generation algorithm, by performing one possibility for each stage of the algorithm at random, rather than running through all possibilities. The probability of any given graph occurring can hopefully be calculated a posteriori (the ease of this depends on the particular algorithm). If one knows a lower bound, say p, on the least probability, then all probabilities can be equalised using an accept/reject procedure. We can then generate graphs u.a.r. by choosing between type B (with probability $b/(b+p^{-1})$) and the others (with probability $p^{-1}/(b+p^{-1})$). Since the asymptotic enumeration result proves that almost all graphs of type A are of type B, one can hope that p^{-1} is far less than b, although achieving this may require some careful management in obtaining the random generation algorithm. Then the graphs of type B are usually chosen by the algorithm, and the overall generation algorithm can be expected to run quickly on the average, provided it was not found necessary to sacrifice efficiency too much in order to keep p from becoming too small, and provided the a posteriori probabilities are calculated easily.

Obtaining a suitable implementation of the general idea described above will seldom be automatic, and the rest of this paper is concerned with three applications, for various types of unlabelled graphs. In these applications, the structures of type A are ordered pairs consisting of a labelled graph of a certain type together with an automorphism of that graph, and those of type B are the ordered pairs in which the automorphism is the identity.

Dixon and Wilf [3] gave a solution, described in § 2 below, to the random orbit problem, based on an exact enumeration result. They then applied this to generating random unlabelled graphs. The related alternative solution to the random orbit problem given in § 2 is a variation on theirs which permits easy implementation of the "asymptotic" approach. It is shown in § 3 how to apply this to the generation of unlabelled graphs u.a.r., by continually repeating a trial which has a fixed probability of succeeding. The advantage this has over the algorithm in [3] is discussed. In particular, when n is large (say $n \geqq 40$), graphs with nontrivial automorphisms are very rare. As a result, the present algorithm can be organised in such a way that probably only the first simple part will ever be used, in the generation of a large number of unlabelled graphs. Thus, for practical purposes, only the first part should be programmed! It is not surprising that this first part appears to generate labelled graphs u.a.r. (see Subalgorithm Quick Graph).

An *(n, m)-graph* is a graph with n vertices and m edges. Dixon and Wilf leave open the problem of efficiently generating unlabelled (n, m)-graphs u.a.r. This question is addressed in § 4, where a solution is given which works very quickly on average for most values of m as long as n is fairly large, and remains useful even for some values of m for which many (n, m)-graphs do have nontrivial symmetries. The probability space of labelled n-graphs with edges occurring independently with probability p is used, and the arguments given here are derived from those used by Wright [10], [11] to find the asymptotic number of unlabelled (n, m)-graphs, for various m. In § 5, the proof given in [9], that almost all unlabelled r-regular graphs are asymmetric (fixed $r \geqq 3$), is adapted to generate these graphs u.a.r. It is rather curious that the expected time taken by the resulting algorithm, per graph generated, actually decreases as the size of the graph increases, up to a point.

2. Random orbits. This section begins with a derivation of Algorithm Random Orbit of [3], since much of the background and terminology will be required. The Frobenius-Burnside Lemma gives the number, Q, of orbits of a finite permutation

group G acting on a set T as

$$Q = \frac{1}{|G|} \sum_{\sigma \in G} |\text{Fix}(\sigma)| \tag{2.1}$$

where Fix (σ) is the set of elements of T fixed by the action of σ. (The action of σ is a permutation of T induced by σ, and the set of actions of elements of G form a group isomorphic to G.) Application of this result to a single orbit shows that there are just $|G|$ representatives from that orbit appearing among all the elements of the sets Fix (σ), $\sigma \in G$, i.e., among the second elements of the set W of ordered pairs given by $W = \{(\sigma, L): \sigma \in G, L \in \text{Fix}(\sigma)\}$. Thus, choosing u.a.r. an element (σ, L) of W produces an orbit of G (the orbit containing L) u.a.r. A random element of W can be chosen by first selecting σ with probability proportional to $|\text{Fix}(\sigma)|$, and second choosing $L \in \text{Fix}(\sigma)$ u.a.r. The number of elements of an orbit of G fixed by σ depends only on the conjugacy class of σ. So, instead of working with all possible elements σ of G, we can restrict ourselves to a set of representatives $\sigma_1, \cdots, \sigma_a$ of the conjugacy classes $C_1, \cdots, C_a$ of G, and select σ_i with probability proportional to $|C_i||\text{Fix}(\sigma_i)|$. This yields the following algorithm.

ALGORITHM RANDOM ORBIT I (Dixon and Wilf [3]).

(Given a permutation group G acting on a set, this algorithm chooses an orbit U u.a.r. from the set of all orbits of the action of G.)

Step 1. Choose a conjugacy class of G so that for $1 \leqq i \leqq a$ the probability of choosing C_i is $|C_i||\text{Fix}(\sigma_i)|/Q|G|$.

Step 2. If C_i was chosen in Step 1, then choose u.a.r. an $L \in \text{Fix}(\sigma_i)$.

Step 3. Return the orbit U containing L.

We wish to consider problems in which Q is difficult to calculate precisely. In some situations, one may wish to abandon carrying out an algorithm and start it afresh. We use *restart* to signify this act, and we may if necessary specify the algorithm being restarted. This leads to an algorithm consisting of a procedure or trial which is repeated indefinitely until normal termination occurs.

Suppose we have a procedure *choosefix* which, for $\sigma \in G$, chooses $L \in \text{Fix}(\sigma)$ with probability $P(L)$. In the applications in §§ 3 and 4, $P(L)$ depends on σ but is independent of L, but that is not so in § 5. We allow the possibility that *choosefix* does not output an L, but restarts the main algorithm instead. It is sometimes convenient to work with subsets of G other than conjugacy classes, so let $R_1, \cdots, R_k$ be sets which partition the elements of G. Suppose that for each i, $1 \leqq i \leqq k$, we have an upper bound B_i on $\{|R_i|/P(L): L \in \text{Fix}(\sigma), \sigma \in R_i\}$. For this it is necessary that $B_i \geqq |R_i||\text{Fix}(\sigma)|$ for each $\sigma \in R_i$, and if $P(L) = 1/\text{Fix}(\sigma)$ then of course this is also sufficient. Put $B = \sum_{i=1}^{k} B_i$. For the next algorithm, there is no need to calculate or even estimate Q further than the upper bound B.

ALGORITHM RANDOM ORBIT II.

(This repeats a trial which, when successful, achieves the same effect as Algorithm Random Orbit I.)

Step 1. Choose one of $R_1, \cdots, R_k$ so that the probability of choosing R_i is B_i/B.

Step 2. Choose $\sigma \in R_i$ u.a.r.

Step 3. Obtain L from *choosefix* (σ).

Step 4. Continue with probability $|R_i|/B_iP(L)$. Otherwise, *restart*.

Step 5. Return the orbit U containing L.

To verify that U is selected u.a.r., it is enough to observe that the probability, when the algorithm terminates, that a given σ and a given $L \in \text{Fix}(\sigma)$ were chosen is

$1/B$. Hence, in effect an element of W is chosen u.a.r. It follows also that in any one trial of the algorithm, the probability of finding U (with no *restart*) is $|G|Q/B$.

Note that if $P(L)$ is the same for each $L \in \text{Fix}(\sigma)$, i.e. *choosefix* (σ) selects L u.a.r., then Step 4 can be moved above Step 3. If, moreover, $P(L)$ is independent of σ for those $\sigma \in R_i$, then Step 4 can be moved to between Step 1 and Step 2. If R_i is a conjugacy class then Step 2 can be omitted.

When the magnitudes of the B_i vary considerably, it is advantageous to lump the smaller ones together in order to speed up Step 1. In this way, difficulties in calculating B can possibly be avoided. This is especially useful if k is large. For instance, suppose that for $1 \leqq i \leqq k$ we know an upper bound A_i on $\sum_{j=i}^{k} B_j$ with $A_i + B_{i-1} \leqq A_{i+1}$, and set $A_{k+1} = 0$. Then Step 1 of the previous algorithm can be implemented by the following.

ALGORITHM WEIGHTED CHOICE.

For $i = 1$ to k do:

With probability B_i/A_i, exit with R_i. With probability $1-(A_{i+1}+B_i)/A_i$, *restart* Algorithm Random Orbit II. Otherwise (i.e., with probability A_{i+1}/A_i) increment i.

End

In order to minimise expected run time, the general aim is to make the A_i easy to calculate and to choose the B_i to be fairly tight bounds, with, as far as possible, B_i decreasing with i. Note that if B_1/A_1 is almost 1, the other B_i and A_i will usually not even have to be calculated.

3. Random n-graphs. Let T be the set of all labelled graphs on n vertices and let G be the symmetric group S_n acting on T, so that the action of $\sigma \in S_n$ is a permutation $\rho(\sigma)$ defined as follows. For each labelled n-graph L, $(\rho(\sigma))(L)$ is the result of permuting the vertices of L according to σ. Let σ^* denote the permutation of the edges of the complete graph K_n induced by σ. Then L is fixed by $\rho(\sigma)$ (i.e. $L \in \text{Fix}(\sigma)$) if and only if the edge set of L is the union of orbits of σ^*. Thus $|\text{Fix}(\sigma)| = 2^{f(\sigma)}$, where $f(\sigma)$ is the number of orbits of σ^*. Consequently, for random unlabelled n-graphs we implement *choosefix* (σ) by choosing independently at random the orbits of σ^* to be filled, each with probability 0.5, giving $P(L) = 1/2^{f(\sigma)}$ for all $L \in \text{Fix}(\sigma)$. Hence, we can perform Step 4 of Algorithm Random Orbit II immediately after Step 2. Note that $|S_n| = n!$ Also, the conjugacy class of σ depends only on the partition of n induced by the cardinalities of the orbits of σ (i.e., the lengths of the cycles in the disjoint cycle decomposition of σ). So if n is fairly small, we can choose $\sigma_1, \cdots, \sigma_k$ to correspond to the partitions of n, and then Algorithm Random Orbit I (or II, with $R_i = C_i$ and $B_i = 2^{f(\sigma_i)}|C_i|$) generates random unlabelled graphs. This is essentially the algorithm given in [3]. The following modification is proposed, using Algorithm Random Orbit II.

An orbit (of a permutation) of cardinality i is called an *i-cycle*. Let $R_1 = \{\text{id}\}$, where id is the identity permutation in S_n, and for $i \geqq 2$ let R_i be the set of elements of S_n having support of cardinality i, i.e., moving precisely i members of $\{1, \cdots, n\}$. We use $[n]_i$ to denote $n!/(n-i)!$

LEMMA 3.1. *Let R_i be defined as above, and put*

$$B_1 = 2^N, \qquad B_i = 2^{N-H(n,i)}[n]_i \quad (2 \leqq i \leqq n)$$

where $N = \binom{n}{2}$ and $H(n, i) = i(n/2-(i+2)/4)$. With these parameters Algorithm Random Orbit II *generates unlabelled n-graphs u.a.r. in time $O(n^2)$ per graph.*

Proof. We have $|R_i| \leqq [n]_i$. Also, if $\sigma \in R_i$ and $i \geqq 2$ then the number of i-cycles of σ^* is at most

$$\binom{n-i}{2} + i/2 = N - 2H(n, i). \tag{3.1}$$

Hence the number of orbits of σ^* is at most $N-H(n,i)$ and so $|\text{Fix}(\sigma)| \leqq 2^{N-H(n,i)}$ Also $|\text{Fix}(\text{id})| = 2^N$. Thus Algorithm Random Orbit II now generates unlabelled n-graphs u.a.r. For large n, the expected time taken to generate each graph is approximately equal to the time taken to write the graph down ($O(n^2)$). This is because B_1 is much larger than the other B_i. It is fairly irrelevant how Step 2 is performed for large n, since almost always R_i will be R_1 and *id* can be immediately selected as σ. In general, for $i \geqq 2$ Step 2 can be implemented by choosing random permutations of $\{1, \cdots, i\}$ until a derangement of that set is found. The expected number of trials of this procedure is at most 3, when $i = 3$ (see Liu [6] for relevant information). Strictly, to choose σ u.a.r, one should first choose a random i-subset for its support, but since different supports lead to conjugate permutations, this part can be omitted. This establishes the lemma.

Even for small n, this algorithm is reasonably fast. Since there is no possibility of a *restart* in Step 4 when R_1 is chosen in Step 1, the expected number of *restarts* occurring per graph generated is at most $(B-B_1)/B_1$. This is less than 2 for $n \geqq 7$.

The reason for preferring this implementation of Algorithm Random Orbit II to Dixon's and Wilf's algorithm is that the latter involves the preliminary computation of the number of unlabelled n-vertex graphs (or n-graphs). This is feasible only as far as $n = 40$ or 50 by present methods and requires roughly $e^{c\sqrt{n}}$ steps for some constant c. To overcome this, a modification is given in [3] which is computationally much more useful for large n but does not quite generate graphs u.a.r. Instead, one first chooses an $\varepsilon > 0$ such that one is satisfied if the probabilities of various graphs occurring are all within a factor of $1+\varepsilon$ of each other, and one calculates numbers of graphs only approximately. This suffices for practical purposes, but requires a careful implementation to ensure that graphs such as the complete graph, with many automorphisms, occur with about the same probability as graphs with only the identity automorphism. On the other hand, by Lemma 3.1, it is just as easy to use Algorithm Random Orbit II and retain the precise uniformity of the probabilities.

We next consider a modification obtained by using Algorithm Weighted Choice for Step 1. Since $k = n$ is not very large, there is no great gain to be had in the speed of execution in this case. However, if we are interested only in fairly large graphs, with perhaps 40 or more vertices, we probably only need to implement the following "subalgorithm," thereby saving programming effort. It may appear to be a probabilistic algorithm which, if it succeeds, generates a labelled n-graph uniformly at random. However, in the event of failure, the probabilities can be adjusted so that the net result is the uniform generation of unlabelled graphs, as shown by Lemma 3.2.

SUBALGORITHM QUICK GRAPH.

(If properly attended to in the event that Step B is reached, this generates an unlabelled n-graph U u.a.r. for $n \geqq 15$.)

Step 1. With probability $2^{n-1}/(3n^2+2^{n-1})$, do Step A. Otherwise (i.e., with probability $3n^2/(3n^2+2^{n-1})$) do Step B.

Step A. Generate L u.a.r. from the labelled n-graphs. "Ignore" the labels of L to obtain U. Exit with U.

Step B. Write "The rest of the algorithm needs to be programmed."

LEMMA 3.2. *For $n \geqq 15$ Steps* 1 *and* A *of Subalgorithm Quick Graph are the first part of an algorithm which generates an unlabelled n-graph U u.a.r.*

Proof. Define R_i and B_i as for Lemma 3.1, and set $l = n - i$. Then for $i \geqq 2$, $B_i = B_n 2^{l(l-1)/4}/l!$, and so $B_{i+1}/B_i = l2^{(1-l)/2}$. Straightforward checking now shows $B_i > \sum_{j=i+1}^{n} B_j$ for $n-1 > l \geqq 13$. Thus, if we set $A_i = B_i + 2B_{i+1}$ for $i < n-13$ and $A_i = \sum_{j=i}^{n} B_j$ otherwise, Lemma 3.1 implies that Random Orbit II, using Weighted Choice in its

Step 1, generates unlabelled n-graphs u.a.r. For $n \geqq 15$, $A_1 = 2^N(1+2n(n-1)/2^{n-5/2}) < 2^N(1+3n^2/2^{n-1})$, and so the latter can be used instead for A_1. We now have $B_1/A_1 = 2^{n-1}/(3n^2+2^{n-1})$. Hence, this algorithm is equivalent to performing Quick Graph, provided that if Step B of Quick Graph is reached, we *restart* Quick Graph with probability $(A_1-A_2-B_1)/(A_1-B_1)$ and otherwise enter Random Orbit II, with Weighted Choice used for its first step, skipping the case $i=1$ in Weighted Choice.

The point of Subalgorithm Quick Graph is that for large n, it can be used many times over without (probably) ever reaching Step B, and so we have only a short program to encode. For instance, for $n=40$ the probability of needing Step B in one run of the algorithm is roughly 10^{-8}, and so, with probability $\frac{1}{2}$, approximately 8×10^7 runs can be performed without ever needing Step B. For $n=50$ the latter number becomes 5×10^{10}. Thus, for such values of n, Quick Graph will suffice for practical purposes.

4. Random (n, m)-graphs. Direct application of Algorithm Random Orbit I (or II) to the problem of generating random unlabelled (n, m)-graphs meets with an obstacle in Step 2 (or Step 3 of II). Another problem is the calculation of Q for I. For II, the analogous problem is the choice of suitable B_i's, and this is not so hard. For most values of m, the problems associated with II can be dealt with by taking a viewpoint slightly different from the obvious one. We use the probability space $\mathcal{G}(n, p)$ of labelled n-graphs in which the edges are chosen independently at random with probability p, and write $q=1-p$. Let G be S_n as in § 3, let T be the set of labelled (n, m)-graphs, and let $F(\sigma)$ be the set of n-graphs fixed by the action of σ. Then $\mathrm{Fix}\,(\sigma) = T \cap F(\sigma)$. To choose $L \in \mathrm{Fix}\,(\sigma)$ u.a.r., we can select a random element L of $\mathcal{G}(n, p) \cap F(\sigma)$ (i.e., select $L \in F(\sigma)$ with probability proportional to the probability of L occurring in $\mathcal{G}(n, p)$). If L does not have m edges, we *restart* Algorithm Random Orbit II. Each $L \in \mathrm{Fix}\,(\alpha)$ is equally likely to occur because the probability of $L \in T$ occurring in $\mathcal{G}(n, p)$ is $p^m q^{N-m}$ ($N = \binom{n}{2}$ as before), independent of L. Thus, we implement *choosefix* as follows:

For each orbit D of σ^* do:
If i is the number of edges in D(i.e., D is an i-cycle), place all these edges in L with probability $p^i/(p^i+q^i)$, and omit them all with probability $q^i/(p^i+q^i)$.
End
If L now has m edges, exit with L. Otherwise *restart* Algorithm Random Orbit II.

What to use for the R_i depends on tradeoffs between time complexity, ease of programming (and presenting) algorithms, and estimates of which values of m and n are to be used. In most situations it will suffice to define the R_i as in § 3, so we take them as such now.

To arrive at suitable bounds B_i, first note that for $L \in T$, the probability $P(L)$ of choosing L in *choosefix* (σ) is $p^m q^{N-m}/\prod_{i=1}^{n}(p^i+q^i)^{h(\sigma,i)}$, where $h(\sigma, i)$ is the number of i-cycles of σ^*. As $p, q \geqq 0$, it is easily shown that for integers $i \geqq 3$, $p^i+q^i \leqq (p^2+q^2)^{i/2}$. Thus $1/P(L) \leqq p^{-m}q^{m-N}(p^2+q^2)^{(N-h(\sigma,1))/2}$. If $\sigma \in R_i$, $i>1$, then using (3.1), we get that $1/P(L)$ is at least

$$p^{-m}q^{m-N}(p^2+q^2)^{H(n,i)}. \tag{4.1}$$

Since this is valid for any p, set $H(n, 1)=0$ and given $i \geqq 1$ let p_i denote a (or the) value of p for which (4.1) is minimised (or any conveniently calculated value near the

minimum). Then, recalling $|R_i| \leqq [n]_i$ for $i \geqq 2$, we set

$$B_1 = p_1^{-m} q_1^{m-N}, \quad B_i = p_i^{-m} q_i^{m-N} (p_i^2 + q_i^2)^{H(n,i)} [n]_i \quad (2 \leqq i \leqq n).$$

Apart from specifying the p_i and q_i, an implementation of Random Orbit II has now been defined, which generates unlabelled (m, n)-graphs u.a.r. In *choosefix* (σ) with $\sigma \in R_i$, we use $p = p_i$ and $q = q_i$. If $r = p/q$, then $r + 1 = q^{-1}$, and the logarithmic derivative of (4.1) is 0 if and only if

$$Nr(r^2+1) - m(r+1)(r^2+1) + 2H(n,i)r(r-1) = 0.$$

It would be straightforward to automate the use of this cubic equation in finding p_i so as to minimise (4.1), at least approximately. However, as long as $\min(m, N-m)$ is reasonably large, such a computation is probably a waste of time. It is clear that $p_1 = m/N$. For most values of m, this will also serve reasonably well as p_i in the algorithm as defined above. In the following two lemmas we take $m \leqq N/2$, since similar results for $m > N/2$ can be obtained by complementing the graphs.

LEMMA 4.1. *With $p_i = p = m/N$ for $1 \leqq i \leqq n$, and with $2c \log n/n < m/N \leqq \frac{1}{2}$ (c constant and $c > 1$), an unlabelled (m, n)-graph is produced u.a.r. in expected time $O(n^2\sqrt{m})$.*

Proof. For $i \geqq 2$, $B_i/B_1 = (1-2p+2p^2)^{H(n,i)}[n]_i$. To analyse this, we often use C to denote a positive constant, perhaps different at each occurrence of the notation. We have $\log(1-2p+2p^2) < -4c \log n/n + C/n$ (by Taylor's Theorem). As $H(n,i) \geqq i(\frac{1}{2}n - 1)/2$, we have $\log(B_i/B_1) < i(C + \log n - c \log n)$. Since $c > 1$ it follows that for n sufficiently large, $B - B_1$ is negligible compared to B_1. Thus for n large, R_1 is almost always chosen in Step 1 of Algorithm Random Orbit II. If R_1 is chosen, the probability of *choosefix* (id) choosing a graph L with m vertices is $\binom{N}{m} p^m q^{N-m} \sim (2\pi m)^{-1/2}$ by Stirling's formula. Hence, the expected number of times that Algorithm Random Orbit II is restarted before being successful in *choosefix* (id) is asymptotically $\sqrt{2\pi m}$. Since *choosefix* takes time $O(n^2)$, the lemma follows.

Incidentally, from the bound on $(B - B_1)/B_1$ in the proof of Lemma 4.1, it follows immediately that for m as in Lemma 4.1 (but $c > 1.5$), the contributions to the sum in (2.1) from $\sigma \neq \mathrm{id}$ are negligible, and hence $Q \sim \binom{N}{m}/n!$

For the values of m considered so far, almost all unlabelled (n, m)-graphs have no nontrivial symmetries (as shown in [10]). For smaller m, we need to decrease B_i for large i, if the algorithm is to be of use. Put $I = n - n^{3/4}(\log n)^{2/3}$ and set

$$p_i = \begin{cases} m/N, & i = 1, \\ m/(N - 2H(n,i)), & 2 \leqq i < I, \\ (m/N)^{1/2}. & I \leqq i \leqq n. \end{cases}$$

From the results of [11], when $m = \frac{1}{2}n \log n + c_n n$, $c_n \to c > 0$, the expected number of automorphisms of a random labelled (m, n)-graph is $\exp(-e^{-2c})/(1 - e^{-2c})$. Even for these values of m, the algorithm can be made reasonably effective, as shown next using a modification of some of the argument in [10].

LEMMA 4.2. *For $\frac{1}{2}n \log n + cn < m < c'n \log n$ ($c > 0$ and $c' > \frac{1}{2}$), use of the above p_i in the algorithm gives an unlabelled (n, m)-graph u.a.r. in expected time $O(n^2\sqrt{m}) = O(n^{5/2}\sqrt{\log n})$.*

Proof. Let H denote $H(n, i)$. For $2 \leqq i \leqq n$,

$$B_i < n^i p_i^{-m} q_i^{m+2H-N} (1 + p_i^2/q_i^2)^H. \tag{4.2}$$

First suppose $i \leqq I$. Since $n - i > n^{3/4}(\log n)^{2/3}$, $N - 2H > Cn^{3/2}(\log n)^{4/3}$ and so $p_i/q_i < Cn \log n/n^{3/2}(\log n)^{4/3} < C/n^{1/2}(\log n)^{1/3}$. Thus $\log(1 + p_i^2/q_i^2) < C/n(\log n)^{2/3}$. Since

$H < ni$ this gives $(1+p_i^2/q_i^2)^H = e^{o(i)}$. Also,

$$p_i^{-m} q_i^{m+2H-N} = m^{-m}(N-2H)^{N-2H}(N-2H-m)^{m+2H-N}$$
$$\leqq C\sqrt{m}\binom{N-2H}{m},$$

by Stirling's formula, using $m < c'n \log n$. Similarly, $B_1 \geqq C\sqrt{m}\binom{N}{m}$ and so by (4.2)

$$B_i/B_1 < e^{o(i)} n^i \binom{N-2H}{m} \Big/ \binom{N}{m}$$
$$< e^{o(i)} n^i ((N-2H)/N)^m.$$

For n sufficiently large,

$$\frac{N-2H}{N} \leqq \left(1-\frac{i}{n}\right)^2 \left(1+\frac{i^2}{(n-1)(n-i)^2}\right)$$
$$\leqq \left(1-\frac{i}{n}\right)^2 \left(1+\frac{i^2}{n^2}\right)$$

and so $\log((N-2H)/N) < -2i/n$. As $m > \frac{1}{2}n \log n + cn$, this gives

$$B_i/B_1 < e^{i(-2c+o(1))} \qquad (2 \leqq i < I). \tag{4.3}$$

Now suppose $I \leqq i \leqq n$. Then $p_i^{-m} q_i^{m-N} = \sqrt{B_1}$, $q_i^{2H} < 1$ and $(1+p_i^2/q_i^2)^H < (1+p_i^2/q_i^2)^N$. But $\log(1+p_i^2/q_i^2) < Cp_i^2 < C \log n/n$, and so from (4.2), $B_i < e^{Cn \log n}\sqrt{B_1}$. As $B_1 > e^{Cn(\log n)^2}$, this yields $B_i/B_1 < e^{-Cn(\log n)}$, and so for i in this range, B_i is negligible. Since $c > 0$, we now have from (4.3) that $B < CB_1$ and so R_1 is chosen in Step 1 of Algorithm Random Orbit II with positive probability. As in Lemma 1, *choosefix* (id) chooses an (n, m)-graph L with probability asymptotic to $(2\pi m)^{-1/2}$, and so the lemma is established.

From the proofs of the lemmas, we see that provided $N/2 > m > \frac{1}{2}n \log n + \alpha(n)n$ where $\alpha(n) \to \infty$ as $n \to \infty$, we can specify p_i (as for Lemma 4.1 if $m > 2n \log n$, say, and otherwise as for Lemma 4.2) so that $B_1/B \to 1$. Hence, for such m and for large n, it may be profitable to consider using Weighted Choice, as it was used in § 3.

5. Random r-regular graphs. A graph is *r-regular* if every vertex has degree r. It has been proved that almost all r-regular n-graphs have no symmetries when $r \geqq 3$ (r fixed) by Bollobás [2], and independently for $3 \leqq r = O(n^{1/2-\varepsilon})$ by McKay and the author [7]. The proofs involved many variables but were simplified in [9], and the latter asymptotic proof is the basis of the implementation of Algorithm Random Orbit II in this section. The practicability of the algorithm is commented on at the end of this section. In what follows, *restart* refers to Random Orbit II unless otherwise specified.

Let T be the set of labelled r-regular n-graphs, $r \geqq 3$, and let $G = S_n$ as in §§ 3 and 4. A suitable procedure *choosefix* is obtained directly from the proof in [9] as follows. If, at any stage, the graph L being generated gets a vertex of degree greater than r, then *restart*. Let D denote the support of σ, put $a = |D|$ and $D' = \{1, 2, \cdots, n\} - D$.

Step 1. Choose each of the "diagonals" of 2-cycles of σ (these edges are "flipped" by σ^*) to be in L independently with probability $\frac{1}{2}$. Let e_1 denote the number of these actually chosen.

Step 2. Choose a number f u.a.r. from 0 to $ra/4$, and then choose f orbits of σ^* of cardinality at least 2 between vertices in D, to be filled by edges of L. This can be

done by choosing u.a.r. an f-subset of the $a(a-1)/2$ edges joining vertices in D, and using the orbits of σ^* determined by these edges (*restart* if any two of the f edges are in a common orbit). Let k denote the number of edges of L chosen so far.

Step 3. Choose the edges in L from D to D' by repeating the process of taking any vertex u in D with degree less than r and choosing a vertex $v \in D'$ u.a.r., then adding to L all edges in the orbit of σ^* containing uv.

Step 4. Choose the subgraph of L induced by D' u.a.r. from those with the degree sequence now required to make L r-regular (as determined by the edges from D to D'). This can be generated by Rangraph, given in [8]. That involves successive repetitions of a trial which, if successful, generates a graph with given degree sequence u.a.r. To simplify the probability computations, we use that trial only once here. If it is unsuccessful, then *restart*.

THEOREM 5.1. *Let $r \geqq 3$ be fixed. With the above implementation of choosefix, Random Orbit* II *can be used to generate an unlabelled r-regular n-graph u.a.r. in expected time $O(n)$.*

Proof. The sets R_i and the bounds B_i will be defined after calculating a lower bound for $P(L)$. In the following, most of the probabilities mentioned are implicitly conditional probabilities. Let $s(i) = s_i$ denote the number of i-cycles of σ ($i \geqq 1$). The probability of a given outcome of Step 1 in *choosefix* is $2^{-s(2)} \geqq 2^{-a/2}$. In Step 2, a given f is chosen with probability $(1+ra/4)^{-1} \geqq 2/ra$ (provided $a \geqq 2$), and the probability of a given f-subset (and hence of a given set of orbits of σ^*) being chosen is at least

$$\binom{a^2}{f}^{-1} \geqq \frac{f!}{a^{2f}}.$$

There are $ra-2k$ edges of L to be chosen in Step 3, and each orbit of σ^* involving these edges has cardinality at least 2. Thus, the probability of any given outcome of Step 3 is at least $n^{-(ra-2k)/2}$ (the order in which the edges are added is unimportant). In Step 4 we choose a graph with $rn/2 - ra + k$ edges and $r-a$ vertices, with given degrees, say $d_1, \cdots, d_{n-a}$. All of the d_i are equal to r, except for deficiencies due to the edges placed in Step 3. The total of the deficiencies is $ra-2k$. The probability of a given such graph occurring in the trial in Rangraph is exactly

$$\frac{g!2^g \prod d_i!}{(2g)!} \geqq \frac{g!2^g (r!)^{n-a}}{(2g)!\, r^{ra-2k}}$$

where $g = rn/2 - ra + k$.

Thus, we may set $R_1 = \{\mathrm{id}\}$ and $B_1 = (rn)!/(rn/2)!\, 2^{rn/2} (r!)^n$, and for any other σ,

$$P(L)^{-1} \leqq \frac{2^{a/2-1} ra^{2f+1} (r^2 n)^{ra/2-k} (2g)!}{f!\,(r!)^{n-a} 2^g g!}. \tag{5.1}$$

Note that it is simple enough to calculate $P(L)$ after L has been selected, for use in Step 4 of Random Orbit II. Unfortunately, the definition of the other sets R_i is more complicated than in previous sections. This time, we need a different set R_i to contain those σ with any given set of values of a, s_2, s_3, s_4, s_6. Denote this R_i by $R(a, s_2, s_3, s_4, s_6)$, or R for short. Then

$$|R| \leqq \frac{\binom{n}{a} a!}{\prod s_i!} \leqq \frac{n^a a!}{\prod s_i!} \tag{5.2}$$

where the products are over $i = 2, 3, 4$ and 6.

In [9] is an argument which shows that apart from a factor of a, which makes no essential difference, the product of (5.1) and (5.2), when divided by B_1, is less than $D'n^{-Da}$ for some positive constants D and D' which depend only on r. By keeping track of the constants in that argument, one can assign values to D and D' and so obtain

$$B_i = D'n^{-Da}B_1 \tag{5.3}$$

when $R = R_i = R(a, s_2, s_3, s_4, s_6)$. The evaluation of the appropriate constants is straightforward, but requires inequalities such as $n^n e^{-n}\sqrt{2\pi n} < n! < n^n e^{-n}\sqrt{2\pi n}\, e^{1/12}$ (see Erdös and Spencer [5, p. 17]).

To implement Step 2 of Random Orbit II, i.e., to choose $\sigma \in R$ u.a.r., choose s_i disjoint i-cycles for σ at random ($i = 2, 3, 4$ and 6), and then choose a random permutation of the remaining members of $\{1, 2, \cdots, n\}$. Then *restart* Step 2 if the latter permutation has any cycles of lengths 1, 2, 3, 4 or 6. The expected number of *restarts* required is bounded, unless $a - (2s_2 + 3s_3 + 4s_4 + 6s_6) \in \{1, 2, 3, 4, 6\}$ in which case this set of parameters is impossible and should not be considered.

Since Rangraph takes $O(n)$ time (by [8, Thm. 4.3]) this implementation of Random Orbit II takes $O(n)$ time for each restart. Since each s_i is at most a/i, (5.3) gives

$$\sum B_i \leqq a^4 D' n^{-Da} B_1$$

where the sum is over all i for which R_i contains permutations σ moving a vertices. Hence, the sum of all the B_i is $O(B_1)$, and so the probability of choosing B_1 in Step 1 of Random Orbit II is bounded below by a positive constant. For r fixed, the probability that the trial in Rangraph succeeds is similarly bounded below by a positive constant, and hence so is the probability that Random Orbit succeeds, for each *restart*. It is fairly easy to implement Weighted Choice so that Step 1 of Random Orbit II requires $O(1)$ time on average. The theorem follows.

From [9] it is seen that the proof of Theorem 5.1 becomes considerably simpler for $r = 4$ (when s_3 and s_6 are not required as parameters) or $r \geqq 5$ (when none of the s_i are required). Unfortunately, for $r = 3$, and for $r = 4$ to a lesser extent, it is not easy to obtain a practicable algorithm from the theorem. This is because the bound on B_i/B_1 is determined to a large extent by a factor of the form $(Cn^{(2-r)/2})^a$, where C is a constant depending on r. The proof in [9], as it stands, leads to a value of C which is at least 100 when $r = 3$, so $\sigma = id$ is unlikely until n is very large. As a result, the algorithm can be much faster for larger values of n than for smaller, due to the lower probability of a *restart*. It seems that a more careful optimisation of the product of (5.1) and (5.2) (subject to certain constraints given in [9]) should lead to a practical algorithm for reasonable values of n (say $n = 100$), even when r is 3. The optimisation problem could even be automated in the algorithm.

There is also a problem, from the practical viewpoint, with larger values of r. Since the expected number of iterations of the trial in Rangraph is asymptotic to $\exp((r^2-1)/4)$, the problem of finding a practical algorithm for generating random r-regular graphs (labelled *or* unlabelled) remains open when r is at least 7 or 8. Any improvement for the labelled graphs should produce a corresponding improvement for the unlabelled ones.

REFERENCES

[1] E. BENDER, L. B. RICHMOND AND N. C. WORMALD, *Almost all chordal graphs split*, J. Austral. Math. Soc. Ser. A, 38 (1985), pp. 214-221.

[2] B. BOLLOBÁS, *The asymptotic number of unlabelled regular graphs*, J. London Math. Soc., 26 (1982), pp. 201-206.

[3] J. D. DIXON AND H. S. WILF, *The random selection of unlabeled graphs*, J. Algorithms, 4 (1983), pp. 205-213.

[4] P. ERDÖS, D. J. KLEITMAN AND B. L. ROTHSCHLD, *Asymptotic enumeration of K_n-free graphs*, Colloquio Internazionale sulle Teorie Combinatorie (Rome, 1973), Tomo II. Atti dei Convegni Lincei, 17 (1976), pp. 19-27.

[5] [5] P. ERDÖS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[6] C. L. LIU, *Introduction to Combinatorial Mathematics*, McGraw-Hill, New York, 1968.

[7] B. D. MCKAY AND N. C. WORMALD, *Automorphisms of random graphs with specified degrees*, Combinatorica, 4 (1984), pp. 325-338.

[8] N. C. WORMALD, *Generating random regular graphs*, J. Algorithms, 5 (1984), pp. 247-280.

[9] ———, *A simpler proof of the asymptotic formula for unlabelled graphs*, Indian J. Math., 28 (1986), pp. 43-47.

[10] E. M. WRIGHT, *Graphs on unlabelled nodes with a given number of edges*, Acta Math., 126 (1971), pp. 1-9.

[11] ———, *Graphs on unlabelled nodes with a large number of edges*, Proc. London Math. Soc. (3), 28 (1974), pp. 577-594.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics
009

ON SOME DISTANCE PROBLEMS IN FIXED ORIENTATIONS*

P. WIDMAYER†, Y. F. WU‡ AND C. K. WONG§

Abstract. In VLSI design, technology requirements often dictate the use of only two orthogonal orientations, determining both the shape of objects and the distance function, the L_1-metric, to be used for wiring objects. More recent VLSI fabrication technology is capable of creating edges and wires in both the orthogonal and diagonal orientations.

We generalize the distance concept to the case where any fixed set of orientations is allowed, and introduce a family of naturally induced metrics, and the subsequent generalization of geometrical concepts. A shortest connection between two points is in this case a path composed of line segments with only the given orientations. We derive optimal solutions for various basic planar distance problems in this setting, such as the computation of a Voronoi diagram, a minimum spanning tree, and the (minimum and maximum) distance between two convex polygons. Many other theoretically interesting and practically relevant problems remain to be solved. In particular, the new family of metrics may help bridge the gap between the L_1- and L_2-metrics, as those are the limiting cases for two and infinitely many regularly distributed orientations.

Key words. computational geometry, VLSI design, metric, Voronoi diagram, distance, polygon

AMS(MOS) subject classifications. B.7.2, F.2.2, I.3.5

1. Introduction.

Traditionally, computational geometry has dealt with problems in an extensive variety of settings. As to the properties of the space in which questions have been addressed, most problems fall into one of two major categories. On the one hand, there are problems in Euclidean space, e.g., the plane. Objects are characterized by hyperplane boundaries, such as straight lines, with no restrictions on the orientations. Questions refer to the natural topology, for instance the intersection question, and to the Euclidean metric, for instance the all nearest neighbors problem. On the other hand, many problems involving only so-called rectilinearly oriented objects have been considered. These are often expressed as special cases of the Euclidean case, with the objects' orientations being restricted to having to be parallel to the Cartesian coordinate axes. The flavor of questions referring to the topology alone does not change significantly under this restriction. In cases where the metric plays a role, sometimes the Euclidean metric is applied, but sometimes the restrictions on the objects are used to induce a different metric, namely the L_1-metric (the Manhattan distance).

The problems in the second category have received much attention, mainly because of two reasons. They are theoretically appealing because they often provide a first insight into a difficult problem and allow for efficient, often optimal, and simple solutions. The practical relevance of these problems is apparent in the form of the VLSI revolution, where the restriction to orthogonality comes from the limitations of fabrication technology.

* Received by the editors November 12, 1984; accepted for publication (in revised form) August 18, 1986. A preliminary version of this work was presented at the Symposium on Computational Geometry, Baltimore, Maryland, 1985 [27].

† Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, Postfach 6980, 7500 Karlsruhe, West Germany. This work was performed while this author was a visiting scientist at the IBM T. J. Watson Research Center, Yorktown Heights, New York 10598.

‡ MCC CAD, 9430 Research Blvd., Austin, Texas 78759. The work of this author was partially supported by the National Science Foundation under grants MCS 8342682 and ECS 8340031. This work was performed while this author was a summer visitor at the IBM T. J. Watson Research Center, Yorktown Heights, New York 10598.

§ IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.

In the VLSI application, however, the restriction of orthogonality applies not only to chip components, but also to wires (paths) between components. We therefore consider the L_1-metric to be the naturally induced metric in the orthogonal case. As time goes by that special case seems to be a bit too special: it is already quite common for VLSI artwork to contain 45° and 135° line segments. We do not anticipate this addition of allowed orientations to be the last step in the refinement of VLSI fabrication technology. This consideration leads us to a generalization: from problems with orthogonal objects and the L_1-metric we generalize to problems involving objects with several fixed orientations and the metric naturally induced by these fixed orientations.

As far as problems referring to the topology alone, as opposed to the metric, are concerned, the generalization to fixed orientations is not new. Such problems among many others include finding the intersections of objects, the connected components, and the contour. Edelsbrunner [6] has used objects with fixed orientations to approximate arbitrary objects more closely than would be possible with orthogonal objects alone. For problems that can be stated in terms of the intersection, containment, or enclosure relation, he describes a general transformation that allows the reduction of any question, posed for a set of convex objects with k orientations, to the corresponding question for a set of k-dimensional rectangles. For specific problems this general approach need not lead to the most efficient solutions. Güting [10] focusses on some problems for objects with fixed orientations, in order to bridge the complexity gap between his contour algorithm for rectangles and the general contour algorithm for arbitrary polygons. He describes a number of efficient solutions to various topological problems, and a contour algorithm with its complexity in between the two extremes.

Although many of the problems referring to the topology alone are interesting and important, we want to emphasize in this paper the problems involving distances; solutions to other problems with fixed orientations have been reported elsewhere [26]. As we are not aware of any previous work in this direction, we devote the next section to a detailed description of the distance function we are going to use, along with giving some of the consequential basic facts about the induced metric space. In § 3 we show how to optimally construct the Voronoi diagram for a set of points, a useful structure for many distance problems, and mention some applications. Several optimal solutions for distance computations between two sets of points, for instance convex polygons, are described in § 4. Section 5 offers a conclusion.

2. Definitions and basic facts. We deal with objects in the plane $\mathfrak{R}^2$. For $i = 1, 2, \cdots, \infty$, distance d_i is defined as usual: for any two points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $d_i(p_1, p_2) = (|x_1 - x_2|^i + |y_1 - y_2|^i)^{1/i}$.

The Euclidean distance d_2 and the rectilinear (Manhattan) distance d_1 only are of interest. The distances we will refer to lie "between" d_1 and d_2 in a sense that will be clarified.

Given a set A of at least two orientations (angles) in $\mathfrak{R}^2$, pairwise different, we denote by $a = |A|$ the number of orientations in A. We represent the orientations by the angles with the x-axis of the corresponding straight lines. For each orientation α, $0° \leqq \alpha < 180°$. 0° is the orientation of the positive Cartesian x-axis, 90° the orientation of the positive y-axis, etc. Let the orientations $\alpha_1, \alpha_2, \cdots, \alpha_a$ of A be sorted counterclockwise in the described way, with α_1 cyclically following α_a. A line, half line, or line segment is said to be *A-oriented* if its orientation is one of those in A. The *distance* d_A (the *A-distance*) between two points is the Euclidean length of a shortest zigzagging polygonal line segment between the points, such that each line segment of that polygonal

line segment is A-oriented:

$$\forall p_1, p_2 \in \mathfrak{R}^2: \quad d_A(p_1, p_2) = \begin{cases} d_2(p_1, p_2) & \text{if } p_1 \text{ and } p_2 \text{ lie on an } A\text{-oriented line,} \\ \min_{p_3 \in \mathfrak{R}^2} \{d_A(p_1, p_3) + d_A(p_3, p_2)\} & \text{otherwise.} \end{cases}$$

Let us first establish the fact that d_A can always be realized by a polygonal line segment consisting of at most two line segments, i.e., by using at most one extra point.

FACT 1. $\forall p_1, p_2 \in \mathfrak{R}^2: \exists p_3 \in \mathfrak{R}^2: d_A(p_1, p_2) = d_2(p_1, p_3) + d_2(p_3, p_2)$.

Proof. If p_1 and p_2 share an A-oriented line, then p_3 can be chosen to be any point on the line segment $\overline{p_1 p_2}$. Otherwise, consider the sequence $\alpha_1, \alpha_2, \cdots, \alpha_a$ of all orientations ordered counterclockwise. Let α_i and α_j be the orientations in A that are neighbors of the orientation of the line going through p_1 and p_2, with respect to the counterclockwise cyclic order (see Fig. 1(a)). Let p_3 be the intersection point of a line with orientation α_i through point p_1 and a line with orientation α_j through point p_2 (see Fig. 1(b)). Because p_3 shares an A-oriented line with p_1, $d_A(p_1, p_3) = d_2(p_1, p_3)$. Correspondingly, $d_A(p_3, p_2) = d_2(p_3, p_2)$. Observe that $d_A(p_1, p_2) > d_2(p_1, p_2)$ if p_1 and p_2 do not lie on a common A-oriented line. To see that the polygonal line segment $\overline{p_1 p_3 p_2}$ is shortest among all A-oriented polygonal line segments from p_1 to p_2, let us assume the contrary. That is, assume that there exists a point p_3' with $d_A(p_1, p_3') + d_A(p_3', p_2) < d_A(p_1, p_3) + d_A(p_3, p_2)$. Then the polygonal line segment from p_1 to p_2 through p_3' must contain a line segment with an orientation different from α_i and α_j, say with orientation α_k, $i \neq k \neq j$. Let p_1' and p_2' be two distinct points on that line segment, with p_2' closer to p_2 than p_1' (see Fig. 1(c)). Then the polygonal line segment realizing the A-distance between p_2' and p_2 must intersect at least one of the lines with

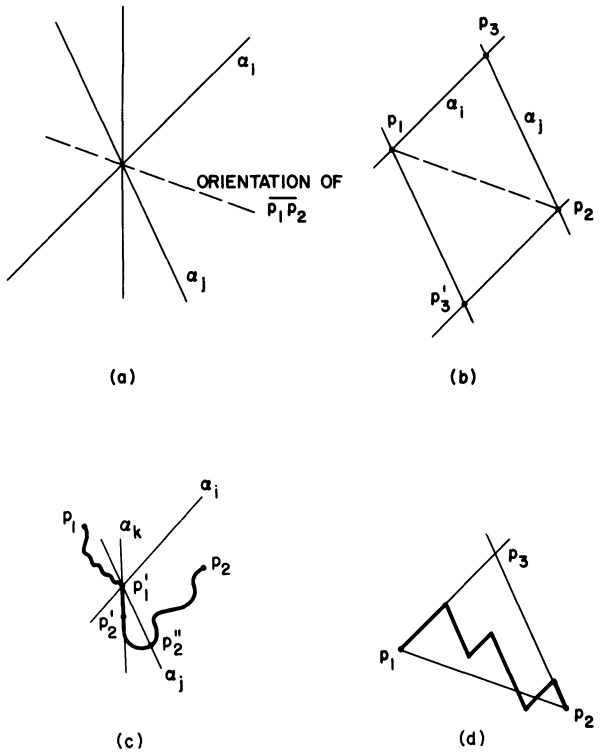


FIG. 1. *A set A of orientations, and the A-distance between points p_1 and p_2.*

orientations α_i and α_j through p_1' at least once; let p_2'' be such an intersection point. But then, with p_1' and p_2'' lying on a common A-oriented line, $d_A(p_1', p_2'') = d_2(p_1', p_2'') < d_A(p_1', p_2') + d_A(p_2', p_2'')$, a contradiction. Therefore, line segment $\overline{p_1 p_3 p_2}$ is shortest. □

Note that there exist infinitely many other polygonal line segments also realizing $d_A(p_1, p_2)$, namely those which are obtained by breaking $\overline{p_1 p_3}$ and $\overline{p_3 p_2}$ into arbitrary pieces and mixing these pieces into a different order (see Fig. 1(d)).

FACT 2. *Distance d_A induces a metric on $\mathfrak{R}^2$ for any given A.*

Proof. (1) $\forall p_1 \in \mathfrak{R}^2$: $d_A(p_1, p_1) = 0$. p_1 lies on any A-oriented line with itself, and hence $d_A(p_1, p_1) = d_2(p_1, p_1) = 0$.

(2) $\forall p_1, p_2 \in \mathfrak{R}^2$: $d_A(p_1, p_2) = d_A(p_2, p_1)$. By Fact 1, $\exists p_3 \in \mathfrak{R}^2$: $d_A(p_1, p_2) = d_2(p_1, p_3) + d_2(p_3, p_2) = d_2(p_2, p_3) + d_2(p_3, p_1) = d_A(p_2, p_1)$.

(3) $\forall p_1, p_2, p_3 \in \mathfrak{R}^2$: $d_A(p_1, p_3) + d_A(p_3, p_2) \geqq d_A(p_1, p_2)$. If p_1 and p_2 share an A-oriented line, then $d_A(p_1, p_2) = d_2(p_1, p_2)$, and hence the triangle inequality for d_2 and the fact that $\forall p, p' \in \mathfrak{R}^2$: $d_A(p, p') \geqq d_2(p, p')$ ensure the claim. Otherwise, it follows directly from the definition of d_A. □

For a set A of orientations, we call $\mathfrak{R}^2$ with the metric induced by d_A the *A-plane*, denoted by $\mathfrak{R}^2_A$. For problems in the A-plane, no other orientations than those given in A will be considered for all lines, half lines, and line segments, whether they are objects given as input to the problem or constructed as output, i.e., solution to the problem, e.g., paths or hulls. Note, however, that during the computation of the solution other orientations will often be considered in intermediate steps. We call objects obeying this restriction *A-objects*. Similarly, we transfer other notions to the A-plane. We call an A-polygon *A-convex*, if its intersection with any A-line is either empty or one connected component. The smallest A-convex A-polygon containing a given A-polygon is called its *A-convex hull*. For a set of points, any smallest A-convex A-polygon containing all the points is called an A-convex hull. Note that the ambiguities from the orthogonal case (see [18]) carry over to our framework, because the former is a special case of the latter.

For two sets of points P_1 and P_2 the A-distance is defined as follows:

$$d_A(P_1, P_2) = \min_{p_1 \in P_1, p_2 \in P_2} \{d_A(p_1, p_2)\}.$$

In case P_1 or P_2 consists of just one point p, equivalently we write $d_A(p, P_2)$ or $d_A(P_1, p)$. For a point p and a distance d, the locus of all points p' with $d_A(p, p') = d$ is called the *A-circle* with radius d at center p.

FACT 3. *The A-circle with radius d at center p is the boundary of the $2a$-gon, with the corner points lying on the intersections of the circle with radius d at center p with the a A-oriented lines through p, and edges between corner points adjacent on the circle (see Fig. 2).*

Proof. The corner points have distance d from p, because they share an A-oriented line with p and have Euclidean distance d from p. In between two adjacent corner points, consider any point p' on the edge of the A-circle (see Fig. 2). The orientation of $\overline{pp'}$ lies between the orientations of the lines associated with both adjacent corners, say p_1 and p_2. Hence, the A-distance between p and p' is realized by the two line segments $\overline{pp''}$ and $\overline{p''p'}$, where p'' lies on $\overline{pp_1}$, and $\overline{p''p'}$ is parallel to $\overline{pp_2}$. Because $\overline{pp_2}$ has the same length as $\overline{pp_1}$, $\overline{p''p_1}$ has the same length as $\overline{p''p'}$. Hence, the A-distance from p to p' is the same as from p to p_1. □

The orientations of A-circle edges are in general different from those in A. Let $B = \{\beta_i \mid i \in \{1, \cdots, a\}\}$ denote the set of all these edge orientations (see Fig. 2).

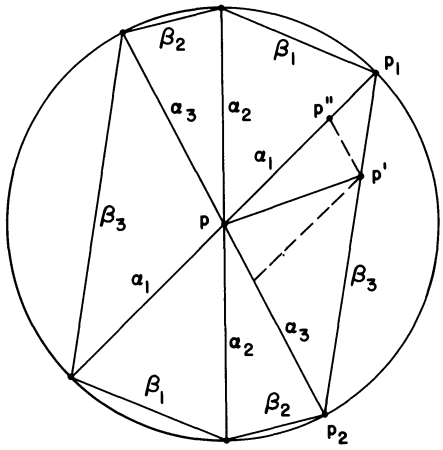


FIG. 2. *An A-circle with A-circle edges.*

For two points p_1, p_2 in the A-plane, the *bisector* $B_A(p_1, p_2)$ of p_1 and p_2 is the locus of all points that have equal A-distance to p_1 and p_2; i.e.,

$$B_A(p_1, p_2) = \{p \in \mathfrak{R}_A^2 | d_A(p_1, p) = d_A(p_2, p)\}.$$

The bisector $B_A(P_1, P_2)$ of two sets of points $P_1, P_2 \subseteq \mathfrak{R}_A^2$ is defined analogously as

$$B_A(P_1, P_2) = \{p \in \mathfrak{R}_A^2 | d_A(P_1, p) = d_A(P_2, p)\}.$$

See Fig. 3 for an example of a bisector.

The form of the bisector of two points p_1, p_2 is the following. The center point of the line segment $\overline{p_1 p_2}$ belongs to the bisector because its distance is realized by the combination of line segments of the same two orientations from both p_1 and p_2. This point is called the *anchor point* of $B_A(p_1, p_2)$. The lines of all orientations of A through p_1 and through p_2 partition the A-plane into (bounded and unbounded) regions through which none of these lines passes, called *fields.* In the example in Fig. 3, the heavy polygonal line shows the lower half of the bisector for points p_1 and p_2. The star on that line is the anchor point.

It is obvious that the boundary of any field consists of at most four half lines and line segments.

FACT 4. *The bisector of two points p_1 and p_2, when restricted to a field of p_1, p_2, is either empty, a line segment, or the whole field.*

Proof. We show that if there are two points p_1', p_2' in a field, where p_1' is closer to p_1 than to p_2, and p_2' is closer to p_2 than to p_1, then the bisector is a single line segment. For fields where only points closer to p_1 or only points closer to p_2 occur (but not both), the bisector is empty, and if none of both types of points occur, then all points are equidistant from both p_1 and p_2; the latter may already happen in the orthogonal case (see [17], [18]).

Assume that for field F, p_1' and p_2' as described exist. Because the distance is a continuous function, there must exist a point p which is equally distant from both p_1 and p_2 and lies within F. Let the boundary of F consist of line segments with orientations α_i, α_{i+1}, α_j, and α_{j+1}, and let d be the distance of p to p_1 (and to p_2), as shown in

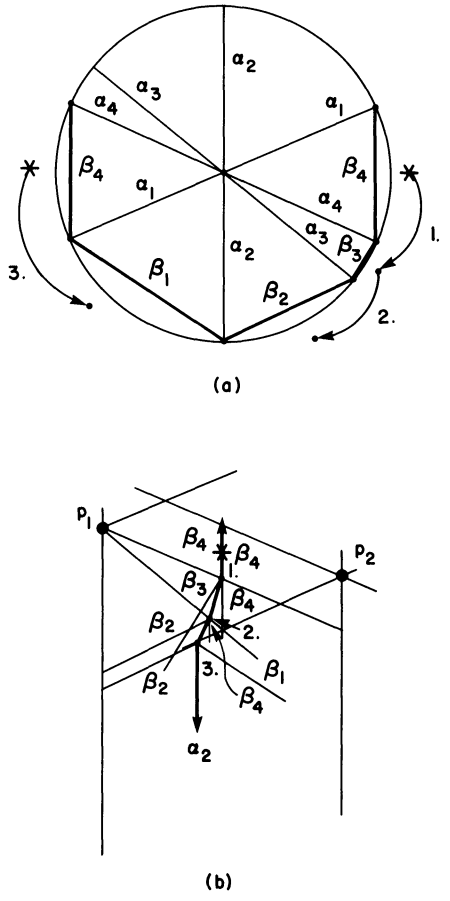


FIG. 3. *The march on the A-circle boundary when following the bisector.*

Fig. 4(a). Then the equations of the two equidistant lines (edges of the A-circles) with distance d for the points p_1 and p_2 can be expressed as

$$y_1 = \beta_i x + f_i(d) \quad \text{for } p_1,$$
$$y_2 = \beta_j x + f_j(d) \quad \text{for } p_2$$

where f_i, f_j are linear functions of d. The curve of all intersection points of these lines fulfills both equations:

$$\beta_i x + f_i(d) = \beta_j x + f_j(d).$$

Hence, x is a linear function of the distance, and, as y_1, y_2 are both linear in x, the resulting curve is linear. Figure 4(a) shows the equidistant point p' with distance d' from both p_1 and p_2, and the line segment passing through both points, p and p'. □

Note that because of the continuity of the distance function, the bisector of two points is a continuous curve. Only in unbounded fields with boundary orientations α_i and α_{i+1}, can the bisector consist of the entire field (see Fig. 4(b)). We handle this case in the same spirit as in [15] for the L_1-case: we just choose the halfline parallel to α_i (we could have chosen α_{i+1} or any halfline in between alternatively) as bisector.

At the anchor point, lying in a field with boundary orientations α_i and α_{i+1} for some $i \in \{1, \cdots, a\}$, the bisector consists of a line segment with direction β_i. When

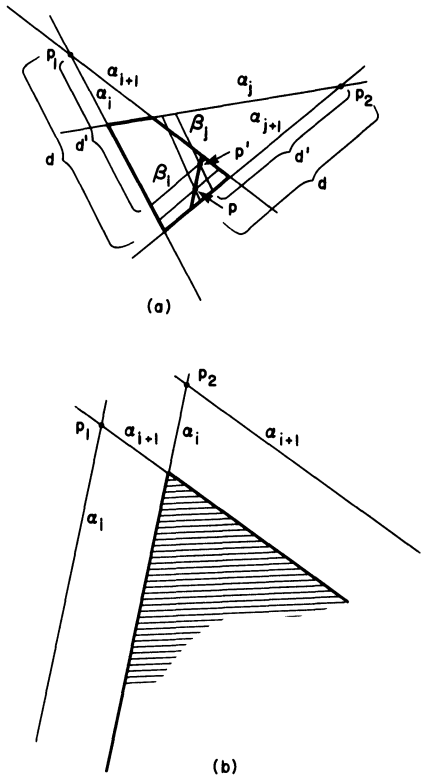


FIG. 4. *A bisector within a field, and a degenerate bisector.*

we follow the bisector in one direction, starting at the anchor point, the bisector leaves the initial field into an adjacent field. Hence, the next line segment consists of the combination of β_{i-1} and β_i or of β_i and β_{i+1}. In general, from one field to an adjacent field, one of the orientations contributing to the combined orientation changes. Tracing the bisecting polygonal line in this way can be identified with a march along the boundary edges of the A-circle: starting with two β_i in opposite positions on the circle, we switch to adjacent β, clockwise from one β_i and counterclockwise from the other, until finally both considered β are adjacent on the A-circle boundary (see Fig. 3). At that point, say when switching to orientations β_j and β_{j+1}, the bisector will continue with a halfline parallel to the common α-orientation of the field boundaries, namely α_{j+1} (see Fig. 3). The reason is that the distance of any point in the field to p_1 as well as to p_2 can be realized by using orientation α_{j+1} within that field.

Taking into account the degenerate case that both orientations contributing to the orientation of the bisector may change at the same step, we have the following fact.

FACT 5. *The bisector of two points p_1, p_2 in the A-plane is an unbounded, continuous polygonal line, consisting of no more than $2a-1$ pieces. It partitions the A-plane into two unbounded regions, $B_A(p_1|p_2)$ and $B_A(p_2|p_1)$, where $p_1 \in B_A(p_1|p_2)$ and $p_2 \in B_A(p_2|p_1)$. All points in $B_A(p_1|p_2)$ are at least as close to p_1 as to p_2, and all points in $B_A(p_2|p_1)$ are at least as close to p_2 as to p_1.* □

3. Voronoi diagram and applications. In our construction of the Voronoi diagram for a set of points we follow a mixed approach derived from the ones given for the L_1-metric in [12], [13] and [15] and generalized to the A-metric. The problem is solved

by divide-and-conquer; we partition the set of points along an orientation of A, as in [13], [15] for L_1. This ensures that when the bisector between both subsets is constructed in the merging step, it will be monotonic in the chosen direction, though not necessarily continuous (see also [2]). The starting bisector is chosen as in [2]. The main process of constructing the bisector in the merging step is a generalization of [15] and was also applied independently by [12], and further elaborated in [13]; in [2] it has been shown how to deal with unbounded excursions of the bisector at discontinuities for convex distance functions.

For a set of n points, $P=\{p_1, p_2, \cdots, p_n\}$, the *Voronoi polygon* of point p_i with respect to P in A-metric, denoted by $V_A(p_i, P)$, is the locus of all points in $\mathfrak{R}^2_A$ that are at least as close to p_i as to any other $p_j \in P$. It is the intersection of all the $B_A(p_j|p_k)$, $p_k \in P-\{p_j\}$. The set of all Voronoi polygons for the points in P is a partition of the A-plane; it is called the *Voronoi diagram* $V_A(P)$ of P. The boundary of a Voronoi polygon consists of portions of bisectors; a portion of a bisector forming part of the Voronoi polygons of p_i and p_j is called a *Voronoi edge* between p_i and p_j. The endpoints of a Voronoi edge are called *Voronoi points*. The straight-line dual $D_A(P)$ of the Voronoi polygon for P is the graph where the nodes are the points in P, and there is an edge between two points whenever these points share a Voronoi edge.

We use the straight-line dual of the Voronoi diagram to solve other problems, such as constructing a minimum spanning tree. Recall that for Euclidean distance, the straight-line dual of the Voronoi diagram triangulates the convex hull of the point set; this triangulation is known as *Delaunay-triangulation* (see [19]). Note, however, that the Voronoi diagram in the L_1-metric and hence in the A-metric is not unique; its straight-line dual need not be a Delaunay-triangulation [14], and in fact, it need not be a triangulation at all. For an illustration see Fig. 5: The straight-line dual of the Voronoi diagram for points p_1, p_2 and p_3 consists of edges $\overline{p_1p_2}$ and $\overline{p_2p_3}$. Edge $\overline{p_1p_3}$ is missing because $B_A(p_1, p_3)$ does not belong to the Voronoi diagram. Nevertheless, a minimum spanning tree for P is a subgraph of $D_A(P)$. The properties of Voronoi diagrams in L_1 (see [12], [13], [15]) can be generalized to A-metric correspondingly; we restrict ourselves here to an outline of the construction algorithm for Voronoi diagrams for sets of points, using the known methods for L_1 as basis.

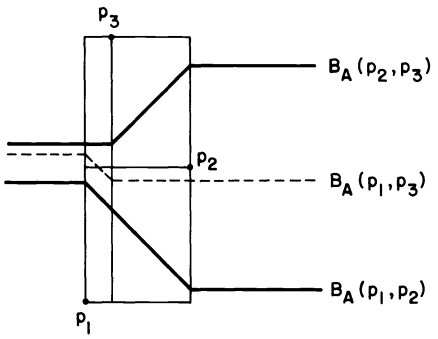


FIG. 5. *A Voronoi diagram in L_1 whose straight-line dual is no triangulation.*

First we choose one of the orientations of A for a straight line to split the problem into two subproblems, say α_2 (this orientation is not horizontal). Then we find the median of the points in P with respect to the chosen orientation α_2 and partition P into the set P_L of points lying to the left of an α_2-oriented partitioning line and a set P_R lying to its right of (almost) equal size. We recursively construct the Voronoi

diagrams $V_A(P_L)$ and $V_A(P_R)$, and merge the two diagrams into $V_A(P)$. The merge phase can be realized in time $O(n)$, where n is the size of P, yielding an overall solution in time $O(n \log n)$.

The merge phase consists of two steps: first, we find a starting bisector for two points $p_1 \in P_L$ and $p_2 \in P_R$, such that an unbounded part of this bisector will belong to the bisector we are constructing between P_L and P_R. Then we extend this bisector by adding Voronoi edges between pairs of points, one from P_L and one from P_R, and cutting the corresponding Voronoi edges from $V_A(P_L)$ and $V_A(P_R)$, respectively.

The starting bisector can be determined in a way generalized from [12] and [13]. This has been described in more detail in [27]. Also, the procedure described independently by [2] can be used to determine the starting bisector in our case. We therefore omit a more detailed explanation here.

The bisector between P_L and P_R is constructed similar to [15], with additional provisions for the case when the merge bisector makes unbounded excursions, as described in [2]. Again, a more detailed description can be found in [27]. For constant a we conclude:

THEOREM 1. *A Voronoi diagram in A-metric for a set of n points in the plane can be constructed in $O(n \log n)$ time using $O(n)$ space, which is asymptotically optimal in the worst case.*

Proof. Correctness follows from the above considerations. Optimality is due, for instance, to a reduction of sorting [12]. □

Because a minimum spanning tree (MST) for a set of points is a subgraph of the straight-line dual of the Voronoi diagram in any metric [21] and $D_A(P)$ is a planar graph, we have:

COROLLARY 1. *A minimum spanning tree in A-metric for a set of n points in the plane can be computed in $O(n \log n)$ time, using $O(n)$ space, which is optimal.*

Proof. Use any MST-algorithm on $D_A(P)$ which does not require more than $O(n \log n)$ time, for instance Cheriton and Tarjan's algorithm for planar graphs [3]. Optimality follows from a reduction of sorting [12]. □

The Voronoi diagram has numerous applications to computational geometry problems (see e.g. [14], [19]); we will simply mention two of them.

COROLLARY 2. *The all-nearest-neighbors problem in A-metric for a set of n points in the plane can be solved in $O(n \log n)$ time, using $O(n)$ space.*

COROLLARY 3. *A nearest neighbor query in A-metric for a set P of n points in the plane with a query point $p \notin P$ can be answered in $O(\log n)$ time with $O(n \log n)$ preprocessing time and $O(n)$ storage.*

Proof. We only need to locate the query point in the planar subdivision induced by the Voronoi diagram. Edelsbrunner, Guibas and Stolfi [8] show how to optimally locate a point in $O(\log n)$ query time with $O(n \log n)$ preprocessing time and $O(n)$ storage. □

4. Distance between special sets of points. When we plan collision free paths through a set of obstacles, knowing the distance between simple polygons is important. When we regard a polygon as a set of points, the distance between two polygons is defined as the distance between two sets of points. Recall that for two sets of points, P and Q, $d(P, Q) = \min_{p \in P, q \in Q} \{d(p, q)\}$. We focus on pairs of nonintersecting simple polygons because this is a practically interesting case. This is not a restriction of the general problem as far as computational complexity is concerned because for two simple polygons with the edges given in cyclic order, the existence of an intersection can be determined in time $O((n_1 + n_2) \log (n_1 + n_2))$ or in time $O(n_1 + n_2)$ if the polygons

are convex, where n_1 and n_2 are the numbers of vertices of the first and second polygon. If two convex polygons have distance d, then the maximal size unit circle that can be pushed through in between the polygons has diameter exactly d, if no rotations of that circle are allowed (see Fig. 6). Also, in VLSI wire routing it is desirable to make shortest wire connections between objects. The distance between objects determines the number of wires that can be run through in between the objects. The orientations allowed for wires are of course subject to the same restrictions as the orientations for object boundaries. Here the importance of operating in the same set of orientations for both aspects becomes apparent. Schwartz [20] gave an $O(\log n_1 \cdot \log n_2)$ algorithm for finding the minimum distance between two convex polygons in the Euclidean plane. Edelsbrunner [7] (and independently Chin and Wang [4]) improved on that and presented a worst-case optimal algorithm using $O(\log n_1 + \log n_2)$ time for that problem. The fact that both algorithms can run in sublinear time is due to the convexity of the polygons. Edelsbrunner [7] shows that the computation of the maximal distance between two convex polygons requires $\Omega(n_1 + n_2)$ time in the worst case, because convexity can no longer be used (viewed from inside a convex polygon, its boundary is not convex). He also describes an optimal algorithm for the latter problem. The same result has also been described by Bhattacharya and Toussaint [1] and by Toussaint and McAlear [24].

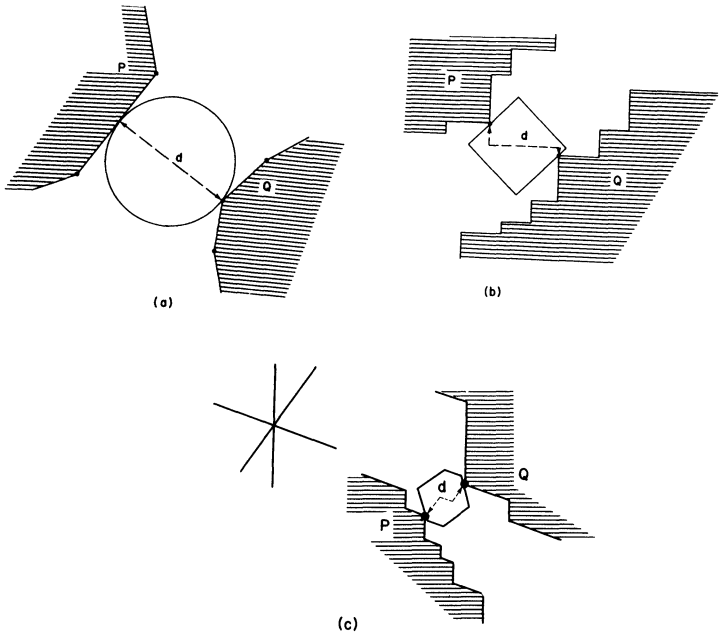


FIG. 6. *Polygon distances.*

In VLSI design an A-metric is much more adequate than the Euclidean metric. A-convexity is a more general notion than convexity in the Euclidean sense: an A-convex polygon need not be convex in L_2. A vertex with inner angle at most 180° (more than 180°) will be called a *convex* (*concave*) vertex of a polygon. Note that a convex polygon in the Euclidean sense has only convex vertices, but an A-convex polygon may have concave vertices as well. In the following we shall describe how to solve the minimum A-distance problem for two A-convex A-polygons in time $O(n_1 + n_2)$, and show the optimality of that procedure.

LEMMA 1. *The computation of the minimum A-distance between two A-convex A-polygons in the A-plane with n_1 and n_2 vertices, respectively, requires $\Omega(n_1+n_2)$ time in the worst case.*

Proof. Consider the special case of the L_1-metric, where n squares of equal size are aligned along a 45° line, such that each square overlaps by the same small amount with both its (or one, for the first and the last one) neighbors. For an illustration see Fig. 7. The contour of the union of the squares consists of two staircase curves, an upper one and a lower one; the staircase curves, with their first and last edges removed, define the relevant part of the boundary of the convex polygons; the squares describe the free space between them. Then the distance between those two polygons changes as soon as any of the convex corner points is moved by a small amount towards the other polygon. Hence, it is necessary to consider each of the $O(n)$ corner points at least once before knowing the distance. □

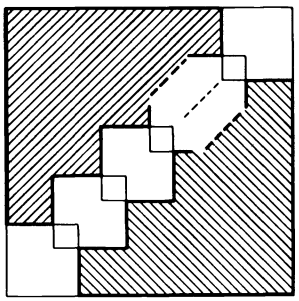


FIG. 7. *Distance-computation for x-y-convex polygons requires linear time.*

Let us first describe our algorithm for the case $a=2$, in L_1-metric, and then extend it to general A. We divide the problem into two subproblems. Any of two closest points, $p \in P$ and $q \in Q$, is either a convex vertex or a point on a boundary edge. If one is a convex vertex and the other is not, then the distance is also realized by a pair of points, none of which is a vertex. Furthermore, both edges on which these points lie have the same orientation, and the A-path realizing the distance uses only one orientation. Hence, we can find the distance by taking the minimum of

— the distance between any two vertices, one from P, the other from Q;
— the distance between any two horizontal edges, one from P, the other from Q, where the edges share a common x-value; and
— the distance between any two vertical edges, one from P, the other from Q, where the edges share a common y-value.

As is common in the field, we assume that the polygons are specified in the input to the problem by a sequence of vertices (edges) given in the order in which they occur when following the boundary in clockwise or counterclockwise direction. By convexity, each polygon boundary consists of at most 4 staircase curves, defined as the sequences of lines between and including two adjacent extreme edges (we say the left, right, upper and lower extreme edge) (see Fig. 8). Simple geometric arguments show that there is exactly one staircase of each polygon where candidate points realizing the polygon distance lie. In any such pair of staircases, the relative position within the polygon of one staircase has to be opposite to that of the other: a staircase between the right and lower extreme edges (a right lower staircase, for short) can realize the

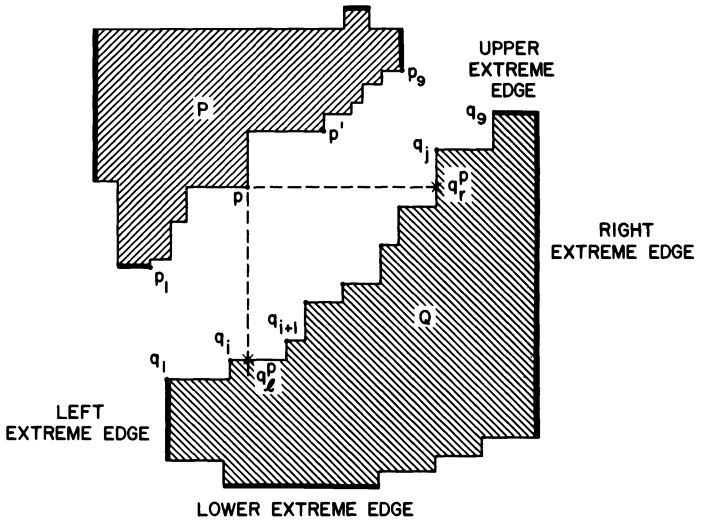


FIG. 8. *Two x-y-convex polygons, P and Q, with staircases facing each other, and candidates for vertices of Q closest to vertex* $p \in P$.

polygon distance only with a left upper staricase, etc. This allows for an easy, efficient test for determining the relative location of the polygons: if a right lower staircase lies partly or wholly in the left upper quadrant of a left upper staircase, i.e., if there is a point (x_{rl}, y_{rl}) of the right lower staircase and a point (x_{lu}, y_{lu}) of the left upper staircase with $x_{rl} < x_{lu}$ and $y_{rl} > y_{lu}$, then the two staircases will contain the points realizing the distance between the polygons. We say that the two respective staircases *face* each other. Note that because each extreme edge belongs to two staircases, it may happen that for a pair of polygons two pairs of staircases face each other. In this case the points realizing the polygon distance belong to both staircases on each polygon, i.e., they are on the extreme edges.

In the following we describe how to compute the distance between a right lower and a left upper staircase; as all other three combinations are similar, this describes a solution of the distance problem.

The minimal horizontal (vertical) edge distance between two staircases can be easily determined by a linear march along the staircase's vertices. Assume that the convex vertices are given in order, sorted from the lower left to the upper right end of each staircase, denoted by $p_1, p_2, \cdots, p_{n_1}$ and $q_1, q_2, \cdots, q_{n_2}$ (see Fig. 8).

In order to compute the two closest vertical line segments, we simultaneously move on both staircases from bottom to top. We may think of sweeping a horizontal line. At any stage, the sweep line will intersect (at most) two vertical line segments. Their distance can be determined in constant time. Sweeping the line consists of replacing one of the two vertical line segments, which is feasible in constant time, because they are given in the desired order. By keeping track of the minimum of all edge distances, we can conclude the following.

LEMMA 2. *For two rectilinear staircases containing* n_1 *and* n_2 *convex vertices the minimal edge distance in* L_1 *and a pair of edges realizing that distance can be computed in time* $O(n_1+n_2)$ *using* $O(n_1+n_2)$ *space.*

The more interesting task is the computation of a pair of closest vertices that might realize the polygon distance. Recall that only two convex vertices can possibly

realize the polygon distance because two concave vertices cannot be a closest pair of vertices and one convex and one concave vertex always have adjacent edges that are closer. Let us first look only at those pairs of vertices that might realize the polygon distance; later we will consider the task of finding a pair of closest vertices. First of all, observe that any vertex $q \in Q$ closest to some given vertex $p \in P$ must lie in or be adjacent to the quadrant of p away from P (see Fig. 8). All vertices q_k, $k < i$ in Fig. 8 cannot be as close to p as q_i is because a rectilinear path from p to q_k realizing the distance $d_1(p, q_k)$ goes through q_i. Similarly for q_k, $k > j$. Note that for the solution of our original problem (the polygon distance) q_i and q_j need not even be considered because $d_1(p, q_i)$ and $d_1(p, q_j)$ are certainly larger than $d_1(p, q_l^p)$ and $d_1(p, q_r^p)$, respectively. But for the sake of solving the closest vertex problem, in the following we will consider them as well.

The structure of the algorithm is roughly as given in the following; we do not give details as to how to treat the extreme cases and to properly exit the loops in order not to obscure the main idea.

ALGORITHM CLOSEST PAIR OF VERTICES
$P :=$ sequence of vertices, along lower right staircase;
$Q :=$ sequence of vertices, along upper left staircase;
$q_{\min} := p_{\min} :=$ dummy; {a closest pair of vertices}
$d_{\min} := \infty$; {distance between $p_{\min}$ and $q_{\min}$}
$q_l := q_r := q_1$;
FOR p FROM $p_1 \in P$ TO $p_{n_1} \in P$ DO
  {move q_l to the left end in the quadrant of p}
  WHILE x-value of $q_l <$ x-value of p DO $q_l := \text{succ}(q_l)$;
  {move q_r to the right end in the quadrant of p}
  WHILE y-value of $\text{succ}(q_r) \leqq y$-value of p DO $q_r := \text{succ}(q_r)$;
  {determine a closest vertex q in the range $q_l \cdots q_r$}
  determine q such that
    $\text{pred}(q_l) \leqq q \leqq \text{succ}(q_r)$, and
    $d(p, q) = \min \{d(p, q') | \text{pred}(q_l) \leqq q' \leqq \text{succ}(q_r)\}$
  {keep track of a closest pair p, q of vertices}
  IF $d(p, q) < d_{\min}$ THEN $d_{\min} := d(p, q)$, $q_{\min} := q$, $p_{\min} := p$;
{a closest pair of vertices is $p_{\min} \in P$, $q_{\min} \in Q$}
END of algorithm.

The correctness of the algorithm follows from the previous considerations. Because p, q_l and q_r move through the sequences P and Q of vertices just once the algorithm will run in linear time if the closest-vertex-in-range query can be answered in constant time. This is clear for the two vertices outside the quadrant of p and can be accomplished for all vertices of Q in that quadrant as follows.

For a certain p, let q_l and q_r be the leftmost and rightmost vertices in Q in the quadrant of p as described. For any $q \in Q$ from q_l to q_r imagine a very small L_1-circle (i.e., a diamond), centered at q, growing in size until its boundary hits p. Then because of the relative position of p with respect to q, this L_1-circle will touch p with its left upper edge. Hence, the point $q \in Q$ with leftmost (= uppermost) 45°-line through q among all those q's in the quadrant of p is closest to p. This is the case for $q = (x_q, y_q)$ with maximal value $(y_q - x_q)$ in the given range of vertices of Q. The fact that the exact position of p is of no importance allows us to preprocess Q, independent of P, such that the subsequent queries for the maximum key in a range of elements can be answered in constant time.

To this end we construct a Cartesian tree first described in [25]. The Cartesian tree for a sequence of numbers, $S = s_1, s_2, \cdots, s_r$, is a binary tree, where each node represents a number. The root represents the maximum number, say s_m; the left subtree is a Cartesian tree for the sequence of numbers $s_1, s_2, \cdots, s_{m-1}$, and the right subtree is a Cartesian tree for the sequence $s_{m+1}, s_{m+2}, \cdots, s_r$. That is, the Cartesian tree is a specific heap structure imposed on the original (unsorted) sequence of numbers. Gabow, Bentley and Tarjan [9] note that this tree structure can be used for *range searching for maximum* by applying the technique of finding a nearest common ancestor for two nodes in a given (static) binary tree, which is feasible by a simple address computation in constant time once linear time and space for preprocessing has been allowed (see Harel and Tarjan [11]). The construction of the Cartesian tree for the sequence $(y_{q_i} - x_{q_i})$, $i = 1, \cdots, n_2$, can be done in time proportional to n_2, and the tree uses space proportional to n_2. Because in the Cartesian tree the nearest common ancestor of the nodes representing $(y_{q_i} - x_{q_i})$ and $(y_{q_j} - x_{q_j})$ represents the maximum of all values between positions i and j, inclusively, each search for the maximum value in a given range bounded by two nodes' values can be answered by a nearest common ancestor query, and hence takes only constant time. That is, a total of n_1 range searches for maximum in a sequence of n_2 values can be accomplished in $O(n_1 + n_2)$ time and $O(n_2)$ space, preprocessing included. This leads to the following lemma.

LEMMA 3. *For two rectilinear staircases containing n_1 and n_2 convex vertices, the minimal vertex distance in L_1 and a pair of vertices realizing that distance can be computed in time $O(n_1 + n_2)$ using $O(n_1 + n_2)$ space.*

The decomposition property of the distance problem for two rectilinear convex polygons into edge and vertex distance for two staircases, together with Lemmas 2 and 3 and the proof of Lemma 1, gives the following theorem.

THEOREM 2. *The L_1-distance between two rectilinear convex polygons with a total of n vertices (corner points) can be computed in time $O(n)$ with $O(n)$ space, which is optimal in the worst case.*

Note that our algorithm for the rectilinear case of the minimum distance problem can also be used to find the minimum vertex distance, under the assumption that the polygons are disjoint. By making slight modifications we can take the concave vertices into account, as well as find the closest pair of vertices even for overlapping polygons. To this end we augment the considered pairs of vertices, looking at three quadrants of any vertex $p \in P$, and searching for the closest vertex $q \in Q$ (see Fig. 9). Instead of asking one maximum-in-range query for each p, we ask three out of the following four queries:

(i) minimum $(y_q + x_q)$ in range;
(ii) minimum $(y_q - x_q)$ in range;
(iii) maximum $(y_q + x_q)$ in range;
(iv) maximum $(y_q - x_q)$ in range.

Each query in (i)-(iv) is carried out on a separate Cartesian tree representing an edge of the L_1-circle (see Fig. 9). It is clear that the overall runtime will still be asymptotically the same. This leads to the following.

COROLLARY 4. *The L_1-distance between the vertices of two rectilinear convex polygons with a total of n vertices, together with two vertices realizing that distance, can be computed in $O(n)$ time with $O(n)$ space, which is optimal in the worst case.*

Edelsbrunner [7] notes that his procedure for finding the minimum polygon distance does not find the minimum vertex distance for two convex polygons and the distance in the Euclidean sense. In fact, Chin and Wang [5] have shown that the computation of the minimum vertex distance between two convex polygons, separable

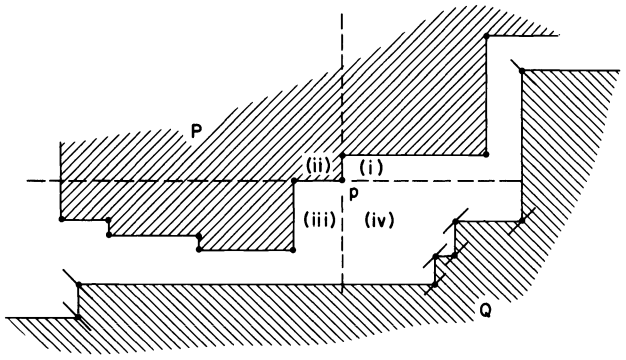


FIG. 9. *Vertex distance between P and Q in* L_1.

by a line, requires $\Omega(n)$ time in the worst case, and they have given an optimal algorithm for that problem. Algorithms with the same time complexity have also been presented by McKenna and Toussaint [16] and by Toussaint and Bhattacharya [23]. Toussaint [22] describes a linear-time algorithm for nonseparable polygons.

Let us now turn to the distance problem for two A-convex A-polygons.

FACT 6. *The boundary of an A-convex A-polygon consists of at most* $2a$ *staircases, each formed by line segments of two orientations that are adjacent in the A-circle. The boundary contains at most* $2a$ *extremes: extremes are boundary edges or boundary corner points forming part of an A-oriented supporting line for the polygon. The extremes lie on the ends of staircases; two adjacent staircases share an extreme* (see Fig. 10).

Proof. Consider a concave vertex, i.e., a vertex with an inner angle of more than 180°. Call the two orientations of its adjacent boundary edges α_i and α_j. If α_i and α_j are not adjacent on the A-circle, then there is an orientation α' between them. But this implies that there exists a line with orientation α' whose intersection with the polygon is not connected, and hence the polygon is not A-convex, a contradiction. This implies that a supporting A-line for the polygon can touch a staircase of two adjacent orientations (called an *A-staircase*) only at its two ends. A line between the two end points of an A-staircase with edge orientations α_i and α_{i+1} has an orientation between α_i and α_{i+1}. We call (α_i, α_{i+1}) the *global orientation* of the A-staircase. At the two ends of an A-staircase with global orientation (α_i, α_{i+1}) there exist two A-lines with orientations α_i and α_{i+1} that are supporting lines of the polygon. Therefore, since we have $2a$ adjacent orientations in the A-circle when walking in cyclic order, this bounds the number of extremes and of staircases (see Fig. 10). □

By the same token as in the L_1-case, the distance computation can then be split into the edge distance between staircases with the same pair of orientations, and the vertex distance computations where an extreme may stand for several adjacent empty staircases. For the edge distance computation only parallel edges are of interest. We can hence compute the edge distance as in the L_1-case by sweeping a A-lines through the plane. Each of the a sweeps will give us as a result the minimum edge distance for a pair of staircases, altogether solving the problem.

LEMMA 4. *For two A-convex A-polygons consisting of a total of n vertices the minimum edge A-distance and a pair of edges realizing that distance can be computed in* $O(n)$ *time using* $O(n)$ *space.*

Accordingly, the distance between the two closest vertices can be found by modifying the closest vertex query for each vertex of one polygon, and walking along the boundary of the other polygon with at most $2a$ ranges (see Fig. 11). For any vertex

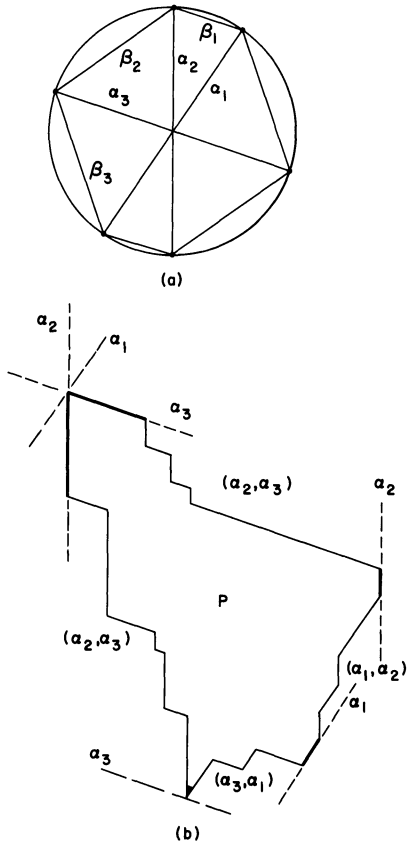


FIG. 10. *An A-convex A-polygon.*

$p \in P$, we query the vertices of Q with $a+1$ ranges, posing $a+1$ out of the $2a$ queries:

$$\begin{array}{l}(i) \text{ minimum } \beta_i \text{ in range} \\ (a+i) \text{ maximum } \beta_i \text{ in range}\end{array} \quad \text{for } i = 1, 2, \cdots, a.$$

Because marching along the boundary of Q with $a+1$ ranges takes time linear in the number of vertices for fixed a, and the same holds for preprocessing and storing information in $2a$ different Cartesian trees, the following lemma results.

LEMMA 5. *For two A-convex A-polygons consisting of a total of n vertices the minimum vertex A-distance and a pair of vertices realizing that distance can be computed in $O(n)$ time using $O(n)$ space, which is optimal in the worst case.*

Both lemmas together yield the following theorem.

THEOREM 3. *The A-distance between two A-convex A-polygons with a total of n vertices, together with a pair of closest points can be computed in $O(n)$ time using $O(n)$ space, which is optimal in the worst case.*

It is interesting to observe that the farthest pair of points (maximum distance) problem, which is harder than the closest pair of points (minimum distance) problem for Euclidean convex polygons and Euclidean metric, is no harder than the latter for A-convex A-polygons and even arbitrary finite sets of points in the A-plane. On the contrary it is conceptually simpler though its worst-case complexity is the same. This is due to the fact that the (minimum) A-distance between two points, p_1 and p_2, is the maximum of the A-distances between any B-oriented line through p_1 and a parallel

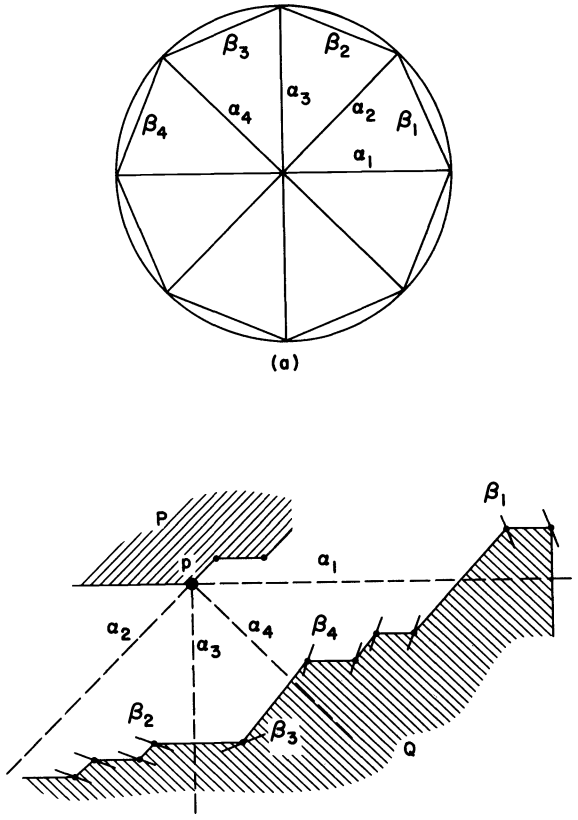


FIG. 11. *Vertex distance in an A-metric.*

line through p_2 (see Fig. 12(a) and (b)). The A-distance between two lines of orientation β_i is realized by a path with orientations α_i and α_{i+1}.

Hence, for any two sets P and Q of n_1 and n_2 points, we can determine the maximum distance from any point $p \in P$ to any point $q \in Q$ as follows. For both sets, P and Q, and for any $\beta_i \in B$, determine the two points in the set with maximum and minimum rank along the direction orthogonal to β_i, i.e., the points through which supporting lines with orientation β_i pass. Keep track of the maximal distance of any two supporting lines for any orientation β_i, one supporting P, the other supporting Q. This is the maximum distance between points from P and Q (see Fig. 12(c)).

THEOREM 4. *The maximum A-distance for two sets of altogether n points and a pair of points, one from each set, realizing that distance, can be computed in $O(n)$ time using $O(n)$ space, which is asymptotically optimal in the worst case.*

Proof. Correctness and efficiency follow from the discussion above; optimality is derived as in the proof of Lemma 1. □

COROLLARY 5. *The maximum A-distance between two A-polygons with a total of n vertices and a pair of farthest vertices can be computed in $O(n)$ time using $O(n)$ space, which is optimal in the worst case.*

Proof. Note that the maximum distance is always realized between a pair of vertices of the polygon boundaries. Then correctness and efficiency follow from Theorem 4, and the optimality as in the proof of Lemma 1. □

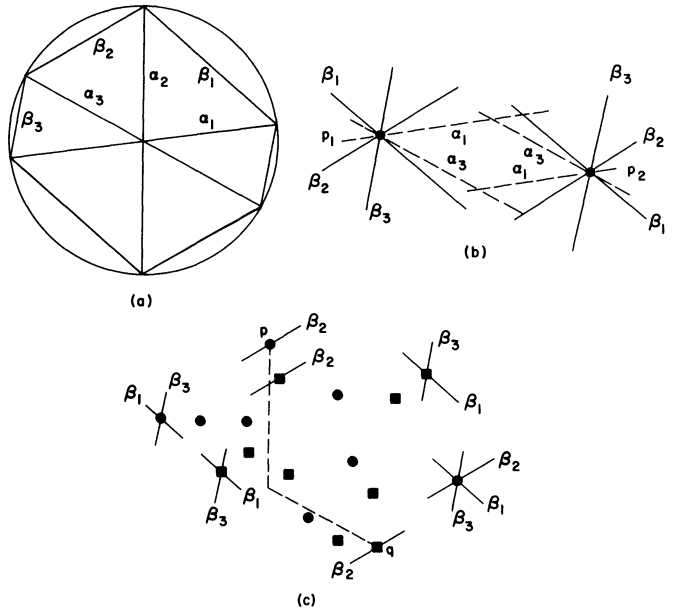


FIG. 12. *A-distance between two points, and the maximum A-distance between two sets of points.*

5. Conclusion. We have presented a new family of metrics, and a number of optimal solutions to some of the basic distance problems, including the computation of Voronoi diagrams, minimum spanning trees, minimum and maximum distances between convex polygons and other sets of points. This family of metrics is naturally induced by the restriction of the input and output of a computational geometry problem to a fixed set of orientations. Any distance in any metric of this family is, hence, realized by a path consisting of line segments of only the given orientations. A special case of such a restriction occurs in VLSI design, where orthogonally and diagonally oriented line segments can be processed with current fabrication technology.

The study of further distance problems in metrics of that family seems to be a promising field of research, considering the multitude of efficient algorithms available for the L_1- and L_2-metrics. In addition, it is interesting to investigate sets of fixed, but regularly distributed (over all angles) orientations, and approaching the Euclidean metric by pushing the number of regularly distributed orientations towards infinity.

REFERENCES

[1] B. K. BHATTACHARYA AND G. T. TOUSSAINT, *Efficient algorithms for computing the maximum distance between two finite planar sets*, J. Algorithms, 4 (1983), pp. 121–136.

[2] L. P. CHEW AND R. L. DRYSDALE, *Voronoi diagrams based on convex distance functions*, Proc. Symposium on Computational Geometry, Baltimore, MD, 1985, pp. 235–244.

[3] R. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, this Journal, 5 (1976), pp. 724–742.

[4] F. CHIN AND C. A. WANG, *Optimal algorithms for the intersection and the minimum distance problems between planar polygons*, IEEE Trans. Comput., C-32 (1983), pp. 1203–1207.

[5] ———, *Minimum vertex distance between separable convex polygons*, Inform. Process. Lett., 18 (1984), pp. 41–45.

[6] H. EDELSBRUNNER, *Intersection problems in computational geometry*, Ph.D. dissertation, Institute für Informationsverarbeitung, Graz, Austria, Report F93, 1982.

[7] ———, *On computing the extreme distances between two convex polygons*, J. Algorithms, 6 (1985), pp. 213–224.

[8] H. EDELSBRUNNER, L. J. GUIBAS AND J. STOLFI, *Optimal point location in a monotone subdivision*, this Journal, 15 (1986), pp. 317-340.

[9] H. N. GABOW, J. L. BENTLEY AND R. E. TARJAN, *Scaling and related techniques for geometry problems*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 135-143.

[10] R. H. GÜTING, *Conquering contours: efficient algorithms for computational geometry*, Ph.D. dissertation, University of Dortmund, West Germany, 1983.

[11] D. HAREL AND R. E. TARJAN, *Fast algorithms for finding nearest common ancestors*, this Journal, 13 (1984), pp. 338-355.

[12] F. HWANG, *An $O(n \log n)$ algorithm for rectilinear minimal spanning trees*, J. Assoc. Comput. Mach., 26 (1979), pp. 177-182.

[13] D. T. LEE, *Two-dimensional Voronoi diagrams in the L_p-metric*, J. Assoc. Comput. Mach., 27 (1980), pp. 604-618.

[14] D. T. LEE AND F. P. PREPARATA, *Computational geometry: a survey*, IEEE Trans. Comput., C-33 (1984), pp. 1072-1101.

[15] D. T. LEE AND C. K. WONG, *Voronoi diagrams in $L_1(L_\infty)$ metrics with 2-dimensional storage applications*, this Journal, 9 (1980), pp. 200-211.

[16] M. MCKENNA AND G. T. TOUSSAINT, *Finding the minimum vertex distance between two disjoint convex polygons in linear time*, Tech. Rep. No. SOCS 83.6, School of Computer Science, McGill University, Montreal, Canada, 1983.

[17] T. M. NICHOLL, D. T. LEE, Y. Z. LIAO AND C. K. WONG, *On the X-Y convex hull of a set of X-Y polygons*, BIT, 23 (1983), pp. 456-471.

[18] TH. OTTMANN, E. SOISALON-SOININEN AND D. WOOD, *On the definition and computation of rectilinear convex hulls*, Inform. Sci., 33 (1984), pp. 157-171.

[19] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[20] J. T. SCHWARTZ, *Finding the minimum distance between two convex polygons*, Inform. Process. Lett., 13 (1981), pp. 168-170.

[21] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, Proc. 16th Annual Symposium on Foundations of Computer Science, 1975, pp. 151-162.

[22] G. T. TOUSSAINT, *An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons*, Computing, 32 (1984), pp. 357-364.

[23] G. T. TOUSSAINT AND B. K. BHATTACHARYA, *Optimal algorithms for computing the minimum distance between two finite planar sets*, Pattern Recognition Letters, 2 (1983), pp. 79-82.

[24] G. T. TOUSSAINT AND J. A. MCALEAR, *A simple $O(n \log n)$ algorithm for finding the maximum distance between two finite planar sets*, Pattern Recognition Letters, 1 (1982), pp. 21-24.

[25] J. VUILLEMIN, *A unifying look at data structures*, Comm. ACM, 23 (1980), pp. 229-239.

[26] P. WIDMAYER, Y. F. WU, M. D. F. SCHLAG AND C. K. WONG, *On some union and intersection problems for polygons with fixed orientations*, Computing, 36 (1986), pp. 183-197.

[27] P. WIDMAYER, Y. F. WU AND C. K. WONG, *Distance problems in computational geometry with fixed orientations*, Proc. Symposium on Computational Geometry, Baltimore, MD, 1985, pp. 186-195.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics
010

TWO PROCESSOR SCHEDULING IS IN $\mathcal{NC}$ *

DAVID HELMBOLD† AND ERNST MAYR†

Abstract. We present a parallel algorithm for the two processor scheduling problem. This algorithm constructs an optimal schedule for unit execution time task systems with arbitrary precedence constraints using a polynomial number of processors and running in time polylog in the size of the input. Whereas previous parallel solutions for the problem made extensive use of randomization, our algorithm is completely deterministic and based on an interesting iteration technique. It is of independent relevance for two more reasons. It provides another example for the apparent difference in complexity between decision and search problems in the context of fast parallel computation, and it gives an $\mathcal{NC}$-algorithm for the matching problem in certain restricted cases.

Key words. scheduling, parallel computation, precedence constraints

AMS(MOS) subject classifications. 68G10, 68Q15, 68R05, 05C70

1. Introduction.

This paper presents results on parallel algorithms for scheduling problems. Our main result is a deterministic $\mathcal{NC}$ algorithm for solving the two processor scheduling problem. This problem falls into the general class of unit time scheduling problems with precedence constraints. The precedence constraints are given as a partial order on the tasks; if task t precedes task t' (written $t \prec t'$) then t must be completed before t' can be started. A solution to the problem is an optimal (i.e. shortest length) schedule indicating when each task is started.

Several results follow immediately from our main result. Since any optimal schedule corresponds to a maximum matching in the complement of the precedence graph, we have an $\mathcal{NC}$ algorithm which finds maximum matchings in the complements of precedence graphs (e.g. interval graphs and permutation graphs). For more details, see [9] and [13]. In addition, our algorithm solves the "obvious" open problem stated in [17].

For research into parallel algorithms, the two processor case is the most interesting unit time scheduling problem. When only a single processor is available (tasks must be scheduled one at a time) finding an optimal schedule is trivial. If the number of processors is an input to the problem, then the unit time scheduling problem becomes $\mathcal{NP}$-complete [16]. It is unknown whether or not there is a tractable sequential solution for a fixed number of processors greater than two.

The two processor scheduling problem has a long history and rich literature. The first polynomial time solution, $O(n^4)$, was published by Fujii, Kasami and Ninamiya in 1969 [3]. Three years later, Coffman and Graham published an $O(n^2)$ algorithm [1]. Gabow found an algorithm that, when combined with Tarjan's union-find result [5], runs in $O(n+e)$ time [4] and hence is asymptotically optimal. The only published parallel algorithm for the problem is Vazirani and Vazirani's randomized parallel solution [17]. The expected running time of their algorithm is a polynomial in the logarithm of the number of tasks.

Several researchers have considered restricting the precedence constraints and allowing the number of processors to vary. If the precedence constraints are restricted to forests [10],[8],[2] then optimal schedules can be found efficiently either sequentially

* Received by the editors March 10, 1986; accepted for publication October 2, 1986. This work was supported in part by a grant from the AT&T Foundation, Office of Naval Research contract N00014-85-C-0731 and National Science Foundation grant DCR-8351757.

† Department of Computer Science, Stanford University, Stanford, California 94305-2085.

or in parallel. If the precedence constraints are restricted to interval orders, then there is a sequential polynomial time algorithm for the problem [14].

With the rising use of highly parallel computers, it is important to identify those problems which can be efficiently solved in parallel. It is generally accepted that those problems in the class $\mathcal{NC}$ (solvable in poly-log time using a polynomial number of processors) are amenable to parallelization while those that are $\mathcal{P}$-complete (polynomial time complete under log-space reduction) are not. The class $\mathcal{RNC}$ consists of those problems solvable in poly-log time using a polynomial number of processors *with high probability* when random coin flips are available as a basic computation step. Algorithms in this class only need to quickly obtain the right answer most of the time.

One fundamental problem which has an $\mathcal{RNC}$ (but no known $\mathcal{NC}$) algorithm is the matching problem [12]. Our results on two processor scheduling provide evidence that the matching problem might be in $\mathcal{NC}$, since the two problems are closely related. An optimal two processor schedule is a maximum matching in $\bar{G}$, the complement of the precedence graph. Conversely, there is a sequential algorithm for converting any maximum matching in $\bar{G}$ into an optimal schedule for G [3].

Our $\mathcal{NC}$ algorithm for the two processor scheduling problem is an improvement over the aforementioned $\mathcal{RNC}$ result for two reasons. First, the algorithm in [17] is a randomized algorithm; even though the expected running time is poly-log, it may take an arbitrarily long time to halt. Secondly, their algorithm heavily relies on powerful $\mathcal{RNC}$ subroutines for computing maximum matchings and node sets of maximum matchings. In contrast, our algorithm is deterministic, does not require a matching subroutine, and contains some interesting parallel techniques.

Because our two processor scheduling algorithm is rather complex, we have divided its presentation into several stages. Our first algorithm simply computes the length of an optimal schedule. Using this algorithm we can locate the "empty slots" or holes in *lexicographically maximum jump* (LMJ) schedules. This in turn enables us to find the lexicographically maximum jump sequence. The final part of our presentation describes how tasks are assigned to jumps in the jump sequence. Once we have assigned tasks to each jump, it is easy to compute an optimal schedule.

2. Basic definitions. An instance of the two processor scheduling problem is a directed, acyclic *precedence graph*, $G = (\mathcal{T}, \prec)$, where $\mathcal{T}$ is the set of tasks and "$\prec$" reflects the precedence constraints between them. If $t \prec t'$ (read t precedes t') then t is a *predecessor* of t' and t' is a *successor* of t. We assume that G is transitively closed, if not its transitive closure can be computed using standard parallel techniques.

Schedules map tasks to integer timesteps. A schedule S is *valid* if S maps at most two tasks to any timestep and $S(t) < S(t')$ whenever $t \prec t'$. Unless stated otherwise, all schedules are assumed to be valid. Without loss of generality, we assume that the image of S is $\{1, \ldots, \omega_S\}$. Timestep T *contains* task t if $S(t) = T$. If a timestep contains only a single task, then that timestep also contains an *empty slot.* An optimal schedule is one which uses the fewest possible timesteps. The value $\mathrm{OPT}(G)$ is the number of timesteps used by optimal schedules for G.

The tasks of G can be partitioned into L *levels.* Task t is on *level* l if the longest path in G from t to a sink contains l nodes (counting both t and the sink). Precedence graphs are drawn with edges directed downwards (see Figure 1), so if level $l > l'$ we say that level l is above or *higher* than l' and l' is *below* or *lower* than l. Two tasks are *independent* if neither precedes the other. Note that every task precedes at least one task from each lower level and two tasks from the same level are always independent.

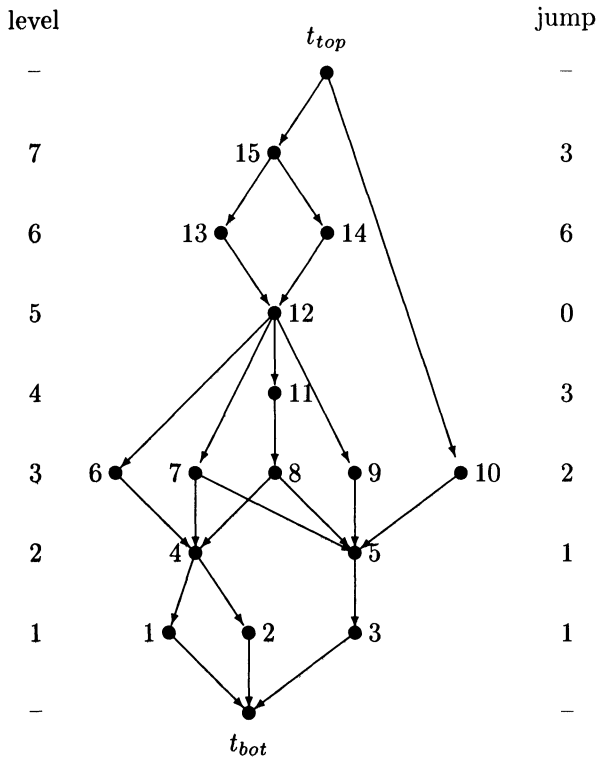


FIG. 1. *This is a precedence graph containing fifteen tasks (transitive arcs have been omitted). The special tasks* t_{top} *and* t_{bot} *are added when computing the length of optimal schedules for G. The levels of the original graph are on the left and the jump sequence is on the right.*

		X_1	X_2		X_3			X_4	X_5	
P_1	t_{top}	15	14	12	11	8	6	4	2	t_{bot}
P_2	–	10	13	–	9	7	5	3	1	–
timestep	1	2	3	4	5	6	7	8	9	10

FIG. 2. *This is an LMJ schedule for the graph in Figure* 1. *Each of the sets* X_i *is boxed.*

A *level schedule* maps the tasks on higher levels first. More precisely, suppose the tasks on levels $L, \ldots, l+1$ have already been mapped to timesteps and there are k unmapped tasks remaining on level l. If k is even the tasks are mapped to the next $k/2$ timesteps. If k is odd, the tasks are mapped to the next $(k+1)/2$ timesteps. The last of these $(k+1)/2$ timesteps will contain either a task from a lower level or an empty slot.

Given a level schedule, we say level l *jumps* to level $l' \leq l$ if the last timestep containing a task from level l also contains a task from level l'. If the last timestep

containing a task from level l has an empty slot, then level l jumps to level 0. The *jump sequence* of a level schedule is the list of levels jumped to (in order of level jumped from). Note that a level may appear in the jump sequence several times.

The *Lexicographically Maximum Jump (LMJ)* sequence is the jump sequence (of some level schedule) that is lexicographically greater than any other jump sequence resulting from a level schedule. An *LMJ schedule* is a level schedule whose jump sequence is the LMJ sequence (see Fig. 2).

THEOREM 1. *Every LMJ schedule is optimal* [4].

The jump from level l is *trivial* when l jumps to either itself or to level 0 (and is *nontrivial* otherwise). If the jump from level l is nontrivial and T is the last timestep containing a task from l then the two tasks mapped to timestep T are an *actual jump*. Two tasks are a *candidate pair* if the tasks form an actual jump in some LMJ schedule.

Note that:

- *jumps* occur between levels, and can be *trivial*.
- the *jump sequence* is a sequence of levels.
- an *actual jump* is a pair of tasks from different levels mapped to the same timestep.
- a *candidate pair* is an actual jump in some LMJ schedule.

The definition of LMJ schedule presented here is equivalent to the definition of *highest level first* schedule in [17] and [4]. Throughout the remainder of this paper we restrict our attention to LMJ schedules.

3. Computing the length of an optimal schedule. To compute the length of an optimal schedule for a precedence graph G we add two new tasks, t_{top} and t_{bot}. Task t_{top} is a predecessor, and t_{bot} a successor, of all other tasks. The modified graph is fed to the distance algorithm. Below we prove that the distance computed between t_{top} and t_{bot} is the length of optimal schedules for the original graph.

The *schedule distance* between tasks t and t', written $D(t,t')$, is the number of timesteps required to schedule all tasks that are both successors of t and predecessors of t'. If $t \not\prec t'$ then $D(t,t') = 0$.

LEMMA 2. *Let t and t' be any two tasks. If there are natural numbers j, k, and a nonempty set of tasks U such that for all $\hat{t} \in U$:*

— $t \prec \hat{t} \prec t'$,

— $\mathrm{D}(t,\hat{t}) \geq j$, *and*

— $\mathrm{D}(\hat{t},t') \geq k$;

then $\mathrm{D}(t,t') \geq j + k + \lceil |U|/2 \rceil$.

Proof. Let S be a valid schedule for those tasks which are successors of t and predecessors of t'. No task in U can be scheduled in the first j timesteps or the last k timesteps. Every task in U is both a successor of t and a predecessor of t', and thus is mapped to a timestep by S. Since at most two tasks are mapped to any timestep, at least $\lceil |U|/2 \rceil$ timesteps contain tasks from U. Therefore S uses at least $j+k+\lceil |U|/2 \rceil$ timesteps (see Figure 3). □

The distance algorithm (Figure 4) begins by guessing that the schedule distance between every pair of tasks is 0. In each iteration it attempts to improve its guesses by examining pairs of dependent tasks and applying Lemma 2. When the guesses converge to the schedule distances, the computed distance between t_{top} and t_{bot} is the length of optimal schedules for G. The distance algorithm has a straightforward $O(\log^2 n)$ time implementation on an n^5 processor P-RAM.

LEMMA 3. *The distance algorithm never computes a distance between two nodes larger than the schedule distance.*

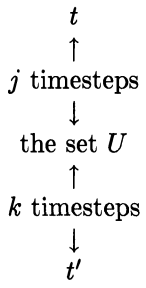


FIG. 3. *Timesteps separating tasks t and t'.*

$d_0(\star,\star) := 0;$
for $r := 1$ **to** $\lceil \log n \rceil$ **do**
  for all t, t' with $t \prec t'$ **do in parallel**
    for all $0 \le j, k < n-1$ **do in parallel**
      $U_{t,t',j,k} := \{\hat{t} : t \prec \hat{t} \prec t',$
        $d_{r-1}(t,\hat{t}) \ge j,$
        $d_{r-1}(\hat{t},t') \ge k\};$
    $d_r(t,t') := \max_{U_{t,t',j,k} \neq \emptyset}\{d_{r-1}(t,t'),$
      $j + k + \lceil |U_{t,t',j,k}|/2 \rceil\};$
$\mathrm{D}(\star,\star) := d_{\lceil \log n \rceil}(\star,\star)$

FIG. 4. *The distance algorithm.*

Proof. The algorithm computes each distance by the method in Lemma 2. Since this method always gives a lower bound on the schedule distance, the distances computed by the algorithm will never exceed the schedule distance. □

The proof that the guesses quickly converge to the proper distances requires a result from [1]. There it is shown how to construct sets of tasks $\chi_1, \ldots, \chi_k$ for any precedence graph such that:

— those tasks in any χ_i are predecessors of all tasks in χ_{i+1}, and
— $\mathrm{OPT}(G) = \sum_i \lceil |\chi_i|/2 \rceil$.

The distance algorithm does not explicitly compute these sets. We only make use of them for the proof of its correctness.

LEMMA 4. *After the main loop has been executed $\lceil \log n \rceil$ times, the distance $d_{\lceil \log n \rceil}(t_{\mathrm{top}}, t_{\mathrm{bot}})$ is at least $D(t_{\mathrm{top}}, t_{\mathrm{bot}}) = \mathrm{OPT}(G)$.*

Proof. Let $\chi_1, \chi_2, \ldots, \chi_k$ satisfy the above conditions for the original graph. Adding $\chi_0 = \{t_{\mathrm{top}}\}$ and $\chi_{k+1} = \{t_{\mathrm{bot}}\}$ creates a set of χ's which satisfies the conditions for the modified graph. Let $d_r(\chi_i, \chi_j)$ denote the least $d_r(t,t')$ where $t \in \chi_i$ and $t' \in \chi_j$. We now prove by induction on r that

$$d_r(\chi_l, \chi_{l+2^r}) \ge \sum_{l<i<l+2^r} \lceil |\chi_i|/2 \rceil .$$

The hypothesis holds after the first iteration since all members of χ_{l+1} are between any task in χ_l and any task in χ_{l+2}. Assume that the hypothesis holds after r iterations of the main loop. Examine how the distance between $t \in \chi_l$

and $t' \in \chi_{l+2^{r+1}}$ is modified in iteration $r+1$. When $j = \sum_{l<i<l+2^r} \lceil |\chi_i|/2 \rceil$ and $k = \sum_{l+2^r<i<l+2^{r+1}} \lceil |\chi_i|/2 \rceil$, the set $U_{t,t',j,k}$ contains all of χ_{l+2^r}. Therefore

$$\begin{aligned} d_{r+1}(\chi_l, \chi_{l+2^{r+1}}) &\geq \lceil |\chi_{l+2^r}|/2 \rceil + \sum_{l<i<l+2^r} \lceil |\chi_i|/2 \rceil + \sum_{l+2^r<i<l+2^{r+1}} \lceil |\chi_i|/2 \rceil \\ &\geq \sum_{l<i<l+2^{r+1}} \lceil |\chi_i|/2 \rceil . \end{aligned}$$

Since each χ contains at least one task, G contains at most n χ's. Thus $d_{\lceil \log n \rceil}(t_{\text{top}}, t_{\text{bot}}) \geq \text{OPT}(G)$. □

THEOREM 5. *Algorithm* 1 *correctly computes the length of the shortest schedule for the precedence graph.*

Proof. The claim follows immediately from Lemmas 3 and 4. □

It is easy to convert a polynomial-time algorithm that computes the length of an optimal schedule to one computing a particular optimal schedule. This is because we can try each possibility for the first timestep, and check if the remainder of the graph can be scheduled in one less timestep. Unfortunately this process is inherently sequential. We must wait until the first k timesteps are fixed before we attempt to fix timestep $k+1$.

The corresponding method for parallel algorithms would be to compute the tasks scheduled at the next timestep for each possible initial schedule. Unfortunately, there are exponentially many possible initial schedules. Therefore we must use additional structural properties of LMJ sequences to determine the actual schedule.

4. Computing the jump sequence. The first step in computing the LMJ sequence for G is to obtain a routine for determining which levels jump to level 0. When this routine is run on appropriate subgraphs of G, we can find the level jumped to by each level in G.

It is easy to determine if the bottom level jumps to level 0 in an LMJ schedule. Modify G by adding a new task to the bottom level which depends on all tasks not on level 1. If the length of optimal schedules for the modified graph is the same as $\text{OPT}(G)$ then LMJ schedules for G end with an empty slot (jump from level 1 to level 0). If the optimal schedule length of the modified graph is greater than $\text{OPT}(G)$ then level 1 jumps to itself in every optimal schedule for G. This modification does not affect the jumps of other levels since no task above level 1 can be mapped to the same timestep as the added task. This idea is used to prove the following lemma.

LEMMA 6. *Given a graph G with L levels, there is an $\mathcal{NC}$ algorithm for finding which levels jump to level* 0 *in the LMJ sequence.*

Proof. For each pair $(l,e), 0 \leq e \leq l \leq L$, we create the graph $G_{l,e}$. Each graph $G_{l,e}$ consists of G plus e additional task. Every task above level l in $G_{l,e}$ has a precedence arc to each of the e additional tasks.

The length of an optimal schedule for $G_{l,e}$ will be the same as the length of the optimal schedule for G only when all of the e added tasks are in jumps from levels that used to jump to level 0. Therefore by determining the largest e such that $\text{OPT}(G_{l,e}) = \text{OPT}(G)$ we can find the number of levels at or below level l that jump to level 0 in LMJ schedules for G. Those values of l where this changes are precisely those levels that jump to 0 in LMJ schedules for G. □

The next step is to convert the algorithm for finding jumps to level 0 into one which finds all of the jumps. Say level l jumps to l'. When level l' and all lower levels

are removed, level l will jump to level 0. By deleting levels until level l jumps to level 0, one can determine which level l jumps to in the original graph.

LEMMA 7. *Given a precedence graph G, there is an $\mathcal{NC}$ algorithm for finding the LMJ sequence.*

Proof. Find the jumps to level 0 in LMJ schedules for the graphs $G_L, G_{L-1}, \ldots, G_1$ where G_l contains only the nodes on levels L down to l (and the precedence constraints between them). If level l does not jump to level 0 in G_l then (l,l) is in the LMJ sequence for G. If level l jumps to level 0 in G_1 then $(l,0)$ is in the LMJ sequence for G. Otherwise, for some $0 < i < l$, the jump $(l,0)$ occurs in G_{i+1} but does not show up in G_i, and level l jumps to level i in the LMJ sequence for G. □

5. Determining the candidate pairs. An important step in constructing an LMJ schedule is to determine whether or not a pair of tasks is a candidate pair (recall that candidate pairs are the actual jumps of LMJ schedules). The following lemma justifies our emphasis on candidate pairs.

LEMMA 8. *Any set of disjoint candidate pairs (one for each nontrivial jump) forms the actual jumps of an LMJ schedule.*

Proof. Assume we have a set of disjoint candidate pairs which do not form the actual jumps of any LMJ schedule. Consider the schedule S constructed as in the definition of level schedule using the set of candidate pairs for the actual jumps. Since S is an LMJ schedule which maps at most two tasks to each timestep, it must violate the precedence constraints.

Let T be the first timestep containing tasks in violation of the precedence constraints. Because of the way S was constructed, T contains some candidate pair (t,t') for the jump (l,l'). Let $\hat{t}$ be a task such that: $\hat{t} \prec t'$, $S(\hat{t}) > T$, and all tasks which precede task $\hat{t}$ are mapped to timesteps before T. Tasks t and $\hat{t}$ must be independent, else t would precede t' by transitivity. Therefore tasks t and $\hat{t}$ can both be mapped to timestep T without disturbing the earlier part of the schedule. Since task $\hat{t}$ is on a level above l', the jump (l,l') does not occur in LMJ schedules – contradiction. □

It is necessary that tasks t and t' be independent and from appropriate levels for them to be a candidate pair. Unfortunately this is not sufficient. Task t may be needed for the actual jump from a higher level or t' may be required for the actual jump from level l'. For example, in Figure 1 tasks 10 and 4 are not a candidate pair.

An obvious attempt to determine if tasks $t \in l$ and $t' \in l'$ form a candidate pair is to delete them and examine the resulting LMJ sequence. The hope is that the LMJ sequence for $G = (\mathcal{T}, \prec)$ is identical to the LMJ sequence for $G' = (\mathcal{T} - \{t,t'\}, \prec)$ (with the exception of the jump from level l) exactly when t and t' are a candidate pair. This method fails. When t and t' are deleted tasks may move to different levels, so the LMJ sequences for G and G' are incomparable.

A more complicated construction is needed to determine if $t \in l$ and $t' \in l'$ are a candidate pair. Our construction creates a new level between levels l and $l-1$. In the first step, task t is moved to the new level while dummy tasks are used to maintain the level structure in the rest of the graph. In the second step, task t' is moved up to the new level. This may require that other tasks are also moved up to the new level. It is shown below (Lemma 9) that any other tasks which get moved up to the new level are used in actual jumps from levels above l. By examining how the LMJ sequence changes during the construction, we can determine if (t,t') is a candidate pair.

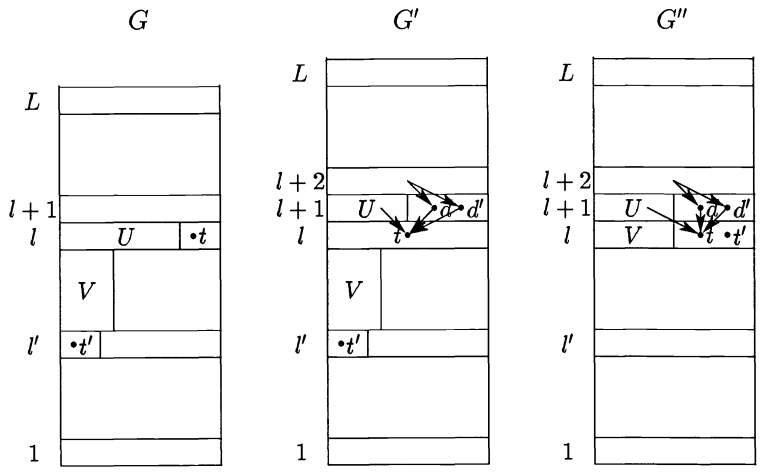


FIG. 5. *We use this construction to check if the jump* $(t, t') \in (l, l')$ *is part of any LMJ schedule. The set* U *contains the tasks on level* l*, except for* t*. The set* V *contains those tasks on levels between* l *and* l' *which are* not *successors of* t*.*

Figure 5 illustrates the construction. Graph G is the original precedence graph. The set U consists of all tasks on level l (except for task t). The set V consists of those tasks on levels between l and l' which are *not* successors of t.

The graph G' is created from G by the following procedure:

— Create two dummy tasks, d and d'.

— Make t a successor of d, d', and the tasks in U.

— Make d and d' successors of all tasks on level $l+1$ of G.

The effect of this procedure is to "move up" level l and every higher level. The new level l contains only task t. The level $l+1$ in G' contains those tasks that were in level l (except for t) and the two dummy tasks. Level $l+2$ in G' contains exactly those tasks on level $l+1$ in G and so on. Two dummy task are required to maintain the parity on level $l+1$ in G'. Note that in G' every task above level $l+1$ is a predecessor of d, d', and t.

The LMJ sequences for G and G' are *similar* if: the jump $(l+1, l+1)$ is in the sequence for G' and whenever the jump (j, k) is in the sequence for G, then the jump (j', k') is in the sequence for G'. Here j' (k') stands for $j+1$ $(k+1)$ if $j > l$ $(k > l)$ and j (k) otherwise.

LEMMA 9. *If the LMJ sequences for* G *and* G' *are similar, then any LMJ schedule for* G' *completes all of the tasks in* V *before level* l *is finished.*

Proof. Otherwise level l would be able to jump to a level above l' in LMJ sequences for G. □

LEMMA 10. *There is an LMJ schedule for* G *where task* t *is in the actual jump from level* l *if and only if the LMJ sequences for* G *and* G' *are similar.*

Proof. Assume G has an LMJ schedule where t is the last task on level l. By inserting a timestep for the two dummy tasks we get a similar LMJ schedule for G'. Assume the LMJ sequences for G and G' are similar. Since level $l+1$ jumps to itself, and no task can jump to either of the dummy tasks, there is an LMJ schedule, S, for G' which maps the two dummy tasks to the same timestep. The schedule obtained by deleting this timestep from S is an LMJ schedule for G. □

The graph G'' is created from G' as follows:
- Delete all precedence constraints between tasks in $V \cup \{t'\}$.
- Let s be some successor of t on level $l-1$; make every task in $V \cup \{t'\}$ a predecessor of s.
- Make t' a successor of both d and d'.

The LMJ sequences for graphs G' and G'' are *similar* if:
- the jump (l, l') is in the sequence for G',
- the jump (l, l) is in the sequence for G'',
- when the jump (j, k), $j > l, k \in [l'+1, l-1]$, is in the sequence for G' the jump (j, l) is in the sequence for G'', and
- all other jumps in G' are also in G''.

THEOREM 11. *The LMJ sequences for G, G' and G'' are similar if and only if tasks t and t' are an actual jump in some LMJ schedule for G.*

Proof. ($\Rightarrow$) Assume that the LMJ sequences for G, G' and G'' are similar and no LMJ schedule for G maps t and t' to the same timestep. By Lemma 10, G has an LMJ schedule where t is used for the actual jump from level l. Since t and t' are independent, t' must be needed for some other jump. All of the other jumps to/from level l' in G are also present in G''. Therefore t' must be used for one of them in G''. However, t' is no longer on level l' in G'', a contradiction.

($\Leftarrow$) Let S be an LMJ schedule for G which uses the pair (t, t') for the actual jump from level l. Adding a timestep containing the dummy tasks makes S an LMJ schedule for G' and G''. Thus G, G' and G'' all have similar LMJ sequences. □

All of the candidate pairs can be found using the above theorem. Simply use the construction to test each pair of tasks in parallel.

6. Parallel selection of candidate pairs. In the previous section we presented a method for determining if a pair of tasks from different levels is a candidate pair. However, a disjoint set of candidate pairs (one for each nontrivial jump) must still be found. This problem is similar to three dimensional matching since we are trying to find a maximum set of nonoverlapping tuples $((l, l'), t, t')$ where (t, t') is a candidate pair for the jump (l, l'). Since general three-dimensional matching is $\mathcal{NP}$-complete [6], the particular structure of our problem must be exploited.

The three-dimensional matching problems that arise are special in two ways. First, the jumps form a tree structure. The *jump tree* is a representation of the LMJ sequence. Nodes in the jump tree correspond to levels in the precedence graph. Each nontrivial jump in the LMJ sequence is represented by an arc in the jump tree.

When the jump tree has two or more disjoint components, candidate pairs for the jumps in each component can be selected independently. Similarly, if levels $l_1, l_2, \ldots, l_k$ all jump to level l then the candidate pairs for the subtrees rooted at the l_i's can be selected independently as long as no task on level l is used twice.

The tasks also have exploitable structure. Each task belongs to a single level and can only be used for jumps to/from that level. Furthermore, the tasks within a level can be ordered based on the candidate pairs they are in. Using this ordering we shall see that only two candidate pairs need to be considered for each jump (see Fig. 6).

The *H-value* of task t from level l, written $H(t)$, is the highest level $\hat{l} > l$ such that t is in a candidate pair for the jump $(\hat{l}, l)$. If t is not in any candidate pair for a jump to level l, then $H(t) = 0$. The tasks on each level are assumed to be sorted by H-value (with ties broken arbitrarily).

LEMMA 12. *Let level l jump to l', task $t \in l$ be in a candidate pair for the jump (l, l'), and task $t' \in l'$ have $H(t') > l$. Then tasks t and t' are independent.*

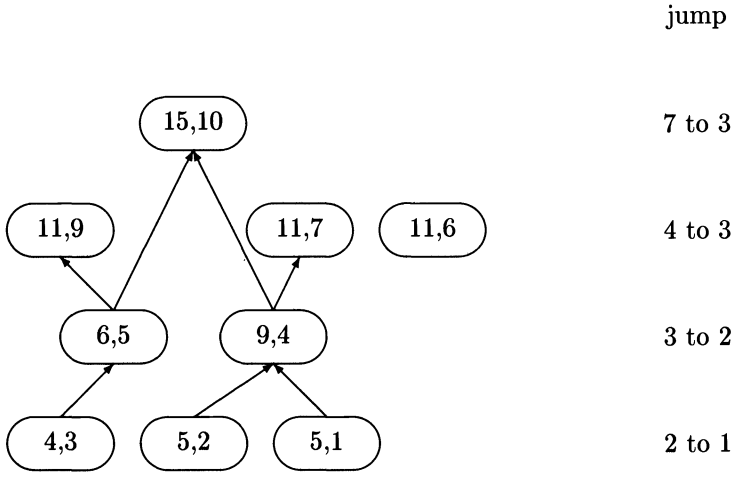


FIG. 6. *Here are the candidate pairs for the graph in Figure* 1. *Implications between the candidate pairs are represented by arcs. Candidate pair* $(4,3)$ *is implied because its level has a trivial jump.*

Proof. Let $\hat{t} \in H(t')$ be in a candidate pair with t'. Task $\hat{t}$ must precede t, else they could be mapped to the same timestep and level $H(t')$ would jump to level l in LMJ schedules, not level l'. Since $\hat{t}$ and t' are independent, no successor of $\hat{t}$ precedes t'. In particular, $t \not< t'$ so t and t' are independent. □

Let levels $l_1 > l_2 > \ldots > l_r$ be the levels jumping to level l. A *landing set* for level l is an ordered set of tasks, $\{t_1, t_2, \ldots, t_r\}$, such that for each $1 \leq i \leq r$, $H(t_i) \geq l_i$. A landing set for level l is *stable* if the jump from level l is trivial or there is some $t_0 \in l$ which is in a candidate pair for the jump from level l and not in the landing set.

LEMMA 13. *If $t \in l$ is in a candidate pair for the jump from level l then there is a landing set for level l which does not contain t.*

Proof. Let S be an LMJ schedule where t is in the actual jump from level l. Let $l_1 > \ldots > l_r$ be the levels jumping to level l and t_i be the task on level l used for the jump from level l_i. Obviously, $H(t_i) \geq l_i$, so $\{t_1, \ldots, t_r\}$ is a landing set for l. □

LEMMA 14. *Let levels $l_1 > l_2 > \ldots > l_r$ be the levels jumping to level l and $\{t_1, \ldots, t_r\}$ be a stable landing set for level l. Then there is an LMJ schedule where each t_i, $1 \leq i \leq r$, is in the actual jump from level l_i to l.*

Proof. Since the landing set is stable there is a $t_0 \in l$ which is in a candidate pair for the jump from level l and not in the landing set. By Lemma 12, for $1 \leq i \leq r$ there is a $\hat{t}_i \in l_i$ which is both in a candidate pair for the jump (l_i, l) and is independent with t_i. Because the subtrees rooted at the l_i's are independent, there is an LMJ schedule where:

— task t_0 is in the actual jump from level l, and
— the actual jump from each level l_i is $(\hat{t}_i, t_i)$.

Therefore each t_i is in a candidate pair for the jump from l_i. □

Suppose the jumps to level l are from levels $l_1 > \ldots > l_r$. Let t_i be the ith task in the H-ordering of l. If either level l has a trivial jump or some $t_j \in l$ following t_i (in H-order) is in a candidate pair for the jump from level l then t_i is the ith task in a stable landing set for l. By Lemma 14, t_i is in a candidate pair for the jump (l_i, l).

Let task $c(t_i)$ be the least task (in H-order) on level l_i in a candidate pair with t_i. The candidate pair $(c(t_i), t_i)$ is the *primary* candidate pair for the jump (l_i, l).

Similarly, if some t_j before t_i in the H-order is in a candidate pair for the jump from level l then t_i is the $i-1$st task in a stable landing set for l. Let $c'(t_i)$ be the least task on level l_{i-1} which is in a candidate pair with t_i. The candidate pair $(c'(t_i), t_i)$ is the *secondary* candidate pair for the jump (l_{i-1}, l).

Implications can be defined between the primary and secondary candidate pairs. These implications are used to select a disjoint set of candidate pairs, one for each nontrivial jump. The implications are bottom up in the sense that candidate pairs for the jump from level l imply candidate pairs for the jumps to level l. Let levels $l_1 > \ldots > l_r$ be the levels jumping to level l and $t_1, \ldots, t_k$ be the tasks on level l in H-order. There are three cases depending on the jump from level l:

- If level l has a trivial jump then the primary candidate pairs for each jump to level l are implied.
- If t_j, $j > r$, is in the implied candidate pair for the jump from level l then the primary candidate pairs are implied.
- If t_j, $j \leq r$, is in the implied candidate pair for the jump from level l then the primary candidate pairs for those jumps from levels $l_1, \ldots, l_{j-1}$ are implied and the secondary candidate pairs for the jumps from $l_j, \ldots, l_r$ are implied.

LEMMA 15. *The set of implied candidate pairs is disjoint and contains exactly one candidate pair for each nontrivial jump.*

Proof. By induction using the hypothesis that a disjoint candidate pair is implied for all jumps to levels below l.

Case 1. Level l has a trivial jump (this case covers level 1). Only the primary candidate pairs are implied for the jumps to level l. There is one primary candidate pair for each jump to level l and they are disjoint.

Case 2. Level l makes a nontrivial jump to a lower level l'. By the induction hypothesis one (the jth) task from level l, t_j, is in the implied candidate pair for the jump (l, l'). Let $l_1 > \ldots > l_r$ be the levels jumping to level l. If $j > r$ then the primary candidate pairs are implied for the jumps to level l. Since t_j is not in any of the primary candidate pairs, this reduces to the previous case.

If $j \leq r$ then the primary candidate pairs for jumps from levels $l_1, \ldots, l_{j-1}$, and the secondary candidate pairs for jumps from levels $l_j, \ldots, l_r$ are implied. These candidate pairs are disjoint and do not use task t_j. Therefore there is one implied candidate pair for each jump to a level below $l+1$ and the implied candidate pairs for these jumps are disjoint. □

Standard parallel techniques such as transitive closure can be used to determine the implied candidate pairs. The total requirements for selecting a disjoint set of candidate pairs are:

H-orders:	$O(\log n)$ time on n^2 processors
$c(t), c'(t)$:	$O(\log n)$ time on n^2 processors
implication graph:	constant time on n^2 processors
path doubling:	$O(\log^2 n)$ time on n^3 processors.

Once disjoint candidate pairs have been assigned to the jumps, the remainder of the schedule can be completed in $O(\log n)$ time using n^2 processors.

THEOREM 16. *There is an $\mathcal{NC}$ algorithm which finds a two processor LMJ schedule for any precedence graph.*

Proof. This paper contains one such algorithm. □

Our algorithm is intended to show that two processor scheduling is in $\mathcal{NC}$; it is not intended to be an efficient solution. Computing the LMJ sequence for a graph G uses L^3 calls to the schedule length algorithm in parallel (recall that L is the number of levels in G). Determining if a pair of tasks is a candidate pair basically involves computing two LMJ sequences. Thus the total requirements for computing all of the candidate pairs is $n^2L^3n^5$ processors and $O(\log n \log L)$ time. The cost of assigning disjoint candidate pairs to jumps is dwarfed by the cost of computing the candidate pairs. Therefore, the requirements for the entire algorithm are n^7L^3 processors and $O(\log n \log L)$ time.

7. Conclusions. Our two processor scheduling result, coupled with a transitive orientation algorithm, allows us to solve several restrictions of the maximum matching problem. By exploiting the special relationship between two processor scheduling and matching, we can find maximum matchings on undirected graphs whose complements have a transitive orientation. Specifically, if G is an interval graph, then $\bar{G}$ has a transitive orientation [7]. Similarly, G is a permutation graph if and only if both G and $\bar{G}$ are transitively orientable [15]. Thus we have a deterministic $\mathcal{NC}$ maximum matching algorithm for both interval graphs and permutation graphs [9],[13].

There are many variations of the fundamental two processor scheduling problem. The tasks can have varying execution times, release times, or deadlines. If tasks have small integer execution times and are preemptable, then the problem reduces to the unit time scheduling problem. So far, our attempts at extending the algorithm to handle deadlines and/or release times have been unfruitful.

One variant of the two processor problem that we know to be $\mathcal{NP}$-complete (under log-space reduction) allows incompatibility edges as well as precedence constraints. When there is an incompatibility constraint between two tasks they can be executed in either order, but not concurrently. Incompatibility constraints arise naturally when two or more tasks need the same resource, such as special purpose hardware or a database file.

It was surprising how much more difficult computing the actual schedule was than simply computing its length. In higher complexity classes such as $\mathcal{P}$ and $\mathcal{NP}$ it is often easy to go from the decision problem to computing an actual solution, because of self-reducibility. However this does not seem to necessarily be the case for parallel complexity classes. To support this observation we note that the random $\mathcal{NC}$ algorithm for finding the cardinality of a maximum matching is much simpler than the random $\mathcal{NC}$ algorithm for determining an actual maximum matching [11].

REFERENCES

[1] E. COFFMAN, JR. AND R. GRAHAM, *Optimal scheduling for two processor systems*, Acta Informatica, 1 (1972), pp. 200–213.

[2] D. DOLEV, E. UPFAL AND M. WARMUTH, *Scheduling trees in parallel*, in VLSI: Algorithms and Architectures. Proceedings of the International Workshop on Parallel Computing and VLSI, Bertolazzi, P. Luccio, F., eds., North-Holland, 1985, pp. 1–30.

[3] M. FUJII, T. KASAMI AND K. NINAMIYA, *Optimal sequencing of two equivalent processors*, SIAM J. Appl. Math., 17 (1969), pp. 784–789.

[4] H. GABOW, *An almost-linear algorithm for two-processor scheduling*, J. Assoc. Comput. Mach., 29 (1982), pp. 766–780.

[5] H. GABOW AND R. TARJAN, *A linear time algorithm for special case of disjoint set union*, in Proceedings of the 15th Ann. ACM Symp. on Theory of Computing, Boston, MA, 1983, pp. 246–251.

[6] M. GAREY AND D. JOHNSON, *Computers and Intractability, A Guide to the Theory of $\mathcal{NP}$-completeness*, W.H. Freeman, San Francisco, CA, 1979.
[7] A. GHOUILÀ-HOURI, *Caractérisation des graphes non orientés dont on peut orienter les arrêtes de manière à obtenir le graphe d'une relation d'ordre*, C.R. Acad. Sci. Paris, 254 (1962).
[8] D. HELMBOLD AND E. MAYR, *Fast scheduling algorithms on parallel computers*, Advances in Computing Research, (1986), to appear.
[9] ———, *Perfect graphs and parallel algorithms*, in Proceedings of the IEEE 1986 International Conference on Parallel Processing, August 1986.
[10] T. HU, *Parallel sequencing and assembly line problems*, Operations Research, 9 (1961), pp. 841–848.
[11] R. KARP, E. UPFAL AND A. WIGDERSON, *Are search and decision problems computationally equivalent?*, in Proceedings of the 17th Ann. ACM Symp. on Theory of Computing, Providence, RI, 1985, pp. 465–475.
[12] ———, *Constructing a perfect matching is in random $\mathcal{NC}$*, in Proceedings of the 17th Ann. ACM Symp. on Theory of Computing, Providence, RI, 1985, pp. 22–32.
[13] D. KOZEN, U. VAZIRANI AND V. VAZIRANI, *$\mathcal{NC}$ algorithms for comparability graphs, interval graphs, and testing for unique perfect matching*, in 5th Conf. Found. of Software Tech. and Theor. Comp. Sci., New Dehli, India, 1985.
[14] C. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval-ordered tasks*, this Journal, 8 (1979), pp. 405–409.
[15] A. PNUELI, A. LEMPEL AND S. EVEN, *Transitive orientation of graphs and identification of permutation graphs*, Can. J. Math., 23 (1971), pp. 160–175.
[16] J. ULLMAN, *$\mathcal{NP}$-complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.
[17] U. VAZIRANI AND V. VAZIRANI, *The two-processor scheduling problem is in $\mathcal{RNC}$*, in Proceedings of the 17th Ann. ACM Symp. on Theory of Computing, Providence, RI, 1985, pp. 11–21.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987
© 1987 Society for Industrial and Applied Mathematics
011

LANGUAGES THAT CAPTURE COMPLEXITY CLASSES*

NEIL IMMERMAN†

Abstract. We present a series of operators of apparently increasing power which when added to first-order logic produce a series of languages in which exactly the properties checkable in a certain complexity class may be expressed. We thus give alternate characterizations of most important complexity classes. We also introduce reductions appropriate for our setting: first-order translations, and a restricted, quantifier free version of these called projection translations. We show that projection translations are a uniform version of Valiant's projections, and that the usual complete problems remain complete via these very restrictive reductions.

Key words. first-order expressibility, computational complexity, first-order translations, projection translations

AMS(MOS) subject classification. 68025

Introduction. We present in this paper a series of languages adequate for expressing exactly those properties checkable in a series of computational complexity classes. For example, we show that a property of graphs (respectively groups, binary strings, etc.) is in polynomial time if and only if it is expressible in the first order language of graphs (respectively groups, binary strings, etc.) together with a least fixed point operator. As another example, a property is in logspace if and only if it is expressible in first order logic together with a deterministic transitive closure operator.

The roots of our approach to complexity theory go back to 1974 when Fagin showed that the NP properties are exactly those expressible in second order existential sentences. It follows that second order logic expresses exactly those properties which are in the polynomial time hierarchy. We show that adding suitable transitive closure operators to second order logic results in languages capturing polynomial space and exponential time, respectively.

The existence of such natural languages for each important complexity class sheds a new light on complexity theory. These languages reaffirm the importance of the complexity classes as much more than machine dependent issues. Furthermore a whole new approach is suggested. Upper bounds (algorithms) can be produced by expressing the property of interest in one of our languages. Lower bounds may be demonstrated by showing that such expression is impossible.

For example, from the above we know that $P = NP$ if and only if every second order property is already expressible using first order logic plus least fixed point. Similarly nondeterministic logspace is different from P just if there is some sentence using the fixed point operator which cannot be expressed with a single application of transitive closure.

In previous work [Im81], [Im82b], we showed that the complexity of a property is related to the number of variables and quantifiers in a uniform sequence of sentences, $\varphi_1, \varphi_2 \ldots$, where each φ_n expresses the property for structures of size n. Our present formulation is more pleasing because it considers single sentences (in more powerful languages).

* Received by the editors August 29, 1985; accepted for publication (in revised form) September 5, 1986. A preliminary version of this paper appeared in the Proceedings of the 15th Annual ACM Symposium on the Theory of Computing. This research was supported by a National Science Foundation postdoctoral fellowship and by National Science Foundation grant DCR-8603346.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

The first order expressible properties at first seemed too weak to correspond to any natural complexity class. However we found that a property is expressible by a sequence of first order sentences, $\varphi_1, \varphi_2 \ldots$, where each φ_n has a bounded number of quantifiers if and only if this property is recognized by a similar sequence of polynomial size boolean circuits of bounded depth. It follows that the results of Furst, Saxe, and Sipser, [FSS81], and Sipser, [Si83], translate precisely into a proof that certain properties are not expressible in any first order language.

In this paper we also introduce a reduction between problems that is new to complexity theory. *First order translations*, as the name implies, are fixed first order sentences which translate one kind of structure into another. This is a very natural way to get a reduction, and at the same time it is very restrictive. It seems plausible to prove that such reductions do not exist between certain problems. We present problems which are complete for logspace, nondeterministic logspace, polynomial time, etc., via first order translations.

This paper is organized as follows: Section 1 introduces the complexity classes we will be considering. Section 2 discusses first order logic. Sections 3, 4 and 5 introduce the languages under consideration. Section 6 considers the relationship between first order logic and polynomial size, bounded depth circuits. The present (lack of) knowledge concerning the separation of our various languages is discussed.

1. Complexity classes and complete problems. In this section we define those complexity classess which we will capture with languages in the following sections. We list complete problems for some of the classes. In later sections we will show how to express these complete problems in the appropriate languages; and, we will also show that the problems are complete via first order translations. More information about complexity classes may be found in [AHU74].

Consider the following well-known sequence of containments:

$$L \subseteq \mathrm{NL} \subseteq \Sigma_* L \subseteq P \subseteq \mathrm{NP} \subseteq \Sigma_* P.$$

Here L is deterministic logspace and NL is nondeterministic logspace. $\Sigma_* L = \bigcup_{k=1}^{\infty} \Sigma_k L$ is the logspace hierarchy. Here $\Sigma_k L$ is the class of problems accepted by alternating logspace Turing machines that make $k-1$ alternations between existential and universal states, beginning in an existential state. $\Sigma_* P = \bigcup_{k=1}^{\infty} \Sigma_k P$ is the polynomial time hierarchy, [St77]. Most knowledgeable people suspect that all of the classes in the above containment are distinct, but it is not known that they are not all equal.

We begin our list of complete problems with the graph accessibility problem (GAP). We will consider a *graph* to be a directed graph with vertices $v_0, v_1, v_2, \ldots, v_{\max}$. An *undirected graph* is just a graph whose edge relation is symmetric. Define

$$\mathrm{GAP} = \{G | \exists \text{ a path in } G \text{ from } v_0 \text{ to } v_{\max}\}.$$

THEOREM 1.1. [Sa73]. GAP *is logspace complete for* NL.

We will see later that GAP is complete for NL in a much stronger sense. The GAP problem may be weakened to a deterministic logspace problem by only considering those graphs which have at most one edge leaving any vertex:

$$\mathrm{1GAP} = \{G | G \text{ has outdegree 1 and } \exists \text{ a path in } G \text{ from } v_0 \text{ to } v_{\max}\}.$$

THEOREM 1.2. [HIM78]. 1GAP *is one-way logspace complete for* L.

A problem which lies between 1GAP and GAP in complexity is

$$\text{UGAP} = \{G | G \text{ undirected and } \exists \text{ a path in } G \text{ from } v_0 \text{ to } v_{\max}\}.$$

Let BPL (bounded probability, logspace) be the set of problems, S, such that there exists a logspace coin-flipping machine, M, and if $w \in S$ then Prob(M accepts w) $> 2/3$, while if $w \notin S$ then Prob(M accepts w) $< 1/3$. It follows from the next theorem that UGAP is in BPL. Thus UGAP is probably easier than GAP.

THEOREM 1.3. [AKLLR79]. *If r is a random walk of length $2|E|(|V|+1)$ in an undirected connected graph G then the probability that r includes all vertices in G is greater than or equal to one half.*

Lewis and Papadimitriou [LP82] define *symmetric machines* to be nondeterministic Turing machines whose next move relation on instantaneous descriptions is symmetric. That is if a symmetric machine can move from configuration A to configuration B then it is also allowed to move from B to A. Note that it is not immediately obvious how to build Turing machines that are symmetric in the above sense. Lewis and Papadimitriou show that the definition makes sense. Let Sym-L be the class of problems accepted by symmetric logspace machines. Then we have

THEOREM 1.4. [LP82].

1. L $\subseteq$ Sym-L $\subseteq$ NL.
2. UGAP *is logspace complete for* Sym-L.

John Reif [Re84] extended the notion of symmetric machines to allow alternation. Essentially an alternating symmetric machine has a symmetric next move relation except where it alternates between existential and universal states. Let $\Sigma_*\text{Sym-L} = \bigcup_{k=1}^{\infty} \Sigma_k\text{Sym-L}$ be the *symmetric logspace hierarchy*. Reif showed that several interesting properties, including planarity for graphs of bounded valence, are in the symmetric logspace hierarchy. It follows that they are also in BPL.

Reif also showed that BPL is contained in $O[\log n]$ time and $n^{O[1]}$ processors on a probabilistic hardware modification machine (HMM).

THEOREM 1.5. [Re84].

$$\Sigma_*\text{Sym-L} \subseteq \text{BPL} \subseteq \text{Prob-HMM-TIME}[\log n],\text{PROC}[n^{O[1]}].$$

Let us prove for example that Σ_2Sym-L is contained in BPL. Given a Σ_2Sym-L machine M and its input, our BPL machine can cycle through all of M's polynomially many universal configurations, a, checking to see that:

1. Configuration a is reachable from the start configuration by a series of exclusively existential moves, and
2. No final, rejecting configuration is reachable from a.

If such a configuration a is found then M accepts, otherwise it rejects. Furthermore, properties (1) and (2) are UGAP questions and therefore answerable by our BPL machine.

One may also consider harder versions of the GAP problem. Let an *alternating graph* $G = (V, E, A)$ be a directed graph whose vertices are labelled universal or existential. $A \subset V$ is the set of universal vertices. Alternating graphs have a different notion of accessibility. Let APATH(x,y) be the smallest relation on vertices of G such that:

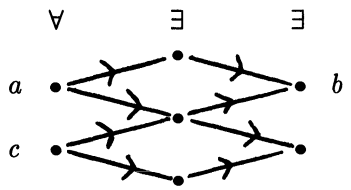


FIG. 1. *An alternating graph.*

1. APATH(x,x).
2. If x is existential and for some edge $\langle x, z\rangle$ APATH(z,y) holds, then APATH(x,y).
3. If x is universal, there is at least one edge leaving x, and for all edges $\langle x, z\rangle$ APATH(z,y) holds, then APATH(x,y).

See Fig. 1 where APATH(a,b) holds, but APATH(c,b) does not. Let

$$\text{AGAP} = \{G | \text{APATH}_G(v_0, v_{\max})\}.$$

It is not hard to see that AGAP is the alternating version of GAP, and thus is complete for ASPACE[log(n)]. Recalling that this class is equal to P [CKS81], we have

THEOREM 1.6. [Im81]. AGAP *is logspace complete for* P.[1]

2. First order logic. In this section we introduce the necessary notions from logic. The reader is referred to [En72] for more background material.

A *vocabulary* $\tau = \langle \underline{R}_1^{a_1} \ldots \underline{R}_k^{a_k}, \underline{c}_1 \ldots \underline{c}_r \rangle$ is a tuple of relation symbols and constant symbols. $\underline{R}_i^{a_i}$ is a relation symbol of arity a_i. In the sequel we may sometimes omit the superscripts and the underlines to improve readability. A finite *structure* with vocabulary τ is a tuple, $\mathcal{A} = \langle \{0, 1, \ldots, n-1\}, R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}, c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}} \rangle$, consisting of a universe $|\mathcal{A}| = \{0, \ldots, n-1\}$ and relations $R_1^{\mathcal{A}} \ldots R_k^{\mathcal{A}}$ of arities $a_1, \ldots, a_k$ on $|\mathcal{A}|$ corresponding to the relation symbols $\underline{R}_1^{a_1} \ldots \underline{R}_k^{a_k}$ of τ, and constants $c_1^{\mathcal{A}} \ldots c_r^{\mathcal{A}}$ from $|\mathcal{A}|$ corresponding to the constant symbols $\underline{c}_1 \ldots \underline{c}_r$ from τ.

For example, if $\tau_0 = \langle \underline{E}^2 \rangle$ consists of a single binary relation symbol then a structure $G = \langle \{0 \ldots n-1\}, E \rangle$ with vocabulary τ_0 is a graph on n vertices. Similarly if $\tau_1 = \langle \underline{M}^1 \rangle$ consists of a single monadic relation symbol then a structure $S = \langle \{0 \ldots n-1\}, M \rangle$ with vocabulary τ_1 is a binary string of length n.

If τ is a vocabulary, let

$$\text{STRUCT}(\tau) = \{G | G \text{ is a structure with vocabulary } \tau\}.$$

We will think of a *problem* as a set of structures of some vocabulary τ. Of course it suffices to only consider problems on binary strings, but it is more interesting to be able to talk about other vocabularies, e.g., graph problems, as well.

Let the relation $s(x, y)$ be the successor relation on the naturals, i.e., $s(x, y)$ holds iff $x + 1 = y$. Throughout this paper we will assume that $\underline{s}$ is a logical relation symbol denoting the successor relation. We will also assume that *all structures under*

[1] Note that the AGAP problem is easily seen to be equivalent to the monotone circuit value problem shown to be complete for P in [Go77].

consideration have at least two elements. Furthermore the constant symbols $\underline{0}$ and $\underline{\max}$ will always refer to the first and last elements of the universe, respectively. [2]

We now define the *first order language* $\mathcal{L}(\tau)$ to be the set of formulas built up from the relation and constant symbols of τ and the logical relation symbols and constant symbols: $=, \underline{s}, \underline{0}, \underline{\max}$, using logical connectives: $\wedge, \vee, \neg$, variables: $x, y, z, \ldots$, and quantifiers: $\forall, \exists$.

If $\varphi \in \mathcal{L}(\tau)$ let $\mathrm{MOD}(\varphi)$ be the set of finite models of φ:

$$\mathrm{MOD}(\varphi) = \{G \in \mathrm{STRUCT}(\tau) | G \models \varphi\}.$$

Let FO be the set of all first order expressible problems:

$$\mathrm{FO} = \{S | (\exists \tau)(\exists \varphi \in \mathcal{L}(\tau)) S = \mathrm{MOD}(\varphi)\}.$$

The following result is well known [Fa74], [AU79], [Im81]; but a new proof of the strictness of the containment follows from Corollary 6.3.

THEOREM 2.1. FO *is strictly contained in* L.

3. First order logic plus transitive closure. In view of Theorem 2.1 we wish to strengthen first order logic so that we may express properties of complexity at least logspace . Let $\varphi(x_1 \ldots x_k, x'_1 \ldots x'_k)$ be any formula. It represents a binary relation on k-tuples. We add to our language the operator TC where $\mathrm{TC}[\lambda \bar{x}, \bar{x}' \varphi]$ denotes the reflexive, transitive closure of the relation $\varphi(\bar{x}, \bar{x}')$.[3] Let (FO + TC) be the set of properties expressible using first order logic plus the operator TC. Let (FO + pos TC) be the set of properties expressible using only positive applications of TC, i.e., not within any negation signs. It is easy to see for example that GAP $\in$ (FO + pos TC):

$$\mathrm{GAP} = \{G \mid G \models \mathrm{TC}[E(x, x')](0, \max)\}.$$

THEOREM 3.1. NL = (FO + pos TC).

Proof. It is easy to see that NL contains (FO + pos TC). If $A(x, x')$ can be checked in NL, then so can $\mathrm{TC}[A(x, x')]$: simply guess an A path. Going the other way we are given a nondeterministic logspace Turing machine, M, accepting a set, S, of structures of a certain vocabulary τ. We must produce a sentence ψ in (FO + TC) such that

$$S = \{G \in \mathrm{STRUCT}(\tau) | G \models \psi\}.$$

The first idea needed to write ψ is that an instantaneous description (ID) of M is of size $O[\log n]$ and can be coded with finitely many variables ranging from 0 to $n - 1$.

For example, suppose that M accepts a graph problem, i.e. the vocabulary, $\tau = \langle \underline{E}^2 \rangle$, consists of a single binary relation symbol. Suppose also that M uses $k \cdot log(n)$ bits of work tape for problems of size n. Input to M consists of n^2 bits – the adjacency matrix for E. An ID for M is a $2k + 3$-tuple: $\langle q, r_1, r_2, w_1, h_1 \ldots w_k, h_k \rangle$. Here q codes M's state and variables r_1, r_2 code the input head position. Note that

[2] As we will see the availability of a successor or ordering relation seems crucial to simulate computation. The last two assumptions are nonstandard and unnecessary. However they are convenient because they make some proofs neater. Otherwise we would have to modify many formulas to deal specially with the trivial case where the universe has only one element. Without the assumption that two distinct constant symbols exist, many irrelevent quantifications would be needed.

[3] Where confusion does not arise we may abbreviate $\mathrm{TC}[\lambda \bar{x}, \bar{x}' \varphi(\bar{x}, \bar{x}')]$ to $\mathrm{TC}[\varphi(\bar{x}, \bar{x}')]$.

the input head is looking at a 1 or 0 according as $E(r_1, r_2)$ holds or does not hold in the input structure. Finally $w_1 ... w_k$ code the $k \cdot log(n)$ bits of M's work tape. One h_i is equal to j where the work head is pointing to the jth bit of w_i; the rest of the h_i's are $n-1$.

The second idea is that using TC we can compute the jth bit of w_i. Let $\mathrm{ON}(w, j)$ mean that $j < \log n$ and bit j of w is on. Starting with the successor relation s we can use TC to express addition and then use TC again to tell if a certain bit in a variable is on:

LEMMA 3.2. *The following predicates are expressible in* (FO + pos TC)*:*

1. $\mathrm{PLUS}(x, y, z) \equiv$ "$x + y = z$".
2. $\mathrm{ON}(w, j)$.

Proof. First, define a binary relation α on pairs such that $\alpha(x, y, x', y')$ holds if and only if $x' = x - 1$ and $y' = y + 1$:

$$\alpha(x, y, x', y') \equiv \big(s(x', x) \wedge s(y, y')\big).$$

Using transitive closure we then get:

$$\mathrm{PLUS}(x, y, z) \equiv \mathrm{TC}[\lambda xy, x'y'\alpha](x, y, 0, z).$$

Now define β as follows:

$$\beta(w, j, w', j') \equiv \big(\exists z(\mathrm{PLUS}(w', w', z) \wedge (w = z \vee s(z, w))) \wedge s(j', j)\big).$$

There is an edge from $\langle w, j\rangle$ to $\langle w', j'\rangle$ if $w' = w/2$ and $j' = j - 1$. Let $\mathrm{ODD}(z)$ abbreviate $\exists x \exists y(\mathrm{PLUS}(x, x, y) \wedge s(y, z))$. It follows that

$$\mathrm{ON}(w, j) \equiv \exists z\big(\mathrm{ODD}(z) \wedge \mathrm{TC}[\lambda wj, w'j'\beta](w, j, z, 0)\big). \qquad \square$$

Once we can tell what the work head is looking at we can write the predicate $\mathrm{NEXT}(\mathrm{ID}_a, \mathrm{ID}_b)$ meaning that ID_b follows from ID_a in one move of M.[4] (Note that the successor relation is used not only to code n bits into a single variable, but also to say that the read head moves one space to the left or right. Any input structure is given to a Turing machine in some order, and it may search the structure in that order.)

Using one more positive application of TC we can express $\mathrm{PATH}(\mathrm{ID}_a, \mathrm{ID}_b)$ meaning that there is a computation of M starting at ID_a and leading to ID_b. Let $\psi = \mathrm{PATH}(\mathrm{ID}_i, \mathrm{ID}_f)$ where ID_i and ID_f are M's initial and final ID's respectively. Then $\psi \in$ (FO+pos TC) and an input structure, G, of the correct vocabulary satisfies ψ if and only if M accepts G. This proves Theorem 3.1. $\square$

The sentence $\psi = \mathrm{PATH}(\mathrm{ID}_i, \mathrm{ID}_f)$ in the above proof has an interesting form. It is written with several positive applications of TC, but we show in the next theorem that these may be merged into one. Thus for each problem C in NL there is a $2k$-ary first order formula φ such that a structure G is a member of C iff G satisfies $\mathrm{TC}[\varphi](\bar{0}, \overline{\mathrm{max}})$. This suggests that GAP is complete for NL via an extremely weak kind of reduction. We call these new reductions *first order translations* and we discuss them right after we show that ψ can indeed be written in this simple form.

THEOREM 3.3. *Let* $\varphi \in$ (FO + pos TC). *Then* φ *is equivalent to a formula of the form*

$$\mathrm{TC}[\psi(u_1, \ldots, u_k, u'_1, \ldots, u'_k)](\bar{0}, \overline{\mathrm{max}})$$

[4] This is a standard construction. See [Fa74], [Im81], [Im82a], [Im82b].

where ψ is first order and $\bar{0}$ (resp. $\overline{\max}$) is the constant symbol 0 *(resp.* max*) repeated k times.*

Proof. By induction on the complexity of φ. There are five cases:

1. The formula φ is atomic or the negation of an atomic formula. In this case let u, u' be variables not occurring in φ. Then $\varphi \leftrightarrow \mathrm{TC}[\varphi(u,u')](0,\max)$.
2. $\varphi \equiv \mathrm{TC}[\psi(\bar{u},\bar{u}')](\bar{q},\bar{r})$. We wish to replace $\bar{q},\bar{r}$ with $\bar{0},\overline{\max}$. Put

$$\begin{aligned}\alpha_2(s,t,\bar{u},s',t',\bar{u}') \equiv\ & \big[s=0\wedge t=0\wedge \bar{u}=\bar{0}\ \wedge\ s'=0\wedge t'=\max\wedge\bar{u}'=\bar{q}\big]\\ \vee&\big[s=0\wedge t=\max\wedge\bar{u}\neq\bar{r}\ \wedge\ s'=0\wedge t'=\max\wedge\psi(\bar{u},\bar{u}')\big]\\ \vee&\big[s=0\wedge t=\max\wedge\bar{u}=\bar{r}\ \wedge\ s'\\ &\qquad=\max\wedge t'=\max\wedge\bar{u}'=\overline{\max}\big].\end{aligned}$$

To make this kind of proof more readable we will write formulas such as the above in tabular form. For example, α_2 is described by Table 1.

TABLE 1

α_2	1	2	3
Before s	0	0	0
t	0	max	max
$\bar{u}$	$\bar{0}$	$\neq \bar{r}$	$\bar{r}$
After s'	0	0	max
t'	max	max	max
$\bar{u}'$	$\bar{q}$	s.t. $\psi(\bar{u},\bar{u}')$	$\overline{\max}$

The variables t, s split the ρ-path in three stages: ($st = 00$): Set $\bar{u}'$ to $\bar{q}$ and go to next stage. ($st = 0\max$): Take a ψ step and stay in this stage. When you reach $\bar{r}$ go to next stage. ($st = \max\max$): Set $\bar{u}' = \overline{\max}$ and stop. Thus as desired:

$$\varphi \leftrightarrow \mathrm{TC}[\alpha_2(s,t,\bar{u},s',t',\bar{u}')](\bar{0},\overline{\max})\ .$$

3. $\varphi \equiv \exists x \mathrm{TC}[\lambda\bar{u},\bar{u}'\psi(\bar{u},\bar{u}',x)](\bar{0},\overline{\max})$. Here the notation means that the transitive closure is taken over the relation $\psi(\bar{u},\bar{u}')$ and the variable x occurs free in ψ. Define a formula $\alpha_3(\bar{u},x,\bar{u}',x')$ via Table 2.

TABLE 2

α_3	1	2	3
Before $\bar{u}$	$\bar{0}$	$\neq \overline{\max}$	$\overline{\max}$
x	0	any	any
After $\bar{u}'$	$\bar{0}$	s.t. $\psi(\bar{u},\bar{u}',x)$	$\overline{\max}$
x'	any	x	max

Notice that α_3 allows a guess of x' on the first step. This x must remain until we reach $\overline{\max}$. Thus

$$\varphi \leftrightarrow \mathrm{TC}[\alpha_3(\bar{u},x,\bar{u}',x')](\bar{0},\overline{\max})\ .$$

4. $\varphi \equiv \forall x \mathrm{TC}[\lambda\bar{u},\bar{u}'\psi(\bar{u},\bar{u}';x)](\bar{0},\overline{\max})$. In this case we simulate the $\forall x$ by searching through all x's in order using successor. I'm not sure whether the result holds if s is not available. (See Table 3.)

TABLE 3

α_4	1	2
Before $\bar{u}$	$\neq \overline{\max}$	$\overline{\max}$
x	any	$\neq \max$
After $\bar{u}'$	s.t. $\psi(\bar{u}, \bar{u}', x)$	$\bar{0}$
x'	x	s.t. $s(x, x')$

Note that the form of an α_4 path is as follows: Start with $\bar{u} = \bar{0}$ and $x = 0$ and proceed along column 1 with x fixed until $\bar{u} = \overline{\max}$. Then in column 2 we increment x and repeat until we have checked that for all x, $\mathrm{TC}[\lambda\bar{u}, \bar{u}'\psi(\bar{u}, \bar{u}'; x)](\bar{0}, \overline{\max})$ holds. Thus $\varphi \leftrightarrow \mathrm{TC}[\alpha_4(\bar{u}, x, \bar{u}', x')](\bar{0}, \overline{\max})$

5. $\varphi \equiv \mathrm{TC}\big[\lambda\bar{u}, \bar{v}\mathrm{TC}[\lambda\bar{x}, \bar{x}'\psi(\bar{x}, \bar{x}', \bar{u}, \bar{v})](\bar{0}, \overline{\max})\big](\bar{0}, \overline{\max})$. Here we combine two nested TC's into one on α_5 defined in Table 4.

TABLE 4

α_5	1	2	3	4
Before $\bar{u}$	$\bar{0}$	any	any	any
$\bar{v}$	$\bar{0}$	$\neq \bar{0}$	$\neq \overline{\max}$	$\overline{\max}$
$\bar{x}$	$\bar{0}$	$\neq \overline{\max}$	$\overline{\max}$	$\overline{\max}$
After $\bar{u}'$	$\bar{0}$	$\bar{u}$	$\bar{v}$	$\overline{\max}$
$\bar{v}'$	any	$\bar{v}$	any	$\overline{\max}$
$\bar{x}'$	$\bar{0}$	s.t. $\psi(\bar{x}, \bar{x}', \bar{u}, \bar{v})$	$\bar{0}$	$\overline{\max}$

We claim that $\varphi \leftrightarrow \mathrm{TC}[\alpha_5(\bar{u}, \bar{v}, \bar{x}, \bar{u}', \bar{v}', \bar{x}')](\bar{0}, \overline{\max})$. This holds because an α_5 path consists exactly of a series of $\psi(\cdot, \cdot, \bar{u}, \bar{v})$ paths from $\bar{0}$ to $\overline{\max}$ with $\bar{u}, \bar{v}$ fixed (column 2). At the end of any such path we know that $\mathrm{TC}[\lambda\bar{x}, \bar{x}'\psi(x, x'; u, v)](\bar{0}, \overline{\max})$ holds. The α_5 path may now appropriately step from $(\bar{u}, \bar{v}, \overline{\max})$ to $(\bar{v}, \bar{v}', \bar{0})$, i.e. reach $\bar{v}$ and begin trying to move from $\bar{v}$ to some new $\bar{v}'$ (column 3). Finally if we ever reach $\bar{x} = \overline{\max}$ when $\bar{v} = \overline{\max}$ then we are done (column 4).

Note that the cases of disjunction and conjunction follow easily from cases 3 and 4, respectively. □

DEFINITION 3.4. Let τ_1 and τ_2 be vocabularies where $\tau_2 = \langle \underline{R}_1^{a_1} \ldots \underline{R}_r^{a_r} \rangle$. Let k be a constant. An *interpretation*, σ, of $\mathcal{L}(\tau_2)$ in $\mathcal{L}(\tau_1)$ is a sequence of r formulas from $\mathcal{L}(\tau_1)$: $\Sigma_i(x_{11} \ldots x_{1k} \ldots x_{a_i k})$ $i = 1 \ldots r$, where each Σ_i has $k \cdot a_i$ free variables.

Such an interpretation σ translates any structure $G \in \mathrm{STRUCT}[\tau_1]$ to a structure $\sigma(G) \in \mathrm{STRUCT}[\tau_2]$ where the universe of $\sigma(G)$ is the set of all k-tuples from the universe of G, i.e., $|\sigma(G)| = |G|^k$. The successor relation on $\sigma(G)$ is defined using the lexicographical ordering on these k-tuples. Thus $0^{\sigma(G)} = \langle 0^G, \ldots, 0^G \rangle$ and $\max^{\sigma(G)} = \langle \max^G, \ldots, \max^G \rangle$. Each nonlogical relation R_i of $\sigma(G)$ is defined using the corresponding Σ_i:

$$R_i^{\sigma(G)} = \{\langle \bar{g}_1, \ldots, \bar{g}_{a_i} \rangle \in |\sigma(G)|^{a_i} \mid G \models \Sigma_i(\bar{g}_1, \ldots, \bar{g}_{a_i})\}.$$

See [En72] for a discussion of interpretations between theories. See also [LG77] for a proof that SAT is NP complete via "elementary reductions" – essentially the same thing as our first order translations.

DEFINITION 3.5. Given two problems: $S \subset \mathrm{STRUCT}[\tau_1]$ and $T \subset \mathrm{STRUCT}[\tau_2]$, a *first order translation* of S to T is an interpretation, σ, of $\mathcal{L}(\tau_2)$ in $\mathcal{L}(\tau_1)$ such that:

$$G \in S \Leftrightarrow \sigma(G) \in T.$$

As our first example we prove:

COROLLARY 3.6. GAP *is complete for* NL *via first order translations.*

Proof. Let $S \subseteq \text{STRUCT}[\tau_1]$ be any problem in NL. By Theorems 3.1 and 3.3 there is a constant k and a formula $\psi(u_1, \ldots, u_k, u'_1, \ldots, u'_k) \in \mathcal{L}(\tau_1)$ such that given $\mathcal{A} \in \text{STRUCT}[\tau_1]$ we have

$$\mathcal{A} \in S \Leftrightarrow \mathcal{A} \models \text{TC}[\psi](\bar{0}, \overline{\max}) \quad .$$

Let $\tau_2 = \langle \underline{E}^2 \rangle$. Let $\sigma = \langle \psi(\bar{u}, \bar{u}') \rangle$ be an interpretation of $\mathcal{L}(\tau_2)$ in $\mathcal{L}(\tau_1)$. Then σ is a first order translation of S to GAP because for any $\mathcal{A} \in \text{STRUCT}[\tau_1]$ we have:

$$\begin{aligned} \mathcal{A} \in S \quad &\Leftrightarrow \quad \mathcal{A} \models \text{TC}[\lambda \bar{u}, \bar{u}' \psi(\bar{u}, \bar{u}')](\bar{0}, \overline{\max}) \\ &\Leftrightarrow \quad \sigma(\mathcal{A}) \models \text{TC}[\lambda x, x' E(x, x')](0, \max) \\ &\Leftrightarrow \quad \sigma(A) \in \text{GAP} \, . \end{aligned}$$

□

The first order translation is an extremely weak kind of reduction and it is clearly appropriate for comparing the power of logical languages. It follows from Theorem 6.2 that first order translations are a uniform version of constant depth reductions, cf. [CSV84]. It would seem that projection reductions [SV85] are even weaker. However an examination of the proof of Theorem 3.3 reveals that not only have we removed all but one occurrence of TC, but we have removed all the quantifiers as well. In fact we have proved the following stronger version of Theorem 3.3.

THEOREM 3.7. *Let* $\varphi \in (\text{FO} + \text{pos TC})$. *Then* φ *is equivalent to a formula of the form* $\text{TC}[\psi](\bar{0}, \overline{\max})$ *where* ψ *is a quantifier free formula in disjunctive normal form:*

$$\psi \equiv \bigvee_{i=1}^{I} \alpha_i \wedge \beta_i$$

where

1. *Each* α_i *is a conjunction of the logical atomic relations,* $s, =$, *and their negations.*
2. *Each* β_i *is atomic or negated atomic.*
3. *For* $i \neq j$, α_i *and* α_j *are mutually exclusive.*

Call a first order translation $\sigma = \langle \Sigma_1, \ldots, \Sigma_r \rangle$ a *projection translation* if each of its formulas is in the form of ψ in the above theorem. Note that if ψ is represented in tabular form then the above conditions are equivalent to the following:

1. No quantifiers and no logical connectives except '$\neg$' occur in the table.
2. There is at most one occurrence of a nonlogical relation symbol in each column.
3. The columns are mutually exclusive.

It follows immediately that

COROLLARY 3.8. GAP *is complete for* NL *via projection translations.*

We hope that those readers familiar with the projection reductions of [SV85] will check for themselves that projection translations are a uniform version of projection reductions.

We close this section with a discussion of the complexity class (FO + TC). We mistakenly claimed in [Im83] that the equality $\Sigma_* L = (\text{FO} + \text{TC})$ could be derived

simply by closing both sides of the equation of Theorem 3.1 under negation. Unfortunately this is wrong, and we now suspect that the two classes are distinct. We can prove the following:

THEOREM 3.9.

$$\bigcup_{k=1}^{\infty} \Sigma_k L \subseteq (\mathrm{FO} + \mathrm{TC}).$$

Proof. Let M be a $\Sigma_k L$ Turing machine and let $\mathcal{A}$ be an input structure to M. Let the predicates $\mathrm{EPATH}_M(\bar{x}, \bar{y})$ (resp. $\mathrm{APATH}_M(\bar{x}, \bar{y})$) mean that $\bar{x}$ and $\bar{y}$ are ID's of M such that there is a computation path of M each of whose ID's is existential (resp. universal) except for $\bar{y}$ which is universal or final (resp. existential or final). It is immediate from Theorem 3.1 that EPATH and APATH are expressible in (FO + pos TC). Let ID_i and ID_f be M's initial and final ID's. Then M accepts $\mathcal{A}$ if and only if $\mathcal{A}$ satisfies the following sentence:

$$(\exists \mathrm{ID}_1 \forall \mathrm{ID}_2 \exists \mathrm{ID}_3 \ldots \mathrm{Q}_k \mathrm{ID}_k) \quad \Big[\mathrm{EPATH}(\mathrm{ID}_i, \mathrm{ID}_1) \wedge \big[\neg \mathrm{APATH}(\mathrm{ID}_1, \mathrm{ID}_2) \vee [\mathrm{EPATH}(\mathrm{ID}_2, \mathrm{ID}_3) \wedge \ldots \wedge \mathrm{ID}_k = \mathrm{ID}_f] \ldots\big]\Big]. \qquad \square$$

Note that the above proof requires no nesting of TC's and $\neg$TC's which is why we suspect that $\Sigma_* L \neq (\mathrm{FO} + \mathrm{TC})$.

4. Other transitive closure operators. We next employ other operators not quite as strong as TC to capture weaker complexity classes. For example, let STC be the symmetric transitive closure operator. Thus if $\varphi(\bar{x}, \bar{x}')$ is a first order relation on k-tuples, then $\mathrm{STC}[\lambda \bar{x}, \bar{x}' \varphi]$ is the symmetric, transitive closure of φ.

$$\mathrm{STC}[\lambda \bar{x}, \bar{x}' \varphi] \equiv \mathrm{TC}[\lambda \bar{x}, \bar{x}' \varphi(\bar{x}, \bar{x}') \vee \varphi(\bar{x}', \bar{x})].$$

The following theorem, whose proof is similar to the proof of Theorem 3.1, shows that STC captures the power of symmetric logspace:

THEOREM 4.1.

1. Sym-L = (FO + pos STC).
2. UGAP *is complete via first order translations for* Sym-L.

We can also add a deterministic version of transitive closure which we call DTC. Given a first order relation $\varphi(\bar{x}, \bar{x}')$ let

$$\varphi_d(\bar{x}, \bar{x}') \equiv \varphi(\bar{x}, \bar{x}') \wedge [(\forall \bar{z}) \neg \varphi(\bar{x}, \bar{z}) \vee (\bar{x}' = \bar{z})].$$

That is, $\varphi_d(\bar{x}, \bar{x}')$ is true just if $\bar{x}'$ is the unique descendent of $\bar{x}$. Now define:

$$\mathrm{DTC}(\varphi) \equiv \mathrm{TC}(\varphi_d).$$

Note that (FO + pos DTC) is closed under negation. Thus:

THEOREM 4.2.

1. $L = (\mathrm{FO} + \mathrm{DTC})$.
2. 1GAP *is complete for* L *via first order translations.*

To prove Theorem 4.1 (resp. Theorem 4.2) we must check that the proofs of Theorem 3.1, Lemma 3.2, and Theorem 3.3 go through when we replace TC by STC

(resp. DTC). Notice that the occurrences of TC in the proof of Lemma 3.2 are of the form $\mathrm{TC}[\gamma](\bar{s},\bar{t})$ where for every tuple $\bar{x}$ there is at most one $\bar{x}'$ such that $\gamma(\bar{x},\bar{x}')$ and furthermore there is no $\bar{x}'$ such that $\gamma(\bar{t},\bar{x}')$. It follows that

$$\mathrm{TC}[\gamma](\bar{s},\bar{t}) \equiv \mathrm{DTC}[\gamma](\bar{s},\bar{t}) \equiv \mathrm{STC}[\gamma](\bar{s},\bar{t}) \quad .$$

Thus Lemma 3.2 still holds with STC or DTC replacing TC as needed. See [LP82] for a more detailed discussion of when nondeterministic (TC) computations or deterministic (DTC) computations may be replaced by symmetric (STC) computations.

To finish the proof of part (1) of Theorems 4.1 and 4.2 we follow the proof of Theorem 3.1. We can express $\mathrm{NEXT}(\mathrm{ID}_a,\mathrm{ID}_b)$ using ON and no additional uses of TC. Finally to express PATH from NEXT one additional use of the STC for Theorem 4.1 or DTC for Theorem 4.2 suffices.

To prove part (2) of Theorems 4.1 and 4.2 we observe that Theorem 3.3 holds when TC is replace by STC or DTC. First consider STC. The reader should check that the theorem and proof remain true if we replace TC everywhere by STC. A typical example is case 4. Here $\varphi \equiv (\forall x)\mathrm{STC}[\lambda\bar{u},\bar{u}'\psi(\bar{u},\bar{u}',x)](\bar{0},\overline{\max})$. Recall that we defined α_4 by Table 5.

TABLE 5

α_4	1	2
Before $\bar{u}$	$\neq \overline{\max}$	$\overline{\max}$
x	any	$\neq \max$
After $\bar{u}'$	s.t. $\psi(\bar{u},\bar{u}',x)$	$\bar{0}$
x'	x	s.t. $s(x,x')$

We claim that:

$$\varphi \leftrightarrow \mathrm{STC}[\alpha_4(\bar{u},x,\bar{u}',x')](\bar{0}0,\overline{\max}\max) \; .$$

To prove this last equivalence suppose that φ holds. Then for all x there is a symmetric $\psi(\cdot,\cdot,x)$ path p_x from $\bar{0}$ to $\overline{\max}$. It follows that

$$(p_0 \times 0),\ \langle(\overline{\max},0),(\bar{0},1)\rangle,\ (p_1 \times 1),\ \ldots,\ \langle(\overline{\max},\max-1),(\bar{0},\max)\rangle,\ (p_{\max} \times \max)$$

is an α_4 path from $\bar{0}0$ to $\overline{\max}\max$. Conversely, any minimal α_4 path from $\bar{0}0$ to $\overline{\max}\max$ must include the edges,

$$\langle(\overline{\max},0),(\bar{0},1)\rangle,\ \langle(\overline{\max},1),(\bar{0},2)\rangle,\ldots,\langle(\overline{\max},\max-1),(\bar{0},\max)\rangle$$

in that order. Furthermore the part of the path between any consecutive pair of the above edges will have constant second component. But this just says that for all x, $\mathrm{STC}[\psi(\bar{u},\bar{u}',x)[\bar{0}0,\overline{\max}\max]]$, i.e., φ holds.

In the DTC case we must modify the construction of the proof of Theorem 3.3 so that a deterministic path is not transformed into a nondeterministic path. The most interesting case is the existential quantifier:

$$\varphi \equiv \exists x\mathrm{DTC}[\lambda\bar{u},\bar{u}'\psi(\bar{u},\bar{u}',x)](\bar{0},\overline{\max}).$$

Here instead of letting the path finder guess the correct x we force the path to try all x's and go to $\overline{\max}$ when a correct one is found. We use the fact that there is a path in an n^k vertex graph if and only if there is such a path of length at most n^k-1.

Let arity($\bar{z}$) = arity($\bar{w}$) = arity($\bar{u}$). In the following z is a counter used to cut off a cycling α-path and w is used to find the α-edge leaving u if one exists. We will abuse notation and write '$s(\bar{z}, \bar{z}')$' to mean that $\bar{z}'$ is the successor of $\bar{z}$ in the lexicographical ordering induced by the successor relation s. Define α_3' by Table 6.

TABLE 6

α_3'	1	2	3	4	5
Before	s.t. $\psi(\bar{u}, \bar{w}, x)$	s.t. $\neg\psi(\bar{u}, \bar{w}, x)$	s.t. $\neg\psi(\bar{u}, \bar{w}, x)$		
$\bar{u}$	$\neq \overline{\max}$	$\neq \overline{\max}$	$\neq \overline{\max}$	$\neq \overline{\max}$	$\overline{\max}$
$\bar{z}$	$\neq \overline{\max}$	$\neq \overline{\max}$	$\neq \overline{\max}$	$\overline{\max}$	any
$\bar{w}$	any	$\neq \overline{\max}$	max	any	any
x	any	any	$\neq$ max	$\neq$ max	any
After $\bar{u}'$	$\bar{w}$	$\bar{u}$	$\bar{0}$	$\bar{0}$	$\overline{\max}$
$\bar{z}'$	s.t. $s(\bar{z}, \bar{z}')$	$\bar{z}$	$\bar{0}$	$\bar{0}$	$\overline{\max}$
$\bar{w}'$	$\bar{0}$	s.t. $s(\bar{w}, \bar{w}')$	$\bar{0}$	$\bar{0}$	$\overline{\max}$
x'	x	x	s.t. $s(x, x')$	s.t. $s(x, x')$	$\overline{\max}$
Comment	take edge to $\bar{w}$; increment $\bar{z}$	no edge from $\bar{u}$ to $\bar{w}$: try next $\bar{w}$	no exit from $\bar{u}$: try next x	looping: try next x	The path is found!

Intuitively, an α_3' path from $\langle \bar{u}, \bar{z}, \bar{w}, x\rangle$ moves to $\langle \bar{w}, \bar{z}+1, \bar{0}, x\rangle$ if $\psi(\bar{u}, \bar{w}, x)$. Otherwise we increment $\bar{w}$ until such an ψ-edge is found. If none is found (column 3), or if we are in a loop (column 4) then we try the next x until we finally reach $\overline{\max}$ or exhaust all x's. It follows that:

$$\varphi \leftrightarrow \mathrm{DTC}[\alpha_3'(\bar{u}, \bar{z}, \bar{w}, x, \bar{u}', \bar{z}', \bar{w}', x')](\bar{0}, \overline{\max}) .$$

This completes the proofs of Theorems 4.1 and 4.2. □

In spite of an over optimistic claim in [Im83] we are not sure whether Σ_*Sym-L = (FO + STC) but we suspect that equality does *not* hold. We can prove the following:

THEOREM 4.3.

$$\Sigma_*\text{Sym-L} \subseteq (\text{FO} + \text{STC}) \subseteq \text{BPL}$$

Proof. The first inclusion follows as in the proof of Theorem 3.9. The second inclusion follows from Theorem 1.3 and standard techniques, (see for example [Re84]). Once we know that the predicate $\alpha(\bar{x}, \bar{x}')$ is in BPL, we can test if $\alpha(\bar{x}, \bar{x}')$ holds with exponentially low probablity of error. Then we can take a random α walk and use Theorem 1.3 to see that STC$[\alpha(\bar{x}, \bar{x}')]$ is also testable in BPL. □

The penultimate operator we add in this section is least fixed point, LFP. Given a first order operator on relations:

$$\varphi(R)[\overline{x}] \equiv Q_1 z_1 ... Q_k z_k M(\overline{x}, \overline{z}, R)$$

we say that φ is *monotone* if $R_1 \subset R_2$ implies $\varphi(R_1) \subset \varphi(R_2)$. For a monotone φ define:

$$\mathrm{LFP}(\lambda \bar{x}\varphi) \equiv \min\{R | \varphi(R) = R\}$$

It is well known that LFP(φ) exists and is computable in polynomial time in the size of the structure involved, see e.g. [Im82b].

EXAMPLE 4.4. The least fixed point operator is a way of formalizing inductive definitions of new relations. Recall the AGAP property discussed in Section 1. Consider the following monotone operator:

$$\Gamma(R)[x,y] \equiv \quad (x=y) \vee \Big[(\exists z)(E(x,z) \wedge R(z,y)) \\ \wedge\big(\neg A(x) \vee (\forall z)(\neg E(x,z) \vee R(z,y))\big)\Big].$$

It is easy to see that

$$\mathrm{LFP}(\Gamma) = \mathrm{APATH}$$

and in fact:

THEOREM 4.5. [Im82a],[Va82]. $P = (\mathrm{FO} + \mathrm{LFP})$.

Instead of using LFP we introduce a variant of it that is more in keeping with DTC, STC, and TC. Suppose that $\varphi(\bar{x},\bar{x}')$ and $\alpha(\bar{x})$ are given formulas where $\mathrm{arity}(x) = \mathrm{arity}(y) = k$. For a given structure $\mathcal{A}$ these formulas define an alternating graph $G_{\varphi,\alpha}$ whose universe consists of k-tuples from $\mathcal{A}$, whose edge relation is the set of pairs of k-tuples for which φ holds in $\mathcal{A}$, and whose universal nodes are those k-tuples satisfying α. We define the *alternating transitive closure operator* (ATC) to be the operator whose value on the pair $\varphi(\bar{x},\bar{x}'), \alpha(\bar{x})$ is $\mathrm{APATH}_{G_{\varphi,\alpha}}$. We can define ATC more precisely in terms of LFP as follows. Referring to Example 4.4, let

$$\Gamma(\varphi,\alpha,R)[\bar{x},\bar{x}'] \equiv \quad (\bar{x}=\bar{x}') \vee \Big[(\exists \bar{z})(\varphi(\bar{x},\bar{z}) \wedge R(\bar{z},\bar{x}')) \\ \wedge\big(\neg\alpha(\bar{x}) \vee (\forall \bar{z})(\neg\varphi(\bar{x},\bar{z}) \vee R(\bar{z},\bar{x}'))\big)\Big].$$

DEFINITION 4.6. $\mathrm{ATC}[\lambda\bar{x},\bar{x}'\varphi,\alpha] \equiv \mathrm{LFP}(\bar{x},\bar{x}'\Gamma(\varphi,\alpha))$.

We conclude this section with

THEOREM 4.7.

1. $P = (\mathrm{FO} + \mathrm{ATC})$.
2. AGAP *is complete for P via first order translations.*

Proof. This follows as in the proofs of Theorems 3.1, 4.1 and 4.2, recalling that $\mathrm{ASPACE}[\log n] = P$, [CKS81]. The construction of $\mathrm{NEXT}(\mathrm{ID}_a, \mathrm{ID}_b)$ in the proof of Theorem 3.1 works without change, noting that ATC is at least as powerful as TC. Writing the formula $\alpha(x)$, meaning that ID x is in a universal state, is trivial. The $\mathrm{ASPACE}[\log n]$ machine M accepts a structure $\mathcal{A}$ iff $\mathcal{A} \models \mathrm{ATC}(\mathrm{NEXT},\alpha)(\mathrm{ID}_i, \mathrm{ID}_f)$.

To prove part 2 we just note that Theorem 3.3 remains true with ATC substituted for TC. □

5. Second order logic. In second order logic we have first order logic plus the ability to quantify over relations on the universe. The following theorem of Fagin was our original motivation for this line of research:

THEOREM 5.1. [Fa74]. NP = *(2nd O existential).*

Fagin's theorem says that a property is recognizable in NP iff it is expressible by a second order existential formula. Note that we no longer need "s" as a logical symbol because in second order logic we can say, "There exists a binary relation which is a total ordering on the universe." Closing both sides of Theorem 5.1 under negation gives us that a problem is in the polynomial time hierarchy iff it is expressible in second order logic.

THEOREM 5.2. [St77]. $\Sigma_* P = (2nd\ O)$.

Fagin's original result used $2k$-ary relations to encode the $O[n^{2k}]$ bits of an entire NTIME$[n^k]$ computation. Thus he showed:

$$\text{NTIME}[n^k] \subset (2nd\ O \text{ existential, arity } 2k) \subseteq \text{NP}.$$

Lynch [Ly82] points out that in the presence of addition as a new logical relation on the universe, the second order existential sentences can guess merely the n^k moves and piece together the whole computation in arity k. Thus, he shows:

THEOREM 5.3. [Ly82]. NTIME$[n^k] \subseteq$ (*2nd O existential with* PLUS, *arity k*).

COROLLARY 5.4.

$$\bigcup_{c=1}^{\infty} \Sigma_c\text{TIME}[n^k] = (2nd\ O\ with\ \text{PLUS}, arity\ k).$$

Proof. Let $S \in \Sigma_c\text{TIME}[n^k]$. It follows that

$$S = \{\mathcal{A} \mid \exists w_1 \forall w_2 \ldots Q_{c-1} w_{c-1} T(\mathcal{A}, \bar{w})\}$$

where $w_1, \ldots, w_{c-1}$ are bounded in length by $|\mathcal{A}|^k$ and $T(\mathcal{A}, \bar{w})$ is an NTIME$(|\mathcal{A}|^k)$ property if c is odd and a CONTIME$(|\mathcal{A}|^k)$ property if c is even. It follows from Theorem 5.3 that T is expressible as a second order existential, arity k formula: $\psi \equiv \exists R_c \varphi(R_c)$ if c is odd, (or if c is even, $\psi \equiv \forall R_c \varphi(R_c)$). Finally we can replace each w_i of length at most n^k by a relation R_i on $\mathcal{A}$ of arity k. Thus

$$S = \{\mathcal{A} \mid \mathcal{A} \models \exists R_1 \forall R_2 \ldots Q_c R_c \psi\}.$$

Conversely, let φ be a second order arity k formula,

$$\varphi \equiv \exists R_1 \forall R_2 \ldots Q_c R_c \psi$$

where ψ is first order.

In particular $\psi \equiv (\exists x_1 \ldots Q_a x_a) M(\bar{R}, \bar{x})$ where M is a constant size quantifier free formula. It follows that we can test in $\Sigma_{c+a}\text{TIME}[n^k]$ whether an input $\mathcal{A}$ satisfies ψ. This is done by guessing the R's and x's in the appropriate existential or universal state. It remains to check the truth of a constant size quantifier free sentence. This can easily be done in time linear in $|\langle \mathcal{A}, R_1, \ldots, R_c \rangle|$. □

Note that in the above results the relation "PLUS" need only be added when k is 1 or 2, otherwise it is definable.

As in the previous section we can add closure operators to second order logic in order to express properties which seem computationally more difficult than the polynomial time hierarchy. If $\varphi(\overline{R}, \overline{S})$ is a sentence expressing a binary super relation on k-tuples of relations $\overline{R}$ and $\overline{S}$, then TC(φ), STC(φ), DTC(φ) express the transitive closure, symmetric transitive closure, deterministic transitive closure, respectively, of φ. It is not hard to show:

THEOREM 5.5. *For* $k = 1, 2, \ldots$

1. DSPACE$[n^k] = (2nd\ O$ arity $k, +\text{DTC})$
2. Sym-SPACE$[n^k] = (2nd\ O$ arity $k, +\text{pos STC})$.
3. NSPACE$[n^k] = (2nd\ O$ arity $k, +\text{pos TC})$.

Proof. A k-ary relation R over an n element universe consists of n^k bits. It is an easy exercise to code an $O[n^k]$ space instantaneous description with a set of k-ary relations, $X_1 \ldots X_c$, and to write the first order sentence $\varphi_M(X_1 \ldots X_c, Y_1 \ldots Y_c)$

meaning that ID $\overline{Y}$ follows from $\overline{X}$ in one move of Turing machine M. One aplication of the appropriate transitive closure operator expresses an entire computation. □

The following follows immediately:

COROLLARY 5.6. PSPACE = (2*nd* O + TC) = (2*nd* O + STC) = (2*nd* O + DTC).

In similar fashion we can add a second order ATC operator. Recalling that ASPACE$[n^k] = \bigcup_{c=1}^{\infty}$ DTIME$[c^{n^k}]$ we discover

THEOREM 5.7. *For* $k = 1, 2, \ldots$

$$\bigcup_{c=1}^{\infty} \text{DTIME}[c^{n^k}] = (2nd\ O \text{ arity } k + \text{ATC}).$$

6. Lower bounds. A lower bound by Furst, Saxe and Sipser has interesting consequences for us. (See [FSS81] and also [Aj83] and [Ya85].) The PARITY problem is to determine whether the n inputs to a boolean circuit include an even number which are one.

THEOREM 6.1. [FSS81]. PARITY *cannot be computed by a sequence of polynomial size, bounded depth circuits.*

Theorem 5.1 interests us greatly because of the following relation between bounded depth polynomial size circuits and first order sentences:

THEOREM 6.2. *Given a problem* S *and an integer* $d > 1$ *the following are equivalent:*

1. S *is recognized by a sequence of depth* $d + 1$*, polynomial size circuits, with bounded fan-in at the bottom level.*

2. *There exists a new logical relation* $R \subset \mathbf{N}^a$ *and a first order formula* φ *in which* R *occurs such that* φ *expresses* S*. The formula* φ *contains* d *alternating blocks of quantifiers.*

Proof. (1 $\Rightarrow$ 2): We may assume that the similarity type of S is $\tau = \langle M^1 \rangle$ consisting of a single monadic predicate. Thus to a first order formula an input of size n is a structure $\mathcal{A} = \langle \{0 \ldots n-1\}, M^{\mathcal{A}} \rangle$. To a boolean circuit the same input is the string of n boolean variables $a_1 \ldots a_n$, where $(a_i = 1) \Leftrightarrow i \in M^{\mathcal{A}}$.

Suppose that (1) holds and that the circuits $C_1, C_2, \ldots$ accepting S have depth $d + 1$ and size at most n^k and that their bottom level has fan-in at most k. For definiteness assume that the C_i's top gate is an 'and' gate and that their bottom gate is an 'or' gate. The other cases are analagous. We may also assume – by repeating portions of the circuit if necessary – that the fan-in at all levels of C_n besides the bottom is n^k. We may label each of the n^k edges leaving a gate by a k digit integer in n-ary, $z_1 z_2 \ldots z_k$. In this way each 'or' gate v at the bottom level has a label $z_{11} z_{12} \ldots z_{1k} z_{21} \ldots z_{dk}$. Each such gate in C_n is a disjunction $s_1 \vee \ldots \vee s_k$ of literals: a_i or $\bar{a}_i$.

We wish to define the logical relation R so that for each n, it codes the circuit C_n by listing the literals, $s_1, \ldots, s_k$, of each bottom gate. We will define R in two steps. First we define a relation Q which does the coding. Then we will refine this to R in such a way that we can write the desired formula using only d alternations.

Define the relation Q as follows: $Q(i, z_{11}, z_{12}, \ldots, z_{dk}, y, j)$ holds iff $j = 0$ (resp. $j = \max$) and literal a_y (resp. $\bar{a}_y$) occurs in the gate with label $z_{11} z_{12} \ldots z_{dk}$ in C_{i+1}.

Define the sentence

$$\psi \equiv \forall z_{11} \ldots z_{1k} \exists z_{21} \ldots z_{2k} \ldots \forall z_{d1} \ldots z_{dk} \exists y \big(Q(\max, \bar{z}, y, 0) \wedge M(y) \\ \vee Q(\max, \bar{z}, y, \max) \wedge \neg M(y) \big).$$

For inputs of size n, the sentence ψ mimics the circuit C_n. Inside the square brackets it says that one of the disjuncts of the node $z_{11} \ldots z_{dk}$ is true. It follows that for any structure $\mathcal{A}$, $\mathcal{A} \models \psi$ if and only if $\mathcal{A} \in S$.

What remains to be shown is that we can remove the quantifier $\exists y$ so that our formula has d alternating blocks of quantifiers as required. We do this by saying that for all $y_1 \ldots y_k$, if the y_i's are distinct and correspond to all the disjuncts of the 'or' gate in question, then at least one of them is true. We define R as a modification of Q such that for each n and each node $\bar{z}$ there are exactly k y's such that $R(\max, \bar{z}, y, 0) \vee R(\max, \bar{z}, y, \max)$ holds. This is done by throwing in c dummy nodes w for gates with only $k - c$ disjuncts. For each such w we make $R(\max, \bar{z}, w, 0)$ and $R(\max, \bar{z}, w, \max)$ true. That is we define R as follows: For all $n, \bar{z}$ if only $y_1, \ldots, y_{k-c}$ satisfy $Q(\max, \bar{z}, y, 0) \vee Q(\max, \bar{z}, y, \max)$, then choose $w_1, \ldots, w_c$ distinct from these y_i's (note that we may assume that $k < n$). Let

$$R(\max, \bar{z}, \cdot, \cdot) = Q(\max, \bar{z}, \cdot, \cdot) \cup \{ \langle \max, \bar{z}, w_i, j \rangle \mid i = 1 \ldots c; j = 0, \max \}$$

and put

$$\alpha \equiv \forall y_1 \ldots \forall y_k \Big(\big[y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge \ldots \wedge y_{k-1} \neq y_k \\ \wedge (R(\max, \bar{z}, y_1, 0) \vee R(\max, \bar{z}, y_1, \max)) \\ \wedge \ldots \wedge (R(\max, \bar{z}, y_k, 0) \vee R(\max, \bar{z}, y_k, \max)) \big] \\ \rightarrow \big[(\neg R(\max, \bar{z}, y_1, 0) \wedge \neg M(y_1)) \\ \vee (\neg R(\max, \bar{z}, y_1, \max) \wedge M(y_1)) \vee \ldots \vee (\neg R(\max, \bar{z}, y_k, \max) \wedge M(y_k)) \big] \Big).$$

Here α says that $y_1 \ldots y_k$ are all the disjuncts of the "or" gate in question and at least one of them comes out true. Finally let

$$\varphi \equiv \forall z_{11} \ldots z_{1k} \exists z_{21} \ldots z_{2k} \ldots \forall z_{d1} \ldots z_{dk} (\alpha).$$

(Then φ is as required.)

$(2 \Rightarrow 1)$: The converse is simpler. Assume (2). Then the sentence φ must be in the following form:

$$\varphi \equiv \forall z_{11} \ldots z_{1k} \exists z_{21} \ldots z_{2k} \ldots Q_d z_{d1} \ldots z_{dk} (\gamma(R, M, \bar{z}))$$

where γ is quantifier free. By replacing each universal (resp. existential) block of quantifiers by an "and" (resp. "or") gate, we can rewrite φ as a circuit of depth d and fan-in n^k in which each bottom gate whose label is the $d \cdot k$ tuple $\bar{c}$ is assigned the value $\gamma(R, M, \bar{c})$. The formula $\gamma(R, M, \bar{c})$ for a fixed assignment of $\bar{c}$ is a boolean combination of formulas of the form $M(c_{ij})$, i.e., a constant size boolean combination of literals independent of n. Each such formula can be written in disjunctive normal form if Q_d is $\exists$ and conjunctive normal form if it's $\forall$. The resulting circuit is polynomial size and depth $d+1$ with the bottom level of bounded fan-in as required. □

Note that the role of the new logical relation R in the above proof is to translate a nonuniform sequence of circuits into a single first order formula. A strong lower

bound on the expressive power of first order logic follows easily from Theorems 6.1 and 6.2.

COROLLARY 6.3. *For any k and any logical relation R on* $\mathbf{N}^k$,

$$\text{PARITY} \notin (\text{FO} + R)$$

Sipser also proved the following hierarchy result for bounded depth circuits:

THEOREM 6.4. [Si83]. *For all* $d > 1$ *there is a problem* S_d *accepted by polynomial size, depth d circuits; but not by polynomial size depth d circuits which have bounded fan-in at the bottom level.*

The problem S_d used by Sipser can be expressed by the first order formula

$$\exists x_1 \forall x_2 \ldots Q_d x_d L(x_1, \ldots, x_d)$$

over structures with a single relation L of arity d. A corollary of the above theorem is a very strong hierarchy result for first order logic:

COROLLARY 6.5. *For all* $d \geq 1$, *for all k, and for any logical relation R on* $\mathbf{N}^k$,

$$(\text{FO}, d+1 \textit{ alternations}) \not\subseteq (\text{FO} + R, d \textit{ alternations}).$$

7. Conclusions. We have shown that the important complexity classes, C, have corresponding languages, $\mathcal{L}$, such that C consists of exactly those properties expressible in $\mathcal{L}$. Thus questions of complexity can all be translated to expressibility issues. Separating complexity classes is equivalent to showing that certain of the above operators are more powerful than others. Efficient algorithms may be obtained simply by describing a problem in one of our languages. We have also demonstrated a basic connection between boolean circuits and first order logic.

Open questions and work to be done include the following:

1. More precise bounds on the arity of formulas needed to express properties of a given complexity are needed.
2. We have introduced first order translations as a new kind of reduction. Many open questions arise. It seems possible to prove that first order translations between certain problems cannot exist. Be careful however: showing that there is no first order translation from SAT to 1GAP would prove that NP $\neq$ L. (A tractable approach which we are presently pursuing is to prove lower bounds on the arity of projection translations between these problems.)
3. More knowledge is needed concerning the increase in expressibility gained by alternating applications of operators and negation. Let Σ_k^{TC} be first order logic in which k alternations of TC and negation are allowed, starting with TC. Thus for example, $\Sigma_1^{\text{TC}} = (\text{FO} + \text{ pos TC}) = \text{NL}$. We conjecture the following:

   (a): $\Sigma_1^{\text{TC}} \subsetneq \Sigma_2^{\text{TC}} \subsetneq \ldots$

   (b): $\Sigma_1^{\text{STC}} \subsetneq \Sigma_2^{\text{STC}} \subsetneq \ldots$.

   From a proof of any of the proper containments in (a) or (b) it would follow that $L \neq P$. Thus we would be satisfied with the more modest hierarchy results without s as a logical symbol:

   (c): $\Sigma_1^{\text{TC}}(\text{w.o. } s) \subsetneq \Sigma_2^{\text{TC}}(\text{w.o. } s) \subsetneq \ldots$.

   (d): $\Sigma_1^{\text{STC}}(\text{w.o. } s) \subsetneq \Sigma_2^{\text{STC}}(\text{w.o. } s) \subsetneq \ldots$.

4. We have shown that

$$\Sigma_*\text{Sym-}L \subseteq (\text{FO} + \text{STC}) \subseteq \text{BPL}.$$

It would be fascinating to know whether or not either of the above containments is proper.

5. Of course everyone would like to see a proof that some second order property is not expressible in first order plus least fixed point. This would imply that $P \neq \text{NP}$. It is very hard to prove strict hierarchy results for the languages described in this paper. However we feel that extensions to such results as Theorem 6.4 and Theorem 6.1 (together with Yao's strengthening of it) are steps in this direction.
6. Finally, we hope that attractive versions of the above languages will be developed for actual use as programming and/or database query languages.

Acknowledgments. My ideas for this paper have been clarified during many helpful discussions with colleagues. Thanks in particular to: Sam Buss, Steve Lucas, John Reif, Adi Shamir, and Mike Sipser. Thanks also to one of the referees for helpful suggestions on notation including the use of tables to describe formulas.

REFERENCES

[AHU74] A.V. AHO, J.E. HOPCROFT AND J.D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Ma., 1974.

[AU79] A.V. AHO AND J.D. ULLMAN, *Universality of data retrieval languages*, Sixth Symp. on Principles of Programming Languages, 1979, pp. 110–117.

[Aj83] M. AJTAI, Σ_1^1 *formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–18.

[AKLLR79] ALELIUNAS, KARP, LIPTON, LOVÁZ AND RACKOFF, *Random walks, universal traversal sequences, and the complexity of maze problems*, Proc. 20th IEEE Symp. on the Foundation of Computers, 1979, pp. 218–233.

[CH80b] A. CHANDRA AND D. HAREL, *Structure and complexity of relational queries*, 21st Symp. on Foundations of Computer Science, 1980, pp. 333–347. Also appeared in J. Comput. System Sci., 25 (1982), pp. 99–128.

[CKS81] A. CHANDRA, L. STOCKMEYER AND D. KOZEN, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[CSV84] A. CHANDRA, L. STOCKMEYER AND U. VISHKIN, *Constant depth reducibility*, this Journal, 13 (1984), pp. 423–439.

[En72] H. ENDERTON, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[Fa74] R. FAGIN, *Generalized first-order spectra and polynomial-time recognizable sets*, in Complexity of Computation, R. Karp, ed., Society for Industrial and Applied Mathematics-American Mathematical Society Proc., 7, 1974, pp. 27–41.

[FSS81] M. FURST, J. B. SAXE AND M. SIPSER, *Parity, circuits, and the polynomial-time hierarchy*, Proc. 22nd IEEE Symp. on the Foundation of Computers, 1981, pp. 260–270.

[Go77] L. GOLDSCHLAGER, *The monotone and planar circuit value problems are log space complete for P*, SIGACT News, 9 (1977).

[Gr84] E. GRANDJEAN, *The spectra of first-order sentences and computational complexity*, this Journal, 13 (1984), pp. 356–373.

[HIM78] J. HARTMANIS, N. IMMERMAN AND S. MAHANEY, *One-way log tape reductions*, Proc. 19th IEEE Symp. on the Foundation of Computers, 1978, pp. 65–72.

[Im81] N. IMMERMAN, *Number of quantifiers is better than number of tape cells*, J. Comput. System Sci., 22 (1981), pp. 65–72.

[Im82a] ———, *Upper and lower bounds for first order expressibility*, J. Comput System Sci., 25 (1982), pp. 76–98.

[Im82b] ———, *Relational queries computable in polynomial time*, Proc. 14th Ann. ACM Symp. on the Theory of Computing, 1982, pp. 147–152.

[Im83] ———, *Languages which capture complexity classes*, Proc. 15th Ann. ACM Symp. on the Theory of Computing, 1983, pp. 347–354.

[LP82] H. LEWIS AND C. H. PAPADIMITRIOU, *Symmetric space bounded computation*, Theoret. Comp. Sci., 19 (1982), pp. 161–188.

[LG77] L. LOVÁSZ AND P. GÁCS, *Some remarks on generalized spectra*, Zeitchr. f. math, Logik und Grundlagen d. Math., 23 (1977), pp. 547–554.

[Ly82] J. LYNCH, *Complexity classes and theories of finite models*, Math. Systems Theory, 15 (1982), pp. 127–144.

[Re84] J. REIF, *Symmetric complementation*, J. Assoc. Comput. Mach., 31 (1984), pp. 404–421.

[Sa73] W. SAVITCH, *Maze recognizing automata and nondeterministic tape complexity*, J. Comput. System Sci., 7 (1973), pp. 389–403.

[Si83] M. SIPSER, *Borel sets and circuit complexity*, 15th Symp. on Theory of Computation, 1983, pp. 61–69.

[St77] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.

[SV85] S. SKYUM AND L. G. VALIANT, *A complexity theory based on boolean algeba*, J. Assoc. Comput. Mach., 32 (1985), pp. 484–502.

[SV84] L. STOCKMEYER AND U. VISHKIN, *Simulation of parallel random access machines by circuits*, this Journal, 13 (1984), pp. 409–422.

[Tu82] G. TURÁN, *On the definability of properties of finite graphs*, to appear.

[Va82] L. G. VALIANT, *Reducibility by algebraic projections*, L'Enseignement mathématique, T. XXVIII (1982), pp. 253–268.

[Va82] M. VARDI, *Complexity of relational query languages*, 14th Symp. on Theory of Computation, 1982, pp. 137–146.

[Ya85] A. CHI-CHIH YAO, *Separating the polynomial-time hierarchy by oracles*, 26th IEEE Symp. on Foundations of Comput. Sci., 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987

© 1987 Society for Industrial and Applied Mathematics

LOGSPACE HIERARCHIES, POLYNOMIAL TIME AND THE COMPLEXITY OF FAIRNESS PROBLEMS CONCERNING ω-MACHINES*

LOUIS E. ROSIER† AND HSU-CHUN YEN†‡

Abstract. In this paper, we define a *restricted logspace oracle hierarchy* which turns out to be equivalent to the *logspace alternation hierarchy* (of Chandra, Kozen and Stockmeyer) and thus is contained within the second level of the *logspace oracle heirarchy* (of Ruzzo, Simon and Tompa). We then examine problems concerning various types of "fair" computations with respect to *ω-Finite State Machines* (ω-FSM's) and *ω-One Counter Machines* (ω-1CM's). For example, we consider the nonemptiness problem for ω-FSM's and ω-1CM's where acceptance is defined in the usual fashion, but with a fairness constraint imposed on accepting computations. Our results yield problems that are complete not only for NLOGSPACE and PTIME but the second and third levels of the restricted logspace oracle hierarchy as well. As far as we know, these are the first natural problems shown to be complete for various levels of the logspace alternation hierarchy. The problems are also of independent interest. In fact, the nonemptiness problem (with fairness constraints) for ω-machines has been shown to have immediate applications to the verification of concurrent finite state programs. Furthermore, the results can be used to strengthen known results concerning some related fairness problems that involve temporal logic (e.g. model checking).

Key words. concurrent finite state programs, complete problems, complexity of fairness problems, finite automata, computational complexity, logspace alternation hierarchy, logspace oracle hierarchy, model checking, nonemptiness problem, omega automata, polynomial time, temporal logic

AMS(MOS) subject classifications. 68C25, 68D05

1. Introduction.

One of the more fundamental tools used in categorizing the complexity of decision problems is the notion of completeness. Most notable, of course, is the class of problems which are NP-complete. See e.g. Cook [5], Garey and Johnson [9] and Karp [16]. Other complexity classes for which abundant numbers of complete problems are known include NLOGSPACE, PTIME, and PSPACE. See e.g. Garey and Johnson [9], Hopcroft and Ullman [12], Jones and Laaser [14] and Jones, Lien and Laaser [15]. More recently some natural problems have been shown to be complete for certain complexity classes which are "between" NP and PSPACE. Recent advances here include those of Bentley, Ottmann and Widmayer [1], Papadimitriou [23] and Papadimitriou and Yannakakis [24], which tend to consider problems which are complete for various levels of the Polynomial-Time Hierarchy (Stockmeyer [30]). One could also consider the analogous complexity classes for space hierarchies which fall "between" NLOGSPACE and PTIME. Appropriate consideration here then should focus on the *logspace oracle hierarchy*, and the *logspace alternation hierarchy* (that is contained in the second level of the logspace oracle hierarchy), that were introduced in Chandra, Kozen and Stockmeyer [2], and Ruzzo, Simon and Tompa [28],[1] respectively. Both of these hierarchies are contained in PTIME $\cap$ DSPACE($\log^2 n$). Other complexity classes "below" PTIME for which complete problems are known can be found in Cook [6], Lewis and Papadimitriou [20] and Reif [25].

* Received by the editors June 24, 1985; accepted for publication (in revised form) October 10, 1986. A summary of these results was presented at the 3rd Annual Symposium on Theoretical Aspects of Computer Science in Orsay, France.

† Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712.

‡ Department of Computer Science, Iowa State University, Ames, Iowa 50011.

[1] These were referred to, in [28], as the log n space "oracle hierarchy" and log n space "alternation hierarchy," respectively.

In this paper, we define a *restricted logspace oracle hierarchy* (that turns out to be equivalent to the logspace alternation hierarchy of [2]) that is also contained within the second level of the logspace oracle hierarchy. Our restrictions have to do with fixing the number of oracle calls allowed during a computation. We then examine problems concerning various types of "fair" computations with respect to *ω-Finite State Machines* (ω-FSM's) and *ω-One Counter Machines* (ω-1CM's). In particular, we consider the nonemptiness problem for ω-FSM's and ω-1CM's where acceptance is defined as in Cohen and Gold [4], and Landweber [18] (other acceptance criteria are also considered), but with a fairness constraint imposed on accepting computations. (For a discussion about various types of fairness see e.g. Lehmann, Pnueli and Stavi [19].) We consider the problem for seven different types of acceptance criteria and two different types of fairness. (A summary of the derived complexities can be found in Tables 4.1 and and 4.2.) Our results yield problems that are complete for the following classes:

- NLOGSPACE;
- the second level of the restricted logspace oracle hierarchy;
- the third level of the restricted logspace oracle hierarchy;
- PTIME.

These results, we feel, are interesting for the following reasons:

(i) As far as we know, these are the first natural problems which have been shown to be complete for various levels of the logspace alternation hierarchy of [2].

(ii) The problems are also of independent interest. In fact, the nonemptiness problem (with fairness constraints) for ω-machines has been shown to have immediate applications to the verification of concurrent finite state programs. (See e.g. Vardi [32].)

(iii) In Emerson and Lei [7] the time complexity of the nonemptiness problem for various classes of ω-FSM's was shown to be bounded by a low order polynomial. Their result was derived via a reduction to what they called the canonical form model checking problem. (See [7]. See also Lichtenstein and Pnueli [21].) The results of this paper can be used to show that the canonical form model checking problem of [7] is PTIME-complete.

Related complexity results concerning questions about fair computations can be found in Emerson and Lei [7], Fischer and Paterson [8] and Lichtenstein and Pnueli [21], while completeness results concerning complexity classes "between" DLOGSPACE and NLOGSPACE can be found in Lewis and Papadimitriou [20]. See also Reif [25].

In what follows, we assume the reader is familiar with the basic tenets of automata and complexity theory. Relevant sources would include [9], [12]. The basic computational model used in this paper is the *nondeterministic offline multitape Turing machine* (NTM). Since this paper concerns itself with many different problems the relevant definitions are introduced in the text as they are needed. For brevity, we use the terms DL and NL to denote the complexity classes DLOGSPACE and NLOGSPACE, respectively. All completeness classes mentioned in this paper are with respect to deterministic logspace (many-to-one) reductions.

The remainder of the paper is organized as follows. In § 2, we define the restricted logspace oracle hierarchy and in § 3, we illustrate some useful graph problems that are complete for certain levels within this hierarchy. Section 4 concerns itself with the complexity of various problems with respect to ω-FSM's and ω-1CM's. The last section seeks to apply our results to the canonical form model checking problem of [7].

2. The restricted logspace oracle hierarchy. In [28], the logspace oracle hierarchy which falls between NSPACE($\log n$) and PTIME $\cap$ DSPACE($\log^2 n$) was defined. Our

intention, in this section, is to see what happens to this hierarchy when constraints are imposed on the oracle Turing machines (of [28]) restricting the number of allowable oracle calls made during the course of a computation. In particular, we focus on the case where at most one oracle call is allowed during the course of a computation. What happens is that we obtain a new hierarchy—which we call the restricted logspace oracle hierarchy. We then show that the restricted logspace oracle hierarchy coincides with the logspace alternation hierarchy [2]. This characterization is potentially useful because when considering problems related to this hierarchy it allows us to provide the analysis using either computational model. Furthermore, this hierarchy has the following interesting property:

> As pointed out in [28], the restricted logspace oracle hierarchy (or the logspace alternation hierarchy), unlike its polynomial time analogue, is contained within the second level of the logspace oracle hierarchy.

In subsequent sections, we show that some natural problems concerning the notion of fair computations are complete for certain levels in this hierarchy. This study, we feel, is of interest because it allows us to have more insight into the question of whether the inclusion in "NSPACE($\log n$) $\subseteq$ PTIME" is proper. Results concerning relativized logspace have received broad attention in recent years. See, e.g., Ladner and Lynch [17], Lynch [22] and Ruzzo, Simon and Tompa [28]. Unlike the case with the polynomial-time hierarchy [1], [23], however, little has been done in actually "pigeonholing" natural problems into various classes within the logspace hierarchies. The main contribution of this paper will be exactly that.

Informally, an *oracle Turing machine* (OTM, for short) M is a nondeterministic offline multitape Turing machine with an additional write-only tape called a *query tape* and three distinguished states called *query*, *yes* and *no states*. (An OTM can be deterministic. However, all OTM's mentioned in this paper are nondeterministic, except when otherwise specified.) In addition, a set A, called the *oracle set* is always related to the computation of the OTM M. For convenience, we use $L(M^A)$ to denote the language accepted by the OTM M using A as its oracle set (denoted by M^A). The computation of an OTM is similar to that of an ordinary NTM except when visiting the query states. When in a query state, if the string on the query tape is in the oracle set, then the machine enters the yes state; otherwise, it enters the no state. Moreover, the contents of the query tape will be erased immediately upon entering the yes or no state. The machine can write a symbol on the query tape in every state except the query state. In some sense, it is easier to think of the computation of M^A as follows. Let A' be a machine that accepts the set A. In this discussion, if A is a recursive set, then machine A' can be used in place of the oracle A. When M reaches a query state, it will "consult" the machine A' (which will be referred as the "query machine"), and A' returns yes if the string on the query tape is accepted by A', and returns no otherwise. Then M enters the yes or no state depending on the value returned by A'. Note, however, that the complexity of the computation is only concerned with those resources used by M and not the oracle. Using this concept, one may easily generalize to the class of languages X^Y, where X and Y are complexity classes. Formally, we define X^Y to be the complexity class such that for each language L in X^Y, L can be recognized by an OTM M with oracle set A where M operates within the complexity constraints of X and A is in Y.

In order to study the space hierarchy below PTIME, it is natural to define classes like DL^{NL}, NL^{NL}, $\cdots$, etc. Unfortunately, this kind of definition does not seem to be

of great interest because when the OTM's are defined as above one obtains $NL^{NL} = NL^{DL} = NP$. That is, the logspace hierarchy jumps into the polynomial-time hierarchy on the second level. As pointed out in [17], [28], this phenomenon occurs because nondeterminism when combined with the ability to write long strings on the query tape, interacts in such a way that the machine's power is boosted dramatically. To limit the undesired boost in the power of the OTM's, in [28], they require that only strings of log length are written on the query tape and that the oracle machine behaves deterministically whenever it starts to write on the query tape until it enters the query state. However, the query machine was also, in a sense, allowed read access to the input of the oracle machine. More precisely, the string that appears on the query tape at any instance will be of the form $x \# y$, where x (which cannot be altered by the OTM) is a copy of the OTM's input tape, and $|y|$ is $O(\log |x|)$. One may consider the string $x\#$ to always be on the query tape (i.e. it will not be erased), and only the y part to be alterable during the course of a computation. **In this paper, we consider only OTM's of this sort.** Using such OTM's, a hierarchy, similar to the polynomial time hierarchy, was defined in [28] as follows:

$$\Sigma_0^L = DL,$$
$$\Sigma_{k+1}^L = NL^{\Sigma_k^L},$$
$$\Delta_{k+1}^L = DL^{\Sigma_k^L},$$
$$\Pi_k^L = \text{co-}\Sigma_k^L.$$

The following theorem was also shown in [28].

THEOREM 2.1. $\bigcup_k \Sigma_k^L \subseteq \text{PTIME} \cap \text{DSPACE}(\log^2 n)$.

An interesting question now arises when one also restricts the number of oracle calls allowed during the course of a computation (i.e., along any computation path). Intuitively, the power of an OTM seems to be enhanced as it is allowed to make more oracle calls. For each $d \geqq 1$, we define the following hierarchy:

$$\Sigma_0^{L[d]} = DL,$$

$\Sigma_{k+1}^{L[d]} = \{\mathscr{L} \mid \exists$ an $O(\log n)$ space bounded OTM M such that for every input M consults its oracle at most d times on each computation path, $\exists A \in \Sigma_k^{L[d]}$, such that $\mathscr{L} = L(M^A)\}$,

$\Delta_{k+1}^{L[d]} = \{\mathscr{L} \mid \exists$ a deterministic $O(\log n)$ space bounded OTM M such that for every input M consults its oracle at most d times on each computation path, $\exists A \in \Sigma_k^{L[d]}$, such that $\mathscr{L} = L(M^A)\}$,

$$\Pi_k^{L[d]} = \text{co-}\Sigma_k^{L[d]}.$$

The next theorem shows an interesting result concerning the class $\Sigma_2^{L[d]}$, where $d \geqq 1$ is a fixed constant.

THEOREM 2.2. *Let $d > 1$ be an integer constant and let $A \in NL$. Furthermore, let M be an OTM that consults its oracle at most d times on each computation path. Then there exists an $A' \in NL$ and an OTM M_1 that during the course of any computation consults its oracle at most once such that*:

(1) $L(M^A) = L(M_1^{A'})$, *and*

(2) *for M_1, the last step of any computation is an oracle call and only when the answer to this oracle call is "no" will the machine accept the input.*

Proof. Let $d > 1$ be an arbitrary fixed constant. Let M be an OTM that makes at most d oracle calls on each computation path. Let T be an NTM which accepts A

and runs in $O(\log n)$ space. In what follows, we define A' and show how to construct an OTM M_1 to simulate the computation of M. M_1 will have two more work tapes than M which we shall refer to as T_2 and T_3. The basic idea of the simulation is that, M_1 will simulate the computation of M by writing symbols (those that are supposed to be written on M's query tape) on T_2 until the query state is visited. (Recall that on input x, queries are of the form $x \# y$ with $|y|$ being $O(\log |x|)$.) At this time M_1 "guesses" the oracle's response. If M_1 guesses "yes" then it uses T to nondeterministically verify the guess. If T is able to do so then the simulation continues; otherwise the input is rejected. If, on the other hand, M_1 guesses "no" then M_1 saves the "y part of the no guess" on T_3 (the guess to be verified later in the computation). More precisely, in each query state (of M) M_1 will nondeterministically choose one of the following to execute:

(1) Simulate a computation of T using input T_2 (along with a copy of M's original input). If T accepts, then M_1 enters the yes state; otherwise the input is immediately rejected.

(2) Enter the no state and at the same time, append the y part of T_2 to the end of tape T_3. (A separator "\$" is also added.)

When M_1 enters the yes or no state the string y on T_2 will be immediately erased. The simulation is then continued. Finally when an accepting state is reached, M_1, in order to determine whether the previous "no guesses" were correct, deterministically copies the "saved parts" (on tape T_3) to the query tape and enters the query state. If the oracle's response is "no," M_1 will then stop and accept the input; otherwise, M_1 rejects the input. M_1's oracle is $A' = \{x \# y_1\$ \cdots \$y_m\$ \mid x \in \Sigma^*,\ y_i \in \Sigma^* \text{ for } 1 \leqq i \leqq m,\ 1 \leqq m \leqq d, \text{ and } \exists i,\ 1 \leqq i \leqq m, \text{ such that } x \# y_i \in A\}$. Since A is in NL, A' is also in NL. Now the simulation clearly works, since the function of the oracle call is to verify whether the "no" guesses made during the computation were correct. □

From this theorem we can immediately obtain the following:

COROLLARY 2.1. $\Sigma_2^{L[d]} \equiv \Sigma_2^{L[1]}$, *for any fixed integer constant* $d \geqq 1$.

At this time, we do not know if the above corollary can be extended to higher levels of the hierarchy.

In [2], the logspace alternation hierarchy was introduced based on the model of *alternating Turing machines* (ATM's). Basically the concept of alternation is a generalization of nondeterminism in a way that allows existential and universal quantifiers to alternate during the course of a computation. Four kinds of states exist in an ATM; namely existential, universal, accepting and rejecting states. A universal state can lead to acceptance iff all successors lead to acceptance. On the other hand, an existential state leads to acceptance iff there exists a successor that leads to acceptance. Details can be found in [2]. The complexity classes of languages accepted by time (space) bounded ATM's were also defined in [2]. In particular, $A\Sigma_k^L$ is the set of languages accepted by $O(\log n)$ space bounded ATM's in which the starting state is an existential state and the machine is constrained to make at most $k-1$ alternations during the course of a computation. (Any computation path of such a machine will have at most k alternation blocks.) In [2] such machines are called "k alternation bounded." We adopt this notation here. In what follows, we show that the $A\Sigma_k^L$ hierarchy coincides with the restricted logspace hierarchy $\Sigma_k^{L[1]}$ defined earlier. This characterization proves useful in later sections as both computational models are used in deriving completeness results for various levels of this hierarchy.

LEMMA 2.1. *For all* $k \geqq 1$, $A\Sigma_k^K \subseteq \Sigma_k^{L[1]}$.

Proof (By induction on k).

Induction base: When $k = 1$, we easily see that $A\Sigma_1^L \equiv NL \equiv \Sigma_1^{L[1]}$.

Induction hypothesis: Assume that the assertion is true for $k = n$.

Induction step: Now consider the case when $k = n+1$. Let L be an arbitrary language in $A\Sigma_{n+1}^{L}$. Let M be an $n+1$ alternation bounded ATM that accepts L. Define A' to be the set of strings of the form $x\#p\$q\$w$ such that x is an input string for M, p is an input head position ($1 \leqq p \leqq |x|$), q is a universal state of M, w is a worktape string ($|w|$ is $O(\log |x|)$), and, viewing $p\$q\$w$ as an initial configuration, M accepts from this configuration after at most n alternation blocks. Hence $A' \in A\Pi_n^L$. This plus the induction hypothesis yields $A = \bar{A}'$ (i.e. the complement of A') $\in \Sigma_n^{L[1]}$.

In what follows, we want to construct an oracle machine M' to accept L. Basically, M' is going to simulate the computation of M on input x until the first universal configuration is reached. At this point, M' writes M's current input head position, state and worktape contents on the query tape and then consults the oracle set A. If the answer of the oracle call is "no," M' enters the accepting state; otherwise M' rejects the input. Clearly M' accepts the language L which in turn must therefore be in $\Sigma_{n+1}^{L[1]}$. The lemma then follows. □

LEMMA 2.2. *For all* $k \geqq 1$, $\Sigma_k^{L[1]} \subseteq A\Sigma_1^L$.

Proof (By induction on k).

Induction base: When $k = 1$, we easily see that $\Sigma_1^{L[1]} \equiv NL \equiv A\Sigma_1^L$.

Induction hypothesis: Assume that the assertion is true for $k = n$.

Induction step: Now consider the case when $k = n+1$. Let L be an arbitrary language in $\Sigma_{n+1}^{L[1]}$. Let $A \in \Sigma_n^{L[1]}$. Let M be an OTM that operates in $O(\log n)$ space such that $L(M^A) = L$. Now, ideas similar to those used in the proof of Theorem 2.2 can be used to modify M so that the last step of any computation is the oracle call. According to the induction hypothesis, the language (or set) A is in $A\Sigma_n^L$, and therefore can be accepted by an n alternation bounded ATM W. Let W' be the machine W with universal (accepting) and existential (rejecting) states interchanged. In this way, W', with a universal initial state, will accept the complement of A. Now, we want to construct an $n+1$ alternation bounded ATM M' to accept L. Simply speaking, M' will simulate the computation of M faithfully except that, the string s that is to be written on the query tape by M, will be written on its working tape instead. Therefore, the working tape of M' will look like $\$s\# \cdots$. Now, when a query state is reached, M' will simulate the oracle call by using the mechanism described in Fig. 2.1. More precisely, if both a "yes" or "no" response causes M to accept then M' simply accepts the input. If only a "yes" response causes M to accept then control is passed to W which simply treats the $\$s\#$ as part of its input and simulates the moves accordingly. If only a "no" response causes M to accept then control is passed to W' instead. Notice that the language accepted by M' is L, which is therefore in $A\Sigma_{n+1}^L$. Hence the lemma follows. □

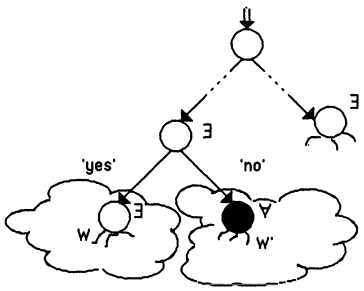


FIG. 2.1. *The machine* M'.

Consequently, we have the following theorem.

THEOREM 2.3. *For all* $k \geqq 1$, $\Sigma_k^{L[1]} \equiv A\Sigma_k^L$.

In [28], it has been shown that $\bigcup_k A\Sigma_k^L \subseteq DL^{NL}$. Using Theorem 2.3, we then have that the restricted logspace oracle hierarchy is contained in DL^{NL}. Let $D^L = \{L_1 \cap L_2 \mid L_1 \in NL, L_2 \in \text{Co-}NL\}$. This class is the space analogue of the class D^P defined in [24], where, in the previous definition, the terms *NL* and Co-*NL* are replaced by the terms *NP* and Co-*NP*, respectively. Since Σ^* is in Co-*NL* (*NL*), we also have that *NL* (Co-*NL*) $\subseteq D^L$, since for any language $L \in NL$ (Co-*NL*), $L \cap \Sigma^* = L$. Furthermore, it should be fairly easy to see that $D^L \subseteq \Sigma_2^{L[1]}$. As a result, we now have the following inclusions:

THEOREM 2.4. $DL \subseteq NL$, $\text{Co-}NL \subseteq D^L \subseteq \Sigma_2^{L[1]} \subseteq \cdots \subseteq \Sigma_k^{L(1)} \subseteq \cdots \subseteq DL^{NL} \subseteq \Sigma_2^L \cdots \subseteq \text{PTIME} \cap \text{DSPACE}(\log^2 n)$. (A more detailed set of inclusions is shown in Fig. 2.2.)

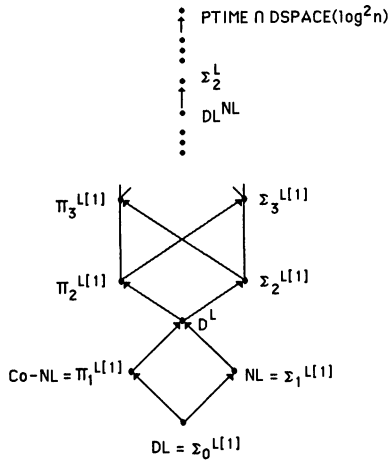


FIG. 2.2. *Inclusions among some sublinear space complexity classes. (Each directed line "→" should be read as "⊆".)*

3. Some complete problems. In this section, we introduce some easy graph problems that are complete for $\Sigma_2^{L[1]}$. These problems are useful, in the sense that they will later be used in § 4 as a vehicle to analyze the complexity of problems concerning ω-machines. In what follows, we define the notion of liveness for a graph, and then we show that the problem of deciding whether a node in a graph is "not" live is complete for $\Sigma_2^{L[1]}$. First, however, the following definitions are required.

An *I-graph G* (or simply a graph if the initial node is clear and understood) consists of a directed graph $G = (V, E)$, where V and E represent the sets of nodes and edges respectively, and there is a unique distinguished node $v_0 \in V$ called the initial node. For two nodes p and q in V, q is said to be *reachable* from p (denoted by $p \xrightarrow{*} q$) iff there exists a sequence of nodes $s_0, s_1, \cdots, s_k$ such that $s_0 = p$, $s_k = q$ and for each i, $0 \leqq i \leqq k-1$, $s_i, s_{i+1} \in V$ and the edge (s_i, s_{i+1}) is in E. We use $p \xrightarrow{+} q$ to denote the case when $k \neq 0$. Similarly, an *infinite path* is an infinite sequence of nodes $s_0, s_1, \cdots, s_n, \cdots$ such that $s_0 = v_0$ (the initial node) and for every i, $s_i \in V$ and the edge (s_i, s_{i+1}) is in E. An infinite path P is said to be *strongly fair* iff for every node v that occurs infinitely often in P, it is the case that all outgoing edges of v have been

traversed infinitely often in P. A node u is said to be *weakly live* iff on every strongly fair path, u occurs infinitely often. A subgraph $C=(V', E')$ of G is called an *Isolated Reachable Component* (IRC, for short), if C satisfies the following conditions:

(1) $\exists v \in V', v_0 \xrightarrow{*} v$ in G.

(2) $\forall v, w \in V', v \xrightarrow{+} w$ in C.

(3) $\forall v \in V'$ and $w \in V, v \xrightarrow{*} w$ in G implies $w \in V'$.

The first condition indicates that the IRC C is reachable from v_0. The second condition guarantees that C is strongly connected. The third condition indicates that no node in $V-V'$ can be reached from any node in V'. (The reader can therefore think of C as sort of a black hole.) The following lemma provides a way to test whether a node in a graph is weakly live. The proof is quite easy and hence will be left to the reader.

LEMMA 3.1. *Given an I-graph G and a node u, u is not weakly live iff there exists an IRC C that does not contain u.*

THEOREM 3.1. *The language $L_1=\{(G, u) \mid G$ is an I-graph, u is a node in G, and u is not weakly live$\}$ is $\Sigma_2^{L[1]}$-complete.*

Proof. Let u_0 be the initial node of the I-graph G. First, we show that L_1 is in $\Sigma_2^{L[1]}$. To do this, we construct a new graph G' from G by adding an edge from every deadend node (i.e. nodes without outgoing edges) to the node u. First note that if u_0 is in such an IRC in G, then there are no reachable deadend nodes in G. In this case the IRC's in G and G' are the same. It should be reasonably easy to see that every IRC not containing u in G is also an IRC not containing u in G'. To see this note that G' may contain an IRC H that was not present in G. (In fact, more than one such H may exist.) Two cases are possible. Either $u \in H$ or $u \notin H$. If $H \not\subseteq G$ then $u \in H$. Now H is not reachable in G. Hence, if $u \notin H$ it must be that u is not reachable in G and that H is reachable from u in G. In either case, by Lemma 3.1 we have that u is not weakly live in G iff there is an IRC C in G' that is also reachable in G such that u is not in C. Note that the transformation from G to G' requires only deterministic logspace.

Since G' has no deadend nodes, u is not weakly live in G iff the following things are true:

(1) $\exists v \in G, u_0 \xrightarrow{*} v$ in G, and

(2) $\neg(v \xrightarrow{*} u$ in $G')$.

Let $G=(V, E)$. Without loss of generality, nodes in V can be numbered from 1 to $|V|$. Hence for each node v in V, a pointer p_v of length $\lceil \log |V| \rceil$ bits is sufficient to represent v. Let $A=\{(G, u) \# p \mid p$ is a pointer to a vertex in G, say v, and $v \xrightarrow{*} u$ in $G'\}$, where G' is the graph obtained from G as indicated above. Clearly A is in NL. Then we can construct an OTM M using A as its oracle set such that $L(M^A)=L_1$. Basically, M nondeterministically chooses a node v in G and writes p_v on its work tape. M then nondeterministically verifies that $u_0 \xrightarrow{*} v$ in G. If M is unable to do so the input is rejected; otherwise, M deterministically copies p_v from the worktape to the query tape and then consults its oracle about $(G, u) \# p_v$. M accepts the input only if the response is "no." Thus, an input (G, u) will be accepted iff 1 and 2 above are true. Hence, L_1 is in $\Sigma_2^{L[1]}$.

In what follows, we show that L_1 is $\Sigma_2^{L[1]}$-hard. Let M be an $O(\log n)$ space bounded OTM that makes at most one oracle call during the course of any computation. Let A be in NL. We also assume that M on input x behaves in the canonical fashion described in Theorem 2.2, i.e., the last step of any computation is the oracle call and M deterministically copies a string y from the work tape to the query tape and then consults its oracle about $x \# y$. M accepts its input only when it receives a negative oracle response. We now show how to construct an I-graph G containing a node u

(depending on M, A, and x), in deterministic logspace, such that M accepts x iff u is not weakly live in G.

Let T be an NTM that accepts A and operates in $O(\log n)$ space. A configuration of M (on input x) is a 4-tuple $c=(q, i, z, w)$, where q is the current state of M, i is the input head position, z represents the contents of the work tapes and w represents the contents of the query tape written thus far. (z implicitly includes the tape head positions as well.) A configuration of T (on some input $x \# y$) is a 3-tuple (q, i, z), where each term is the same as above. We construct a graph $G_{M,x}$ such that each node represents a configuration of M on x and each edge denotes the corresponding transition. Let c_0, the initial configuration of M on x, be the initial node of $G_{M,x}$. Similarly, we define $G_{T,x\#y}$ to be the computation graph of T on $x \# y$. Note that since $M(T)$ operates in $O(\log n)$ space, the size of $G_{M,x}(G_{T,x\#y})$ is polynomially related to the size x. Now, the graph G can be built as follows. G will contain $G_{M,x}$ (as a subgraph) and a new node q'. Furthermore, for each query configuration $c=(q, i, z, w)$, we add an edge from c to the initial node of $G_{T,x\#w}$. In addition, from every accepting node in $G_{T,x\#w}$ we add an edge to the node q', and for every nonaccepting node we add an edge to the initial node of $G_{T,x\#w}$. Lastly, we add an edge from every node of $G_{M,x}$ to c_0—the initial node of $G_{M,x}$, from c_0 to q' and from q' to c_0. See Fig. 3.1. Note that since the string w is always of length logarithmic in x (the input of M), the size of each $G_{T,x\#w}$ will be polynomially related to the size of x. Therefore, the size of the whole graph G is polynomial in the size of x. This implies that the above construction can be done in deterministic logspace.

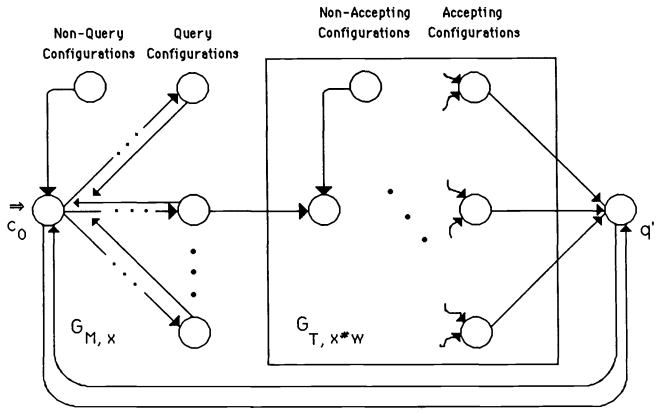


FIG. 3.1. *The i-graph G.*

Now, we claim that M accepts x iff the node q' is not weakly live. To see this, we first note that M accepts x iff for some reachable query configuration (q, i, z, w) in G, an accepting state is not reachable in $G_{T,x\#w}$. This is true iff the node q' is not reachable from the initial node of $G_{T,x\#w}$, which in turn is true iff there exists some IRC in $G_{T,x\#w}$ not containing q'. Thus, by Lemma 3.1 we have that L_1 is $\Sigma_2^{L[1]}$-hard. □

THEOREM 3.2. *The language* $L_2=\{G \mid G \text{ has an IRC}\}$ *is complete for* $\Sigma_2^{L[1]}$.

Proof. We first show that L_2 is in $\Sigma_2^{L[1]}$. Let $G=(V, E)$ be an I-graph. Let v be a new node not in V. Then the I-graph $G'=(V \cup \{v\}, E)$ has an IRC that does not contain v iff G has an IRC. Hence by Lemma 3.1 and Theorem 3.1 we have $L_2 \in \Sigma_2^{L[1]}$.

To show that L_2 is $\Sigma_2^{L[1]}$-hard consider an instance (G, u) of L_1 where $G=(V, E)$ is an I-graph and u is a node in V. We construct a new graph $G'=(V', E')$ by adding

a new node u' such that $V' = V \cup \{u'\}$ and $E' = E \cup \{(u, u')\}$. In other words, G' is constructed by connecting u to a deadend node u'. In this way, u is not weakly live in G iff G' has an IRC. The theorem now follows from Theorem 3.1. □

4. ω-machines with fairness constraints. Our main concern in this section is to define ω-machines where acceptance is defined as in [4], [18] but with the additional criteria that all accepting computations satisfy a fairness constraint. Then we investigate the complexity of the *Nonemptiness Problem* (NEP) for such machines. It turns out that their complexities fall into one of the following categories:

(1) *NL*-complete,
(2) $\Sigma_2^{L[1]}$-complete (the second level of the restricted logspace oracle hierarchy),
(3) $\Sigma_3^{L[1]}$-complete (the third level of the restricted logspace oracle hierarchy),
(4) PTIME-complete.

4.1. ω-finite state machines. A *nondeterministic finite state machine* (FSM) M is a 4-tuple (Q, Σ, q_0, δ) where

Q is the finite set of states,

Σ is the finite set of input symbols,

q_0 is the initial state, and

$\delta: Q \times \Sigma \to 2^Q$ is the transition function.

Let Σ^ω denote the set of infinite strings over Σ. Let $\sigma = a_1, a_2, \cdots$, be an infinite sequence of input symbols, i.e., an element in Σ^ω. A *run* r on σ in the state q is an infinite sequence of states $q_1, q_2, \cdots$, such that $q_1 = q$ and for every $1 \leqq i$, $q_{i+1} \in \delta(q_i, a_i)$. For a run $r = q_1, q_2, \cdots$, we define $P_r(i) = q_i$, i.e., the ith state in the sequence. Letting q be a state, we use $\#_r(q)$ to denote the number of occurrences of q in the run r. Let $\#_r(q) = \infty$ iff q occurs infinitely often in r. We also define $\mathrm{Inf}(r) = \{q \mid \#_r(q) = \infty\}$. In other words, $\mathrm{Inf}(r)$ is the set of states that occur infinitely often in r. An ω-FSM is a 5-tuple $M = (Q, \Sigma, q_0, \delta, F)$ where $M_1 = (Q, \Sigma, q_0, \delta)$ is an FSM and $F \subseteq 2^Q$ is called the set of *designated state sets.* We now introduce the notion of "*i-acceptance,*" where $i = 1, 1', 2, 2'$, or 3, as was defined in [18]. See also [4]. Given an input σ, let r be *any* run on σ in the initial state q_0. We say:

(1) σ is 1-*accepted* by r iff $\exists H \in F, \exists i: P_r(i) \in H$.
(2) σ is 1′-*accepted* by r iff $\exists H \in F, \forall i: P_r(i) \in H$.
(3) σ is 2-*accepted* by r iff $\exists H \in F: \mathrm{Inf}(r) \cap H \neq \varnothing$.
(4) σ is 2′-*accepted* by r iff $\exists H \in F: \mathrm{Inf}(r) \subseteq H$.
(5) σ is 3-*accepted* by r iff $\mathrm{Inf}(r) \in F$.

We define $L_i(M) = \{\sigma \mid \exists$ a run r of M on σ such that σ is i-accepted by $r\}$ to be the language accepted by M in i-acceptance mode. Informally speaking, σ is 1-accepted if r visits some state which belongs to some designated state set in F. σ is 1′-accepted if there exists a set H in F such that every state in r is also in H. σ is said to be 2-accepted if the set of states that appear infinitely often in r contains some state from some designated state set in F. σ is 2′-accepted if the set of states that occur infinitely often is a subset of some designated set in F. Finally, σ is 3-accepted if the set of states entered infinitely often in r coincides exactly with some designated state set in F.

Given an ω-FSM M, the NEP for ω-FSM's is to decide whether M accepts any input. The following theorem is easily shown.

THEOREM 4.1. *For each* $i = 1, 1', 2, 2'$ *or* 3, *the NEP with respect to i-acceptance* ("*Is* $L_i(M) \neq \emptyset$?") *is NL-complete.*

In what follows, we introduce the notion of *fair acceptance* for ω-FSM's. Let $M = (Q, \Sigma, q_0, \delta, F)$ be an ω-FSM. Let $r (= q_0, q_1, \cdots)$ be a run over the input $\sigma (= a_0, a_1, \cdots)$. With respect to r and σ, we can define an infinite sequence of transitions E: $e_0, e_1, \cdots$, such that, e_i is the transition $(q_i, a_i) \to q_{i+1}$. The run r is said to be *strongly fair* iff for every state q such that $\#_r(q) = \infty$, every outgoing transition of q must occur infinitely often in E. The run r is said to be *fair* if whenever a transition e: $(q, a) \to q'$ occurs infinitely often in E, then every transition $(q, a) \to q''$ in $\delta(q, a)$ must also occur infinitely often in E. Note that strong fairness is applied simultaneously to all characters of the input alphabet while fairness is applied on a character by character basis. Furthermore, note that the two notions become identical for single letter input alphabets. (Hence, a fair run only requires those transitions that have been "enabled" infinitely often be executed infinitely often.) We then define the language of fair (strongly fair) i-acceptance by an ω-FSM M to be $L_i^f(M) = \{\sigma \mid \exists$ a fair run r of M on σ such that σ is i-accepted by $r\}$ ($L_i^{sf}(M) = \{\sigma \mid \exists$ a strongly fair run r of M on σ such that σ is i-accepted by $r\}$). Given an ω-FSM M, the fair (strongly fair) NEP will be to determine whether $L_i^f(M) \neq \emptyset$ ($L_i^{sf}(M) \neq \emptyset$).

When reasoning about fair and strongly fair runs it sometimes becomes useful to consider the ω-FSM M as if it were an I-graph with q_0 as its initial node. The nodes (edges) of the I-graph correspond to the states (transitions) of M. If $p, q \in Q$ then (p, q) is an edge in the I-graph iff $q \in \delta(p, a)$ for some $a \in \Sigma$. We call this I-graph M's associated I-graph.

We first consider 3-acceptance.

THEOREM 4.2. *The fair* (*strongly fair*) *NEP with respect to* 3*-acceptance* ("*Is* $L_3^f(M) \neq \emptyset$ (*or* $L_3^{sf}(M) \neq \emptyset$)?") *is NL-complete.*

Sketch of proof. We only show that the problem can be solved in *NL*. The fact that the problem is *NL*-hard can be derived using a standard reduction from the directed graph reachability problem [29]. We first consider the case of strong fairness. Consider an arbitrary ω-FSM $M = (M_1, \{H_1, H_2, \cdots, H_k\})$, where M_1 is an FSM and $\{H_1, \cdots, H_k\}$ is the set of designated state sets. M accepts some input iff there is a strongly fair run r such that $\inf(r) = H_i$, for some i. We now describe an $O(\log n)$ space bounded NTM W that accepts M iff $L_3^{sf}(M) \neq \emptyset$. First W guesses a value for i, $1 \leqq i \leqq k$. Note that the set H_i is part of the input tape; let $\$h_1, h_2, \cdots, h_m\#$ represent this part. To do the checking, two pointers P_1 and P_2 are used. P_1 is associated with H_i and indicates those states in H_i that W currently recalls having visited. Initially P_1 points to h_1. The checking procedure basically consists of two stages. In the first stage, P_2 will be used as a pointer to traverse the state transition graph of M_1 (initially P_2 points to the initial state of M_1), while P_1 remains unchanged. During the traversing, W nondeterministically "guesses" at some point that the current state is in an IRC that contains exactly those states in H_i; and then proceeds to the second stage. In the second stage, assume that P_1 is currently pointing at h_j, $1 \leqq j \leqq m$. Similar to the first stage, P_2 is used to traverse the graph. However, some extra checking is needed at each step. When visiting a state, say p, W first tests whether every adjacent state q of p (i.e. every $q \in \delta(p, a)$ for some input symbol a) is in H_i. If not, the checking procedure terminates unsuccessfully; otherwise, one of the adjacent q's, say q', will be nondeterministically chosen to be visited next. Furthermore, if $q' = h_{j+1}$, P_1 will be advanced to the next position. This procedure in stage 2 repeats until P_1 hits the $\#$. Then, the above traversing continues until the state h_1 is again visited. At this time W accepts the input. It is now reasonably easy to see that W accepts such an input iff there is a

strongly fair run that visits exactly the states in H_i in this manner. Since the space needed for the pointers is logarithmic, the strongly fair NEP can, therefore, be solved in *NL*.

A similar argument can be applied for the fair case. The main modification will be that, for each state, we only have to test those states with the same label (i.e., the same input symbol) to see if they are in H_i. Further details are left to the reader. □

This theorem tells us that, type 3-acceptance is, in some sense, easier to analyze than other types of acceptance. In the remainder of this section, we only consider type 1, 1′, 2, and 2′-acceptance modes.

LEMMA 4.1. *For each* $i = 1, 1', 2,$ *or* $2'$, *the fair NEP with respect to i-acceptance* ("*Is* $L_i^f(M) \neq \emptyset$?") *is* PTIME-*hard.*

Proof. To show this, we reduce the path system problem, which is well known to be PTIME-complete [14], to the fair NEP for ω-FSM's. Recall that a Path System is a 4-tuple $\mathcal{P} = (X, R, S, T)$ where X is a finite set of nodes, S ($\subseteq X$) is a set of starting nodes, T ($\subseteq X$) is the set of terminal nodes, and R ($\subseteq X \times X \times X$) is the set of rules. A node x in X is said to be *admissible* iff either $x \in T$ or $\exists y, z \in X$ such that $(x, y, z) \in R$ and both y and z are admissible. The Path System $\mathcal{P} = (X, R, S, T)$ is said to have a solution iff there is an admissible node in S.

Now, given the Path System $\mathcal{P}$ we want to construct, for each $i = 1, 1', 2,$ or $2'$, an ω-FSM M_i such that $L_i^f(M_i) \neq \emptyset$ iff $\mathcal{P}$ has no solution. Let $S = \{s_1, s_2, \cdots, s_k\}$, $T = \{t_1, t_2, \cdots, t_h\}$, and $R = \{r_1, \cdots, r_d\}$. To do this, we define $M_i = (Q, \Sigma, q_0, \delta, F)$ to be:

$Q = X \cup \{q_0, q'\} \cup \{x^1, x^2, \cdots, x^d \mid \forall x \in X - T\}$,
$\Sigma = \{a, b\}$,
δ: (Let "$q \in \delta(p, a/b)$" mean that q is in both $\delta(p, a)$ and $\delta(p, b)$.)
(1) $s_1 \in \delta(q_0, a/b)$,
(2) $\forall 1 \leqq j \leqq k-1$, $s_{j+1} \in \delta(s_j, a/b)$ and $s_1 \in \delta(s_k, a/b)$,
(3) $\forall 1 \leqq j \leqq h$, $q' \in \delta(t_j, a/b)$,
(4) $\forall x \in (X - T)$, $q_0 \in \delta(x, a/b)$,
(5) $\forall x \in X - T$, $\forall j, 1 \leqq j \leqq d-1$, $x^1 \in \delta(x, a/b)$, $x^{j+1} \in \delta(x^j, a/b)$, and $x \in \delta(x^d, a/b)$,
(6) $\forall x \in (X - T)$, if $r_j = (x, y, z) \in R$ then $y \in \delta(x^j, a)$ and $z \in \delta(x^j, b)$.
$F = \{Q - T\}$.

See Fig. 4.1. For each $i = 1, 1', 2$ or $2'$, we claim that $\mathcal{P}$ has no solution iff M_i has a fair run, i.e., $L_i^f(M_i) \neq \emptyset$.

To show this, first assume that $\mathcal{P}$ has no solution. Let A and B be the sets of nonadmissible and admissible nodes, respectively. Let $A' = A \cup \{x^1, x^2, \cdots, x^d \mid x \in A\}$. Let $B' = B \cup \{x^1, x^2, \cdots, x^d \mid x \in B - T\}$. Clearly, $S \subseteq A'$. According to the construction of M, for every node x^j in A' there exists at most one edge that connects x^j to some node in B'. (Otherwise, the corresponding x would be admissible.) Furthermore, for each state in A' ($\subseteq Q - T$), there are edges back to q_0. Consequently, there exists a fair run, say r, that only visits states in A' ($\subseteq Q - T$). Since $F = \{Q - T\}$, the input associated with r will be accepted by M_i, for $i = 1, 1', 2,$ or $2'$.

Now consider the case when $\mathcal{P}$ has a solution, i.e., some state, say s', in S is admissible. From the construction, we show that there does not exist a fair run on M_i. Let H be the "parse" tree that certifies the admissibility of s'. That is, H is a finite labelled binary tree where:

- each node in H is either a leaf or has both a left child and a right child,

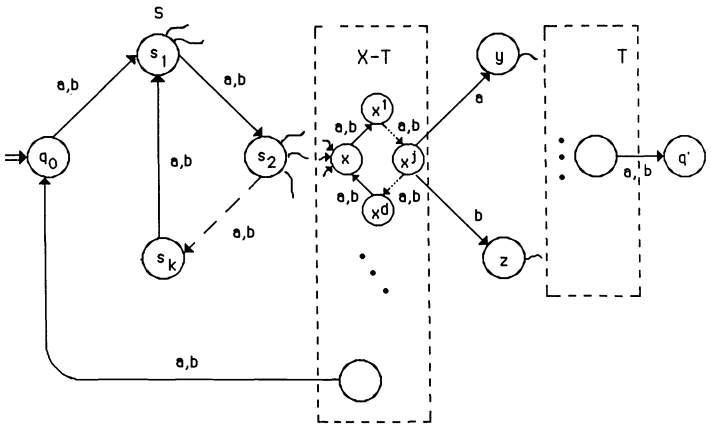


FIG. 4.1. *The ω-FSM M_i.*

- each node is labelled by an element of X,
- each leaf is labelled by an element of T,
- whenever an internal node is labelled by x and its left and right children are labelled by y and z, respectively, it must be that $(x, y, z) \in R$,
- the root is labelled by s'.

Clearly, each node in such a tree must be labelled by an admissible element in X.

Suppose that for some input string there exists a fair run r on M_i ($i = 1, 1', 2,$ or $2'$). (Recall that r is of infinite length.) By construction it must be the case that all states in S occur infinitely often in r. We now claim that there must exist a leaf of H whose label occurs infinitely often in r. Now recall that leaves are labelled by elements of T. As a result, the construction insures that q' must occur in r which is impossible. Hence, for $i = 1, 1', 2,$ or $2'$, a fair run cannot exist on M_i.

It only remains to show the claim. Let H' be the set of nodes in H whose labels belong to $\inf(r)$. Consider a node in H' of maximal depth. Let x label this node. If this node is a leaf the claim is true. So suppose the node is not a leaf and that y and z are the labels associated with its left and right children respectively. Now $(x, y, z) \in R$. Suppose that (x, y, z) is the jth rule in R. Now by construction x^j must occur infinitely often in r; hence, y or z must also appear infinitely often in r. This contradicts the fact that the chosen node in H was of maximal depth. As a result, the claim is true. □

In § 5, the fair NEP for ω-FSM's with respect to i-acceptance ($i = 1, 1', 2,$ and $2'$) will be reduced to the canonical form model checking problem for Fair Computation Tree Logic, which is known to be solvable in PTIME [7]. (See Lemma 5.1 and Corollary 5.1.) Consequently, we have the following result.

COROLLARY 4.1. *For $i = 1, 1', 2,$ or $2'$, the fair NEP for ω-FSM's with respect to i-acceptance is* PTIME*-complete.*

Notice that the proof of the previous lemma required that the constructed ω-FSM have two distinct input symbols. An interesting question arises if we restrict the problem to ω-FSM's over single letter input alphabets. In this case, it turns out that most of the aforementioned problems are $\Sigma_2^{L[1]}$-complete.

THEOREM 4.3. *For $i = 1, 1',$ or $2'$, the fair NEP for ω-FSM's over a 1-letter alphabet with respect to i-acceptance is $\Sigma_2^{L[1]}$-complete.*

Proof. Notice that a fair run for an ω-FSM over a 1-letter alphabet is a strongly fair run, and vice versa. Based on this observation, we first show that the fair NEP is

$\Sigma_2^{L[1]}$-hard. To show this, we reduce the problem L_2 mentioned in § 3 (the problem, given an I-graph $G=(V, E)$, to determine if G contains an IRC), to the fair NEP for ω-FSM's. The reduction simply involves constructing an ω-FSM M where the states (transitions) of M correspond to the nodes (edges) of G, and F (the designated state set) contains the set of all nodes in G. In this way, for all fair accepting modes (i.e., 1, 1′, and 2′), M accepts some input iff G has an IRC.

Next, we show that the fair NEP over single letter alphabets is in $\Sigma_2^{L[1]}$. Consider an ω-FSM $M=(Q, \{a\}, q_0, \delta, \{H_1, H_2, \cdots, H_k\})$. We first construct M' from M by adding a new state q' and transitions from every deadend state (a state with no outgoing transitions) to q'. (This is similar to the construction for I-graphs used in the proof of Theorem 3.1.) In the subsequent discussion, we consider the associated I-graphs of M and M'. In what follows, we construct an OTM M^i and an oracle set A^i, for each i, such that M^i will accept exactly those machines M where $L_i^f(M)\neq\varnothing$. Each M^i is similar to the OTM constructed in Theorem 3.1. However, for different accepting modes, additional checking steps are required. Let:

$A^1=\{M\#j\$p_u \mid$ one of the following is true:

- M is not an ω-FSM,
- j does not represent the index of one of M's designated state sets,
- p_u is not a pointer to a state, say u, in M, or
- $u \xrightarrow{*} q'$ in M"s associated I-graph$\}$.

Let:

$A^{1'}=A^{2'}=\{M\#j\$p_u \mid$ one of the following is true:

- M is not an ω-FSM,
- j does not represent the index of one of M's designated state sets,
- p_u is not a pointer to a state, say u, in M,
- $u \xrightarrow{*} q'$ in M"s associated I-graph, or
- $\exists w \notin H_j\, u \xrightarrow{*} w$ in M"s associated I-graph$\}$.

These sets can easily be seen to be in NL. Intuitively speaking, $M\#j\$p_u$ is not in A^1 if u cannot reach the deadend node q' in M', and hence u can reach some IRC in M's associated I-graph. Similarly, $M\#j\$p_u$ is not in $A^{1'}$ ($A^{2'}$) if u can reach some IRC in M's associated I-graph and the set of states in that IRC is a subset of H_j. Now, M^i ($i=1, 1', 2'$) on input M operates as follows:

(1) M^i guesses a value j, $1\leqq j\leqq k$.

(2) M^i nondeterministically traverses the state graph of M until some state say u (chosen nondeterministically) is visited. If $i=1$ ($1'$) some (each) state visited thus far must be in H_j; if $i=2'$ u must be in H_j.

(3) M^i then accepts iff $M\#j\$p_u\notin A^i$.

The reader can now verify that M^i accepts M iff $L_i^f(M)\neq\varnothing$. Notice that each OTM mentioned above needs to use $O(\log n)$ space for its working tape, to write one string of logarithmic length on the query tape, and to make one oracle call. The theorem then follows. □

COROLLARY 4.2. *For each $i=1$, $1'$, or $2'$, the strongly fair NEP for ω-FSM's (over 1-letter or multiletter alphabets) with respect to i-acceptance is $\Sigma_2^{L[1]}$-complete.*

Proof. In a strongly fair run, it must be the case that for every state occurring infinitely often that all of its outgoing transitions are traversed infinitely often. This is essentially the same problem then as the fair NEP over a 1-letter alphabet that has just been shown to be $\Sigma_2^{L[1]}$-complete. □

Theorem 4.3 (and Corollary 4.2) do not seem to hold for 2-acceptance. We prove, in § 4.3, that the analogous problems for 2-acceptance are complete for $\Sigma_3^{L[1]}$.

4.2. ω-one counter machines. An ω-1CM is a 7-tuple $M=(Q,\Sigma,\Upsilon,q_0,\delta,Z_0,F)$, where Q is the set of states, Σ is the input alphabet, $\Upsilon=\{Z_0,B\}$ is the stack alphabet, q_0 is the initial state, $\delta: Q\times\Sigma\times\Upsilon\to 2^{Q\times\{-1,0,+1\}}$ is the transition function, Z_0 is the bottom-of-stack marker (Z_0 can neither be written nor erased), and F ($\subseteq 2^Q$) is the set of designated state sets. Now, since Z_0 can neither be written nor erased, the stack essentially functions as a counter capable of maintaining any nonnegative integer. The value of the counter (at any given time) is the number of B's on the stack. The actions of the ω-1CM depend on the internal state of the machine, the current input symbol, and the status of the counter (i.e., whether there are zero or a positive number of B's currently on the stack). A move or transition of the machine involves changing the machine's state plus (possibly) altering the value of the counter by at most 1. In the definition of δ above, -1 represents the action of "popping" a B from the stack (i.e., subtracting 1 from the counter), 0 represents the action of not altering the stack and, $+1$ represents the action of "pushing" a B on the stack (i.e., adding 1 to the counter). Roughly speaking, ω-1CM's operate in a similar way as conventional 1CM's except that, we are interested in those inputs over Σ^ω instead of Σ^*. (See Ibarra and Rosier [13] and Valiant and Patterson [31] for detailed discussions involving 1CM's.) We define a configuration of an ω-1CM to be a 3-tuple (q,j,h) where q is the current state of the machine, j is the current input head position and h is the value of the counter. The configuration (q,j,h) is said to be in state q. Let $c_0=(q_0,0,0)$ denote the initial configuration. Let c_1 and c_2 be two configurations. We use $c_1\vdash c_2$ ($c_2\vdash^a c_2$) to denote that c_1 can lead to c_2 in a single computation step (that reads "a" ($\in\Sigma$)). Let $c_1\vdash^* c_2$ represent the reflexive transitive closure of the "$\vdash$" relation. A computation path is a sequence $c_1\vdash c_2\vdash\cdots\vdash c_l$ where $c_i\vdash c_{i+1}$, $1\leqq i\leqq l-1$. If $c_i\vdash^{a_i} c_{i+1}$, $1\leqq i\leqq l-1$, and $x=a_1\cdots a_{l-1}$ we sometimes speak of the computation $c_i\vdash^x c_l$. An infinite computation path is an infinite sequence $c_1\vdash c_2\vdash\cdots$ where $c_i\vdash c_{i+1}$, $\forall i\geqq 1$. For finite computations e: $c_1\vdash\cdots\vdash c_2$ and e': $c_2\vdash\cdots\vdash c_3$, let $e;e'$ denote the computation $c_1\vdash\cdots\vdash c_2\vdash\cdots\vdash c_3$. A loop is a computation path $c_1\vdash c_2\vdash\cdots\vdash c_l$ such that c_1 and c_l are in the same state. A short loop is a loop in which only c_1 and c_l share the same state.

For $i=1$, $1'$, 2, $2'$, and 3, the definition of i-acceptance is essentially the same as it was for ω-FSM's. For example, the input read over an infinite computation path that begins with c_0 is 2-accepted iff $\exists H\in F$ such that the path contains infinitely many configurations that are in some state in H. An infinite computation path that begins with c_0 is said to be a strongly fair run iff whenever there are infinitely many configurations in a state q in the run then every outgoing transition of q must be used infinitely often in the run. For the definition of a fair run, the notion of when a transition is "enabled" is essential. A transition is said to be enabled in a configuration iff the move defined by the transition can be taken. An infinite computation path that begins with c_0 is said to be a fair run iff every transition that is enabled infinitely often is executed infinitely often. Based on these notions, fair (strongly fair) i-acceptance ($i=1$, $1'$, 2, $2'$, and 3) is defined in the same manner as it was for ω-FSM's.

When reasoning about strongly fair ω-1CM computations it sometimes becomes useful to consider the ω-1CM M as if it were an I-graph with q_0 as its initial node. The nodes (edges) of the I-graph correspond to the states (transitions) of M. If $p,q\in Q$ then (p,q) is an edge in the I-graph iff $(q,\Delta)\in\delta(p,a,Z)$ for some $\Delta\in\{-1,0,1\}$, $a\in\Sigma$, $Z\in\Upsilon$. (Note that the computational meaning associated with a transition is lost in the I-graph.) We call this I-graph M's associated I-graph.

Since ω-FSM's are just special cases of ω-1CM's, the fair NEP for ω-1CM's with respect to i-acceptance ($i=1$, $1'$, 2, and $2'$) is at least PTIME-hard. The problems are

also in PTIME. This fact will be shown in § 5. (See Theorem 5.2.) In what follows, we focus on those ω-1CM's over 1-letter input alphabets.

LEMMA 4.2. *For each $i = 1, 1', 2,$ or $2'$, the fair NEP for ω-1CM's over a 1-letter alphabet with respect to i-acceptance is* PTIME*-hard.*

Sketch of proof. The proof of this lemma is almost identical to that of Lemma 4.1. Here we show how to construct an ω-1CM M_i', over a single letter input alphabet, that has the same properties as the ω-FSM M_i constructed in Lemma 4.1. (See Fig. 4.1.) Recall that $M_i = (Q, \{a, b\}, q_0, \delta, F)$. $M_i' = (Q', \{a\}, \{Z_0, B\}, q_0, \delta', Z_0, F')$ is constructed from M_i as follows. Each state of M_i is also a state of M_i' (i.e., $Q \subseteq Q'$). For each of the special states x^j of M_i, M_i' has states x^j, x_1^j and x_2^j. $F' = F \cup (Q' - Q)$. Whenever M_i has the transitions "$p \xrightarrow{a,b} q$," M_i' has the single transition $(q, 0) \in \delta(p, a, Z_0)$. Now recall from the proof of Lemma 4.1, that M_i's 2-letter alphabet $\{a, b\}$ allowed that only one of the transitions "$x^j \xrightarrow{a} y$" and "$x^j \xrightarrow{b} z$" need be enabled at x^j during an infinite computation. M_i''s counter will assume this responsibility. The pair of transitions "$x^j \xrightarrow{a} y$" and "$x^j \xrightarrow{b} z$" in M_i are replaced by the transitions of M_i' shown in Fig. 4.2. There an edge "$p - (\gamma, \Delta) \to q$" represents the transition $(q, \Delta) \in \delta(p, a, \gamma)$ in M_i'. This completes the construction of M_i'. The transition "$x^j \xrightarrow{a} y$" ("$x^j \xrightarrow{b} z$") in M_i can now be simulated in M_i' by the sequence of transitions "$x^j - (Z_0, 0) \to x_1^j - (Z_0, +1) \to x_1^j - (B, -1) \to x_1^j - (Z_0, 0) \to x_2^j - (Z_0, 0) \to y$" ("$x^j - (Z_0, 0) \to x_1^j - (Z_0, +1) \to x_1^j - (B, -1) \to x_1^j - (Z_0, +1) \to x_1^j - (B, 0) \to x_2^j - (B, -1) \to z$"). Note that whenever x_2^j is reached that only one of its outgoing transitions is enabled.

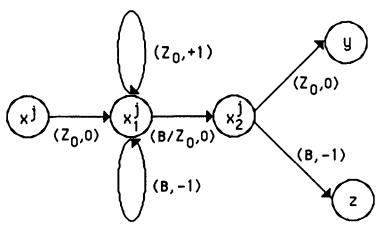


FIG. 4.2. *A mechanism to enable a path from x^j to y or z but not both.*

Now consider a fair computation of M_i'. Every computation of M_i' begins with the counter equal to zero. Hence, whenever the first x^j state is reached the counter will be zero. Furthermore, the "mechanism" shown in Fig. 4.2 guarantees that if, as a result, state y (or z) is reached the counter once again has the value zero. The proof now degenerates into showing that M_i' has the properties required of M_i in Lemma 4.1. The remaining details are left to the reader. □

In what follows, we show that the strongly fair NEP for ω-1CM's with respect to i-acceptance ($i = 1, 1', 2,$ and $2'$) is in $\Sigma_3^{I[1]}$. Let M be an ω-1CM. Let F be its set of designated state sets. Let r be a strongly fair accepting computation of M (with respect to any of the acceptance modes). Now it is not difficult to show that there must exist a prefix of r, $r' = e$; e' (e: $c_0 \vdash \cdots \vdash (v, j, h)$ and e': $(v, j, h) \vdash \cdots \vdash (v, j'', h'')$) with the following properties: (Let S (T, respectively) denote the set of states visited (transitions used, respectively) in e'.)

- $h'' \geqq h$.
- T = the set of transitions used infinitely often in $r = \{t \mid t$ an outgoing transition of some state in $S\}$.
- If $h'' > h$ then no transition in T can require the counter to be zero.

- For 1 (1′) acceptance at least one (each) state visited in r' must belong to some $H \in F$.
- For 2 acceptance v must belong to some $h \in F$.
- For 2′ acceptance S must be a subset of some $H \in F$.

Conversely, if such a prefix exists for $i = 1$, 1′, 2, or 2′, then it is easy to show that $L_i^{sf}(M) \neq \emptyset$. As a result, we claim that M admits a strongly fair accepting computation with respect to i-acceptance iff M admits a finite computation $r' = e; e'$ (e: $c_0 \vdash \cdots \vdash (v, j, h)$ and e': $(v, j, h) \vdash \cdots \vdash (v, j'', h'')$) that satisfies one of the conditions (A) or (B) below:

(A) $h = h'' = 0$ and $\exists H \in F$ such that
- If $i = 1$ then at least one state visited in e belongs to H.
- If $i = 1'$ then each state visited in e belongs to H.
- If $i = 2$ or 2′ then v belongs to H.

And, $\forall$ state w in M, either
(a) $\neg(v \overset{*}{\rightarrow} w)$ in M's associated I-graph, or
(b) e' exercises or uses each outgoing transition of w at least once. If $i = 1'$ or 2′ then each state visited in e' must belong to H.

(B) $h'' \geqq h$ and $\exists H \in F$ such that
- If $i = 1$ then at least one state visited in e belongs to H.
- If $i = 1'$ then each state visited in e belongs to H.
- If $i = 2$ or 2′ then v belongs to H.

And, $\forall$ state w in M, either
(a) $\neg(v \overset{*}{\rightarrow} w)$ in M's associated I-graph, or
(b) e' exercises or uses each outgoing transition of w at least once but uses no transition requiring a zero counter. If $i = 1'$ or 2′ then each state visited in e' must belong to H.

The proof of this claim is straightforward and is left to the reader. Next we show that if there exists a computation of M which satisfies (A) (or (B)), then there exists a "short" such computation. By short we mean that the computation is bounded (in length) by $P(|M|)$, where P is a polynomial that is independent of M. The pumping lemma for pushdown automata would yield the existence of a short path; unfortunately the resulting length could be exponential in $|M|$. Our approach uses techniques from [11], and requires that we first establish a bound on the value of the counter during such a computation. The following lemma insures that if a computation path satisfying (A) exists, there must exist a shorter one in which the counter value remains "small."

LEMMA 4.3. *Let M be an ω-1CM with s transitions. Suppose (q, m, d) is a reachable configuration. If for some input string x, r: $(q, m, d) \vdash^{x} (q', m', d')$ is a computation. Then there exists a string x' and a computation r': $(q, m, d) \vdash^{x'} (q', m'', d')$ with the following properties*:

(1) *the set of transitions used in r is identical to the set of transitions used in r',*

(2) *for every configuration (p, n, h) in r', $h \leqq 3^* s^3 + \max\{d, d'\}$.*

Proof. Let $g = \max\{d, d'\}$. First consider the path r. Suppose at some point, say b, the counter exceeds $3^* s^3 + g$. Let a be the most recent time before b that the counter value was $s^3 + g$, and let c be the next time after b that the counter value becomes $s^3 + g$ again. See Fig. 4.3.

Define intervals (a_i, b_i), (b_i', c_i), $0 \leqq i \leqq 2^* s - 1$, where a_i is the last time before b in which the counter value is $s^3 + i^* s + g$, b_i is the first time after a_i in which the counter value is $s^3 + (i+1)^* s + g$, b_i' is the last time before c in which the counter value is $s^3 + (i+1)^* s + g$, and c_i is the first time after b_i' in which the counter value is $s^3 + i^* s + g$. Now between times a_i and b_i (b_i' and c_i), r must contain some short loop, say l_i (l_i'),

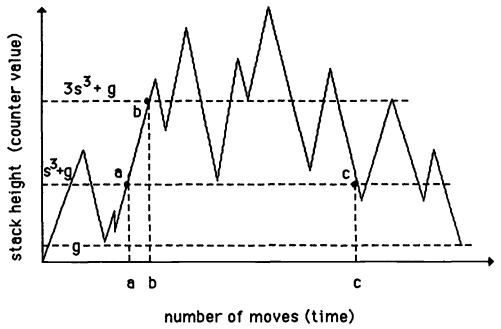


FIG. 4.3. *The computation r.*

in which there was a gain (loss) in the counter value of between 1 and s. This must be the case, since during the interval (a_i, b_i) $((b'_i, c_i))$, the counter gained (lost) s in value and at least s steps were executed.

Now there are 2^*s^2 such short loops between points a and b (b and c). Hence, there must exist a k (j), $1 \leqq k$ $(j) \leqq s$, such that at least 2^*s loops have a gain (loss) of exactly k (j) in the value of the counter. Let L_k (L'_j) denote the set of such loops. Now L_k (L'_j) can be partitioned into two sets L_{k1} and L_{k2} (L'_{j1} and L'_{j2}) such that every transition which occurs in some loop in L_k (L'_j) occurs in some loop in L_{k1} (L'_{j1}). Furthermore, the partition can be done in such a way that $|L_{k1}| \leqq s$ $(|L'_{j1}| \leqq s)$. Assume that this is the case. Let u and v be such that $k^*u = j^*v =$ the least common multiple of k and j, $1 \leqq u, v \leqq s$. Now from r, we can create a shorter computation by removing u loops from L_{k2} and v loops from L'_{j2}. Note that the resulting computation is also valid and ends up at the same final configuration, since the total increase in the counter value by the removed u loops is exactly offset by the total loss caused by the removed v loops. Furthermore, each transition that occurs in r occurs in the shorter computation. This procedure of shortening r can be repeated until there is no time during the resulting computation in which the counter value exceeds 3^*s^3+g. □

By letting $d = d' = 0$ in the preceding lemma, we have that if some computation path satisfying (A) exists, there exists another one on which the counter never exceeds 3^*s^3.

The next lemma insures that if a computation path satisfying (B) exists, there must exist a shorter one in which the counter value remains small.

LEMMA 4.4. *Let M be an ω-1CM with s transitions. If there is a computation path* r: $c_0 \vdash \cdots \vdash (v, i, h) \vdash \cdots \vdash (v', i', h')$, *where* $i' > i$, $h' \geqq h$, *and no transition in* $(v, i, h) \vdash \cdots \vdash (v', i', h')$ *requires zero in the counter then there must exist another computation path* r': $c_0 \vdash \cdots \vdash (v, j, k) \vdash \cdots \vdash (v', j', k')$ *satisfying*:

- $j' > j$,
- $k' \geqq k$,
- *the set of transitions used in* $(v, i, h) \vdash \cdots \vdash (v', i', h')$ *is identical to the set of transitions used in* $(v, j, k) \vdash \cdots \vdash (v', j', k')$,
- *the counter never exceeds* $6^*s^3+s^2+2^*s$ *in* r'.

Proof. The proof here is very similar in flavor to that of the previous lemma. As a result only a sketch of the main idea is provided here. First, r can be divided into two subpaths e: $c_0 \vdash \cdots \vdash (v, i, h)$ and e': $(v, i, h) \vdash \cdots \vdash (v', i', h')$. In what follows, we first show how to shorten e', then e.

Suppose $h' > h+(s^2+s)$. Using the same ideas as were employed in the proof of Lemma 4.3, one can find in e' at least $s+1$ short loops each of which causes a gain

of between 1 and s in the counter. (These loops can be found in places after which the counter remains above s. Hence, each such loop is "removable" in the sense that the loop can be removed from e' and a valid computation remains.) Since at most s transitions are used in e', one of these loops can be removed and the resulting computation still uses each of the transitions used in e'. Furthermore, this procedure can be repeated until a computation path $(v, i, h) \vdash \cdots \vdash (v', i'', h'')$ is found where $i'' > i$ and $h \leqq h'' \leqq h + (s^2 + s)$. Now, by applying the result of Lemma 4.3 directly to this path we can further shorten the path so that the counter never exceeds $h + (s^2 + s) + 3^* s^3$. Let δ be such a computation path.

Now, for the prefix e, consider the following two cases.

CASE 1. $h \leqq 3^* s^3 + s$. In this case, we can use the result of Lemma 4.3 directly to shorten the computation e so that the counter never exceeds $(3^* s^3 + s) + 3^* s^3$. The lemma now follows directly.

CASE 2. $h > 3^* s^3 + s$. We first claim that δ can be shortened in such a way that the counter never drops below s. (The proof is almost identical to that of Lemma 4.3, and hence is left to the reader.) Let δ': $(v, i, h) \vdash \cdots \vdash (v', i'', h'')$ be this shortened computation. Using the same ideas as before, one can find in e at least $s+1$ removable short loops each of which causes a gain of between 1 and s in the counter. Now one of these loops can be removed from e without changing the set of transitions used. Let the resulting computation be $c_0 \vdash \cdots \vdash (v, i''', h''')$, where $0 \leqq i''' \leqq i$ and $h - s \leqq h''' < h$. Note that the sequence of moves used in δ' can be executed from (v, i''', h''') to obtain the computation $(v, i''', h''') \vdash \cdots \vdash (v', i'' - (i - i'''), h'' - (h - h'''))$. If the computation $c_0 \vdash \cdots \vdash (v, i''', h''') \vdash \cdots \vdash (v', i'' - (i - i'''), h'' - (h - h'''))$ does not satisfy the lemma's requirements the entire process described here can be repeated until a computation is found that does. □

With Lemmas 4.3 and 4.4, we are now in position to show the polynomial P exists. Let M be an ω-1CM with s transitions. Let $p = 6^* s^3 + s^2 + 2^* s$. Now consider the "shortest" computation path of M that satisfies (A) (or (B)). Consider a step $(q, j, d) \vdash (q', j', d')$ in the path. If we ignore the value of j and j' there are at most $s^* p^2$ distinct steps in the computation. Mark the first and last steps as well as the step which results in the final configuration e. Now make a left to right pass over the computation and mark each step which is distinct from those already marked. Now between each of the marked steps there can be at most $s^* p$ distinct configurations; otherwise a segment of the form $(v, j, h) \vdash \cdots \vdash (v, j', h)$ could be removed from the computation without affecting the path's required properties. Hence the length of the computation is bounded by $(s^* p)^* s^* p^2$. As a result, we have that the length of M's shortest computation path satisfying (A) (or (B)) is bounded by $P(|M|) = |M|^2 * (6^* |M|^3 + |M|^2 + 2^* |M|)^3$.

We are now ready to show the following.

LEMMA 4.5. *For each $i = 1, 1', 2,$ or $2'$, the strongly fair NEP for ω-1CM's (over 1-letter or multiletter alphabets) with respect to i-acceptance is in $\Sigma_3^{L[1]}$.*

Proof. Let $M = (Q, \Sigma, \Upsilon, q_0, \delta, Z_0, F)$ be an ω-1CM. Let $F = \{H_1, \cdots, H_k\}$ be its set of designated state sets. For $i = 1, 1', 2, 2'$, we show how to construct OTM's M^i (that make at most one oracle call during the course of a computation) and oracle sets A^i (in $\Sigma_2^{L[1]}$) such that M^i using the oracle A^i accepts M iff $L_i^{sf}(M) \neq \varnothing$. Now recall that M in i-acceptance mode accepts something iff there exists a computation path $r' = e; e'$ (e: $c_0 \vdash \cdots \vdash (v, j, h)$ and e': $(v, j, h) \vdash \cdots \vdash (v, j'', h'')$) of length $\leqq P(|M|)$ which satisfies either condition (A) or (B). But such a computation will exist iff there exists a computation path e: $c_0 \vdash \cdots \vdash (v, j, h)$ of length $\leqq P(|M|)$ such that (*) is satisfied:

(*) $\exists m, 1 \leqq m \leqq k$, such that

- If $i = 1$ then at least one state visited in e belongs to H_m.
- If $i = 1'$ then each state visited in e belongs to H_m.
- If $i = 2$ or $2'$ then v belongs to H_m.

and, either

(A)′ $h = 0$ and $\forall w \in Q$ either

(a) $\neg(v \xrightarrow{*} w)$ in M's associated I-graph, or

(b) there is a computation path $(v, j, h) \vdash \cdots \vdash (v, j'', h)$ of length $\leqq P(|M|)$ that exercises or uses each outgoing transition of w at least once. (If $i = 1'$ or $2'$ then each state visited on this path must belong to H_m.)

or

(B′) $\forall w \in Q$ either

(a) $\neg(v \xrightarrow{*} w)$ in M's associated I-graph, or

(b) $\exists h'' \geqq h$ such that there is a computation path $(v, j, h) \vdash \cdots \vdash (v, j'', h'')$ of length $\leqq P(|M|)$ that exercises or uses each outgoing transition of w at least once but uses no transition requiring a zero counter. (If $i = 1'$ or $2'$ then each state visited on this path must belong to H_m.)

Now, we want to show now such a computation path can be detected in $\Sigma_3^{L[1]}$. For $i = 1, 1', 2, 2'$, define:

$B^i = \{M \# p_v \$ j \$ h \$ m \$ t \$ p_w \mid p_v(p_w)$ is a pointer to a state, say $v\ (w)$, of M and $\exists j''\ (P(|M|) \geqq j'' > j)$, $\exists h''\ (h'' \geqq h)$ such that:

Either:

$h = 0$, $t = 1$, and there exists a computation path $(v, j, h) \vdash \cdots \vdash (v, j'', h)$ of length $\leqq P(|M|)$ that exercises each outgoing transition of w at least once. (If $i = 1'$ or $2'$, it is also required that every state visited on this path be in H_m.)

Or:

$t = 0$, and there exists a computation path $(v, j, h) \vdash \cdots \vdash (v, j'', h'')$ of length $\leqq P(|M|)$ that exercises each outgoing transition of w at least once but uses no transition requiring a zero counter. (If $i = 1'$, or $2'$, it is also required that every state visited on this path be in H_m).$\}$

It is easy to show that $B^i \in NL$. Now let:

$$A^i = \{M \# p_v \$ j \$ h \$ m \$ t \mid \exists w \in Q \text{ such that } v \xrightarrow{*} w \text{ in } M\text{'s associated } I\text{-graph and } M \# p_v \$ j \$ h \$ m \$ t \$ p_w \notin B^i\}.$$

Since B^i is in NL, we immediately have that $A^i \in \Sigma_2^{L[1]}$.

Now M^i on input M operates as follows:

(1) M^i guesses a value m, $1 \leqq m \leqq k$. M^i also guesses a value of 0 or 1 for t.

(2) M^i guesses (step by step) a computation (of M) of length less than $P(|M|)$ ending in say configuration (v, j, h). If $i = 1\ (1')$, then one (each) state visited in this computation must be in H_m. If $i = 2$ or $2'$ M^i requires v to be in H_m.

(3) M^i accepts M if $M \# p_v \$ j \$ h \$ m \$ t \notin A^i$.

The reader can now verify that M^i accepts the input M iff $L_i^{sf}(M) \neq \varnothing$. This proves the lemma. □

LEMMA 4.6. *For each $i = 1, 1', 2,$ or $2'$, the strongly fair NEP for ω-1CM's over 1-letter alphabets with respect to i-acceptance is $\Sigma_3^{L[1]}$-hard.*

Proof. As we know from Theorem 2.3, the restricted logspace oracle hierarchy is the same as the logspace alternation hierarchy. In what follows, we show that the problem is $A\Sigma_3^L$-hard. Let M be a 3 alternation bounded ATM and let x be an input string. A configuration of M on x is a 3-tuple (p, j, s) where p is a state, j an input

head position, and s the worktape contents. (s implicitly includes the position of the worktape heads.) The configuration (p, j, s) is called a universal (existential, accepting, rejecting, respectively) configuration iff p is a universal (existential, accepting, rejecting, respectively) state. Without loss of generality we can assume that each computation path of M on x has exactly 3 alternation blocks. Furthermore, we can assume that no configuration repeats on a path, and that each path results in either acceptance or rejection. Since M on input x uses only $O(\log|x|)$ space for its working tape, the number of distinct configurations is polynomial in $|x|$ (the length of x).

We now show how to construct an ω-1CM W to simulate the computation of M on x in such a way that M accepts x iff W accepts some input with respect to i-acceptance, $i = 1, 1', 2$, or $2'$. (The same ω-FSM W is constructed for each value of i.) Let c_0 be the initial configuration of M on x. Let V (E, U, respectively) be the set of all (existential, universal, respectively) configurations of M on x. Let T be a new symbol not in V. We define $W = (Q, \Sigma, \Upsilon, q_0, \delta, Z_0, F)$ as follows:

(1) $Q = V \cup ((V \cup \{T\}) \times V) \cup (V \times V \times V)$,

(2) $\Sigma = \{a\}$,

(3) $\Upsilon = \{Z_0, B\}$,

(4) $q_0 = c_0$,

(5) δ: For convenience, let "$(q, \Delta) \in \delta(p, a, Z_0/B)$" mean that (q, Δ) is in both $\delta(p, a, Z_0)$ and $\delta(p, a, B)$. Now for each configuration $p, q \in V$, $r \in U$ and $t \in E$, we have:

$(q, 0) \in \delta(p, a, Z_0/B)$ if p can reach q in one step,
$([r, r], 0) \in \delta(r, a, Z_0/B)$,
$([r, r], 0) \in \delta([p, r], a, Z_0)$,
$([q, r], 0) \in \delta([p, r], a, Z_0/B)$ if p can reach q in one step,
$([t, t, r], +1)$ and $([t, t, r], 0)$ are both in $\delta([t, r], a, Z_0/B)$,
$([q, t, r], 0) \in \delta([p, t, r], a, Z_0/B)$ if p can reach q in one step,
$([T, r], 0) \in \delta([v, t, r], a, Z_0/B)$ if v is a rejecting configuration,
$([T, r], -1) \in \delta([w, t, r], a, B)$ if w is an accepting configuration,
$([r, r], 0) \in \delta([T, r], a, Z_0)$,

(6) $F = Q$.

The state graph of W is pictorially described in Fig. 4.4, where the label (Z, Δ) on edge (p, q) means that $(q, \Delta) \in \delta(p, a, Z)$. (The label $(Z_0/B, \Delta)$ on edge (p, q) means that $(q, \Delta) \in \delta(p, a, Z_0/B)$.) First, note that since $F = Q$, any strongly fair run in W is an accepting run with respect to i-acceptance for $i = 1, 1', 2$, or $2'$. We now claim that M accepts x iff $L_i^{sf}(W) \neq \emptyset$, $i = 1, 1', 2$, or $2'$. First suppose that there is a strongly fair computation of W. Let Φ denote such a computation. Then from the construction it should be clear that there must exist a reachable universal configuration r of M on x such that W's state is infinitely often $[r, r]$ in Φ. By construction this can only happen if r is reachable from c_0 via a computation having, at the time r occurs, only 2 alternation blocks. Let t be any existential configuration of M on x that is reachable from r. (There must exist at least one such t.) Now W's state in Φ can be $[r, r]$ infinitely often only if it is also $[t, r]$ infinitely often. This means that the transitions from $[t, r]$ to $[t, t, r]$ labelled $(Z_0/B, 0)$ and $(Z_0/B, +1)$ must also occur infinitely often in Φ. However, once the counter becomes nonzero, the only way for W to reenter state $[r, r]$ requires that the counter again be zero. This can only happen if there is an accepting configuration w which is reachable from t. (See Fig. 4.4.) Thus, if W has a strongly fair computation then M accepts x. The converse can be established in a similar fashion. Thus, M accepts x iff W has a strongly fair run. Since the number of possible configurations of M on input x is polynomial in $|x|$, the above construction can be done in deterministic logspace. This completes the proof. □

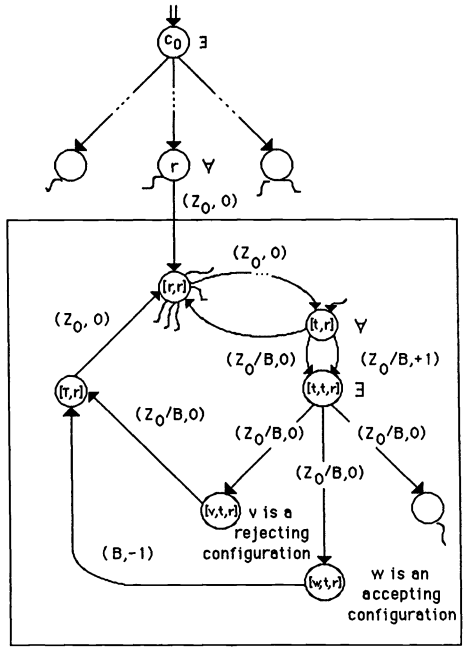


FIG. 4.4. *The ω-1CM W.*

We then have the following theorem.

THEOREM 4.4. *For each $i = 1, 1', 2,$ or $2'$, the strongly fair NEP for ω-1CM's (over 1-letter or multiletter alphabets) with respect to i-acceptance is $\Sigma_3^{L[1]}$-complete.*

4.3. Extensions for other acceptance modes. Other acceptance modes for ω-machines have also been studied in the literature. See e.g. [7], [32]. The emptiness problem for certain classes of ω-machines (where acceptance depends on a fairness constraint) was considered in [32] and was shown to have immediate applications to the verification of concurrent programs. See also [7]. In [7] three modes of acceptance were considered: Buchi acceptance, pairs acceptance and complemented pairs acceptance. (Buchi acceptance and complemented pairs acceptance were also considered in [32]; there however the latter was called Streett acceptance.) Now Buchi acceptance is the same as 2-acceptance and hence has already been defined and partially discussed. The remainder of this subsection is concerned with the (fair, strongly fair) NEP for ω-FSM's and ω-1CM's where acceptance is defined with respect to Buchi, pairs and complemented pairs acceptance.

We now give a definition for pairs and complemented pairs acceptance. Now ω-machines are defined as before, except that F is now a subset of $2^Q \times 2^Q$. Given an input σ, let r be any run on σ in the initial state q_0.

(6) pairs acceptance: σ is *pairs accepted* by r iff there exist $(A, B) \in F$: $\text{Inf}(r) \cap A = \emptyset$ and $\text{Inf}(r) \cap B \neq \emptyset$.

(7) complemented pairs acceptance: σ is *c-pairs accepted* by r iff for all $(A, B) \in F$ $\text{Inf}(r) \cap A \neq \emptyset \Rightarrow \text{Inf}(r) \cap B \neq \emptyset$.

The notions of acceptance and fair acceptance for ω-FSM's and ω-1CM's are then defined as before. For pairs and c-pairs acceptance, similar PTIME-hardness and completeness results hold; i.e., Corollary 4.1, Lemma 4.2 and Theorems 5.1 and 5.2 (of the next section) hold. Also, Theorem 4.4 holds for these two acceptance modes as does the lower bound of Theorem 4.3 (and Corollary 4.2). The proofs for these

results are similar to the ones presented earlier and hence are left to the reader. Theorem 4.1 holds for pairs acceptance but not for c-pairs acceptance. The NEP for c-pairs acceptance is PTIME-complete. (This follows from the fact that the fairness constraints can be enforced (for a given machine M) by adding additional pairs (A, B) to F. Hence, for such machines the NEP and fair NEP are both PTIME-complete.) Another contrasting result is shown as follows.

THEOREM 4.5. (a) *The fair (Buchi acceptance, pairs acceptance, c-pairs acceptance) NEP for ω-FSM's over 1-letter alphabets is* $\Sigma_3^{L[1]}$*-complete.*

(b) *The strongly fair (Buchi acceptance, pairs acceptance, c-pairs acceptance) NEP for ω-FSM's (over 1-letter or multiletter alphabets) is* $\Sigma_3^{L[1]}$*-complete.*

Proof. Here we only prove part (a). The proof for part (b) is identical. We first prove the lower bound. We reduce an arbitrary problem in $A\Sigma_3^L$ to the fair NEP for ω-FSM's over a 1-letter alphabet. Let M be an $O(\log n)$ space bounded 3 alternation bounded ATM. Given an input string x, we show how to construct an ω-FSM M_i (i = Buchi, pairs, c-pairs, respectively) with the property that M accepts x iff M_i fairly accepts some input string (under Buchi, pairs, or c-pairs acceptance, respectively.) The construction here is similar to (but not the same as) the one used in Lemma 4.6. This problem is different because the hardness is caused by the complexity of finding a strongly fair path. In Lemma 4.6, we used a counter to at times preclude the use of certain transitions. Here this is not possible. The reader should now recall the terminology and assumptions regarding ATM's made in Lemma 4.6. They are the same here. Let c_0 be the initial configuration of M on x. Let V (E, U, respectively) be the set of all (existential, universal, respectively) configurations of M on x. We define $M_i = (Q, \Sigma, q_0, \delta, F_i)$, i = Buchi, pairs, or c-pairs, as follows (see Fig. 4.5):

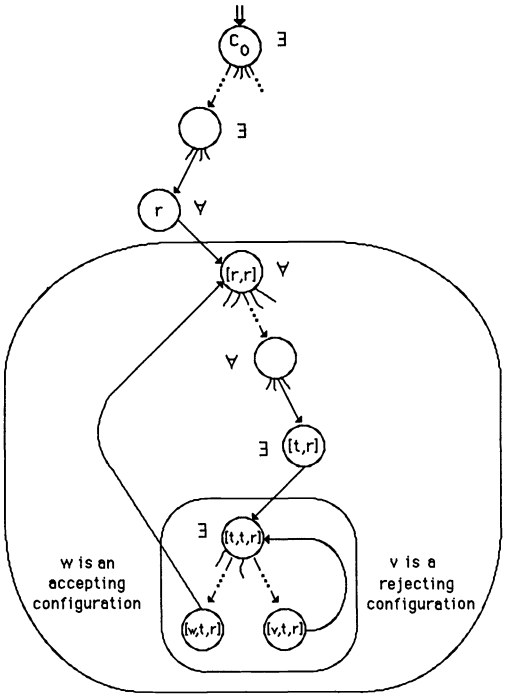


FIG. 4.5. *The ω-FSM M_i.*

(1) $Q = V \cup (V \times V) \cup (V \times V \times V)$,
(2) $\Sigma = \{a\}$,
(3) $q_0 = c_0$,
(4) δ: For each configuration $p, q \in V$, $r \in U$ and $t \in E$, we have:
$q \in \delta(p, a)$ if p can reach q in one step,
$[r, r] \in \delta(r, a)$,
$[q, r] \in \delta([p, r], a)$ if p can reach q in one step,
$[t, t, r] \in \delta([t, r], a)$,
$[q, t, r] \in \delta([p, t, r], a)$ if p can reach q in one step,
$[t, t, r] \in \delta([v, t, r], a)$ if v is a rejecting configuration,
$[r, r] \in \delta([w, t, r], a)$ if w is an accepting configuration,
(5) $F_{\text{Buchi}} = \{\{[r, r]\} \mid r \in U\}$,
$F_{\text{pairs}} = \{(\varnothing, \{[r, r]\}) \mid r \in U\}$,
$F_{c\text{-pairs}} = \{(\{[t, r]\}, \{[w, t, r] \mid w \text{ is an accepting configuration}\}) \mid t \in E, r \in U\}$.

Now consider a fair run on M_i. In each case (i = Buchi, pairs, or c-pairs, respectively) the run corresponds to an accepting computation iff some state $[r, r]$ ($r \in U$) occurs infinitely often in the run. By construction such a run will exist iff there exists r, $r \in U$ is a reachable configuration of M, such that for all t, $t \in E$ and t is reachable from r, t can lead to acceptance. (This works because whenever such an r exists the construction has insured that r is only reachable from c_0 via a computation having, at the time r occurs, only 2 alternation blocks.) Hence M accepts x iff M_i fairly accepts some input string.

We now consider the upper bound. The upper bound for fair Buchi acceptance follows from Lemma 4.5. Here we only show the upper bound for the case of fair c-pairs acceptance. The proof for fair pairs acceptance is similar and is left to the reader. Let $M_{c\text{-pairs}} = (Q, \{a\}, q_0, \delta, F)$ be an ω-FSM where acceptance is by complemented pairs. Let $F = \{(H_{i1}, H_{i2}) \mid 1 \leqq i \leqq l\}$. For convenience, in what follows, we let the term "$u \xrightarrow{*} v$" mean that $u \xrightarrow{*} v$ in M's associated I-graph. We also let the term "$u \xrightarrow{*} H_{ij}$" mean that there exists a state v in H_{ij} such that $u \xrightarrow{*} v$ in M's associated I-graph. Now $M_{c\text{-pairs}}$ fairly accepts a^ω:

$\Leftrightarrow$
$\exists u$, $q_0 \xrightarrow{*} u$ such that
$\forall i, 1 \leqq i \leqq l, \forall v \in Q$ ([$\neg(u \xrightarrow{*} v)$ or $v \xrightarrow{*} u$] and [$\neg(u \xrightarrow{*} H_{i1})$ or ($u \xrightarrow{*} H_{i1}$ and $u \xrightarrow{*} H_{i2}$)])
$\Leftrightarrow$
$\exists u$, $q_0 \xrightarrow{*} u$ such that
$\neg\{\exists i, 1 \leqq i \leqq l, \exists v \in Q$ such that ([$(u \xrightarrow{*} v)$ and $\neg(v \xrightarrow{*} u)$] or [$u \xrightarrow{*} H_{i1}$ and $\neg(u \xrightarrow{*} H_{i2})$])\}.

At this point it should be easy for the reader to see that the condition $\{\cdots\}$ above can be verified by an oracle in $\Sigma_2^{L[1]}$. The upper bound for fair c-pairs acceptance follows. □

A summary of the complexity results broken down by acceptance type, fairness type and machine type is shown in Tables 4.1 and 4.2; where the abbreviation Ψ-C denotes that the problem is complete for the complexity class Ψ.

5. Applications to related problems. This section is devoted to applying our previous results to the Fair Model Checking Problem (FMCP) for Fair Computation Tree Logic (FCTL), which was considered in [7]. In [7], the authors show that the FMCP is, in general, NP-complete, but that the "canonical form" FMCP is in PTIME. However, no lower bound was given for this restricted problem. In what follows, we

TABLE 4.1
The complexity of the fair nonemptiness problem for ω-FSM's.

	fair	*s*-fair	fair(1-letter)
1	PTIME-*C*	$\Sigma_2^{L[1]}$-*C*	$\Sigma_2^{L[1]}$-*C*
1′	PTIME-*C*	$\Sigma_2^{L[1]}$-*C*	$\Sigma_2^{L[1]}$-*C*
2	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	$\Sigma_3^{L[1]}$-*C*
2′	PTIME-*C*	$\Sigma_2^{L[1]}$-*C*	$\Sigma_2^{L[1]}$-*C*
3	*NL-C*	*NL-C*	*NL-C*
pairs	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	$\Sigma_3^{L[1]}$-*C*
c-pairs	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	$\Sigma_3^{L[1]}$-*C*

TABLE 4.2
The complexity of the fair nonemptiness problem for ω-1CM's.

	fair	*s*-fair	fair(1-letter)
1	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*
1′	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*
2	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*
2′	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*
3	*NL-C*	*NL-C*	*NL-C*
pairs	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*
c-pairs	PTIME-*C*	$\Sigma_3^{L[1]}$-*C*	PTIME-*C*

show that the canonical form FMCP is PTIME-hard, and thus PTIME-complete. As a result, we show that the problems considered in § 4 are all in PTIME. Before proceeding, some preliminaries are required. The following definitions are almost literally taken from [7]. Additional details can be found in [7].

In [7] they inductively define state formulae (true or false of states) and path formulae (true or false of paths) as follows:

(S1) Any atomic proposition P is a state formula.

(S2) If p and q are state formulae then $p \wedge q$ and $\neg p$ are state formulae.

(S3) If p is a path formula then Ep is a state formula.

(P1) Any state formula p is a path formula.

(P2) If p and q are path formulae then $p \wedge q$ and $\neg p$ are path formulae.

(P3) If p and q are path formulae then Xp and pUq are path formulae.

The following connectives are then introduced via the abbreviations: $p \vee q \equiv \neg(\neg p \wedge \neg q)$, $Ap \equiv \neg E \neg p$, $Fp \equiv trueUp$, $Gp \equiv \neg F \neg p$, $F^{\infty}p \equiv GFp$, and $G^{\infty}p \equiv \neg F^{\infty} \neg p$.

The intuitive meanings of the formulae are as follows: $p \wedge q$ indicates the conjunction of p and q, $p \vee q$ indicates the disjunction of p with q, $\neg p$ indicates the negation of p, Ep means for some infinite path p holds, Ap means for all fullpaths p holds, Xp indicates next time p, pUq means q eventually holds and p holds continuously until then, Fp means p holds at some future time, Gp means that p always holds, $F^{\infty}p$ means that p is true infinitely often, and $G^{\infty}p$ means that p is true almost everywhere, i.e., at all but a finite number of times.

The semantics of a formula is defined with respect to a structure. A structure $M = (S, R, L)$ is a labelled transition graph where S is a nonempty finite set of states, R ($\subseteq S \times S$) is a total binary relation representing the possible transitions between states, and L is a labelling which assigns each state to a set of atomic propositions

(i.e. those propositions which are true at that state). An *infinite path* is an infinite sequence of states $x_0, x_1, \cdots, x_i, \cdots$, such that $x_i \in S$ and $(x_i, x_{i+1}) \in R$, $\forall i \geqq 0$. One writes $M, s \models p$ ($M, x \models p$) to mean that the state formula p (path formula p) is true in structure M at state s (of path x, respectively). For a structure M, a state s and an infinite path $x = x_0, x_1, \cdots$, the $\models$ relation is defined inductively as follows (let $x^i = x_i, x_{i+1} \cdots$):

(1) $M, s \models P$ if $P \in L(s)$ for any atomic proposition P.
(2) $M, s \models \neg p$ iff not $(M, s \models p)$.
(3) $M, s \models p \wedge q$ iff $M, s \models p$ and $M, s \models q$.
(4) $M, s \models Ep$ iff $\exists$ an infinite path x starting at s such that $M, x \models p$.
(5) $M, x \models p$ iff $M, x_0 \models p$ for any state formula p.
(6) $M, x \models \neg p$ iff not $(M, x \models p)$.
(7) $M, x \models p \wedge q$ iff $M, x \models p$ and $M, x \models q$.
(8) $M, x \models Xp$ iff $M, x^1 \models p$.
(9) $M, x \models pUq$ iff $\exists i \geqq 0$, $M, x^i \models q$ and $\forall j \geqq 0$ [$j \leqq i$ implies $M, x^j \models p$].

From the above recursive definitions we see that $M, x \models F^\infty p$ iff there exists infinitely many $i \geqq 0$ such that $M, x^i \models p$ and $M, x \models G^\infty p$ iff $\exists i \geqq 0$ ($\forall j \geqq i$, $M, x^j \models p$).

Given a structure M, an FCTL specification (p_0, Φ_0) consists of a state formula p_0 and a path formula Φ_0 (the fairness constraint). p_0 is of the form either A_Φ ("for all fair paths") or E_Φ ("for some fair path") followed one of the operators F, G, X, or U. The path quantifiers are subscripted by Φ to emphasize that they range over paths meeting the fairness constraint Φ_0. Φ_0 is constructed from the propositions $F^\infty p$ ("infinitely often p") and $G^\infty p$ ("almost always p") using the standard boolean operators $\neg$, $\vee$ and $\wedge$. (Here p denotes any propositional argument.) See [7] for more details. Now, the *Model Checking Problem for FCTL* (FMCP) is: Given a structure $M = (S, R, L)$, a FCTL specification (p_0, Φ_0) and a designated state $s \in S$, to determine whether $M, s \models p_0$ when considering only paths that satisfy the fairness assumption Φ_0—paths x such that $M, x \models \Phi_0$. See [7] for a formal statement of the problem.

In [7], they show that when Φ_0 is in the canonical form $\bigvee_{j=1}^{n} \bigwedge_{i=1}^{k_j} (G^\infty p_{ij} \vee F^\infty q_{ij})$, the FMCP is in PTIME. In what follows, we show that the fair NEP for ω-FSM's under 1, 1′, 2, 2′, pairs or c-pairs acceptance is deterministic logspace reducible to the canonical form FMCP. Since the former problem is known to be PTIME-hard, the canonical form FMCP is PTIME-hard.

LEMMA 5.1. *The fair NEP for ω-FSM's, under fairness types* 1, 1′, 2, 2′, *pairs, or c-pairs, can be reduced to the canonical form FMCP by a deterministic logspace reduction.*

Proof. The basic idea of the proof is that given an ω-FSM W, we construct a structure $M = (S, R, L)$ with a designated state s and a specification (p_0, Φ_0) (of the indicated canonical form) in deterministic logspace such that, W fairly accepts some input iff the FMCP has a solution with respect to M, s and (p_0, Φ_0).

Let $W = \{Q, \Sigma, q_0, \delta, \{H_1, H_2, \cdots, H_t\}\}$. (Without loss of generality we can assume that $\Sigma = \{a, b\}$. See Lemma 4.1.) For pairs and c-pairs acceptance modes, $H_i = (C_i, D_i)$, $1 \leqq i \leqq t$. Let $Q = \{q_0, q_1, \cdots, q_r\}$. We define $S = Q \cup \{a_{i,j} \mid q_j \in \delta(q_i, a)\} \cup \{b_{i,j} \mid q_j \in \delta(q_i, b)\} \cup \{q'\}$. The relation R of M is constructed from the graph of W by replacing each edge (q_i, q_j) labelled read(a) (read(b)) by two edges $(q_i, a_{i,j})$ and $(a_{i,j}, q_j)$ ($(q_i, b_{i,j})$ and $(b_{i,j}, q_j)$). In other words, the state $a_{i,j}$ ($b_{i,j}$) is inserted between q_i and q_j, if $q_i \to q_j$. See Fig. 5.1. To insure that R is total we add to R, the edge (s, q') for every $s \in S$. We define L as follows:

(1) For $0 \leqq i \leqq r$, Q_i is true only at state q_i.
(2) (a) (For 1, 1′, 2, and 2′ acceptance only.) For $1 \leqq i \leqq t$, $\mathcal{H}_i$ is true at a state $s \in S$ iff $s \in H_i$.

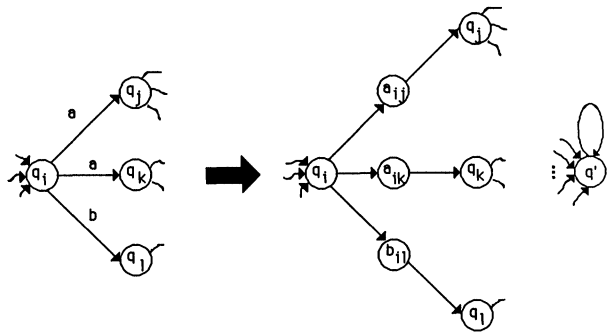


FIG. 5.1. *The construction of M from W.*

(b) (For pairs and c-pairs acceptance only.) For $1 \leqq i \leqq t$, $\mathscr{C}_i(\mathscr{D}_i)$ is true at a state $s \in S$ iff $s \in C_i$ ($s \in D_i$).

(3) For $0 \leqq i, j \leqq r$, $A_{i,j}$ ($B_{i,j}$) is true only at $a_{i,j}$ ($b_{i,j}$). (Note $a_{i,j}$ ($b_{i,j}$) need not exist. If not then $A_{i,j}$ ($B_{i,j}$) is false at every state.)

(4) Q' is true only at q'.

Now we define a portion of the fairness constraint to be the conjunct of the following:

(1) $\bigwedge_{i=0}^{r} G^{\infty}(\neg Q_i) \vee F^{\infty}((\bigvee_{j=0}^{r} A_{i,j}) \vee (\bigvee_{j=0}^{r} B_{i,j}))$
* For each q_i, either q_i is visited only finitely often or one of its outgoing edges must be traversed infinitely often. *\

(2) $\bigwedge_{0 \leqq i,j \leqq r} G^{\infty}(\neg A_{i,j}) \vee F^{\infty}\left(\bigwedge_{\text{all } k \neq j \text{ s.t., } q_k \in \delta(q_i, a)} A_{i,k}\right)$
* For each state, if one of its outgoing edges labelled "read(a)" occurs infinitely often, then all of its outgoing edges with the same label must occur infinitely often. *\

(3) Replace A (a) with B (b) in (2) above.

(4) $G^{\infty}(\neg Q')$
* insures that no edge (s, q') is used *\

Let $\Phi' = (1) \wedge (2) \wedge (3) \wedge (4)$. Let the designated state be q_0 (i.e. the initial state of W). Now the specification (p_0, Φ_0) for each acceptance mode is as follows:

(1) $p_0 = E_{\Phi} F(\bigvee_{i=1}^{t} \mathscr{H}_i)$ and $\Phi_0 = \Phi'$.

(1′) $p_0 = \bigvee_{i=1}^{t} E_{\Phi} G \mathscr{H}_i$ and $\Phi_0 = \Phi'$.

(2) $p_0 = E_{\Phi}$ **true** and $\Phi_0 = \Phi' \wedge F^{\infty}(\bigvee_{i=1}^{t} \mathscr{H}_i)$.

(2′) $p_0 = E_{\Phi}$ **true** and $\Phi_0 = \bigvee_{i=1}^{t} (\Phi' \wedge G^{\infty} \mathscr{H}_i)$.

(pairs) $p_0 = E_{\Phi}$ **true** and $\Phi_0 = \bigvee_{i=1}^{t} (\Phi' \wedge G^{\infty} \neg \mathscr{C}_i \wedge F^{\infty} \mathscr{D}_i)$.

(c-pairs) $p_0 = E_{\Phi}$ **true** and $\Phi_0 = \Phi' \wedge [\bigwedge_{i=1}^{t} (G^{\infty} \neg \mathscr{C}_i \vee F^{\infty} \mathscr{D}_i)]$.

It should be clear that W fairly accepts something iff the FMCP with respect to M, q_0 and (the proper) (p_0, Φ_0) has a solution. This completes the proof. □

With this lemma, we immediately have the following.

COROLLARY 5.1. *For $i = 1, 1', 2, 2'$, pairs, or c-pairs, the fair NEP for ω-FSM's (over multiletter alphabets) with respect to i-acceptance is* PTIME-*complete.*

This completes the proof of Corollary 4.1. As a result we also have the following.

THEOREM 5.1. *The canonical form FMCP for FCTL is* PTIME-*complete.*

In what follows, we generalize the result in Lemma 5.1 for ω-1CM's. Let P be the polynomial mentioned in § 4.2. Let W be an ω-1CM. Let $(p, h) \vdash (q, h')$ ($(p, h) \vdash^{t} (q, h')$) denote that in state p with counter value h, W can, in a single step, reach state q with counter value h' (via transition t). Recall from § 4.2 that for every fair run r of W there exists a finite path r': $(q_0, 0) \vdash \cdots \vdash (q, h) \vdash \cdots \vdash (q, h+d)$

such that the set of transitions used in $(q, h) \vdash \cdots \vdash (q, h+d)$ coincide with the set of transitions that occur infinitely often in r. Furthermore, r' satisfies:

(1) $0 \leqq h, d \leqq P(|W|)$,

(2) the length of the path is no more than $2^*P(|W|)$, and

(3) either $h=0$ and $d=0$, or $d>0$ and no move in the path depends on the counter being zero.

Let t_2 be the sequence of transitions taken in the path $(q, h) \vdash \cdots \vdash (q, h+d)$. Then r' can easily be extended into a fair path by applying the sequence of transitions $(t_2)^\omega$ to (q, h). The significance of this fact is that we can capture the notion of W's infinite computations using only a finite state model. Now given an ω-1CM W (over any of the previously mentioned acceptance modes) we can construct in polynomial time a labelled directed graph G_W that, in some sense, models the computation of W. The vertices of G_W are pairs (q, h), where q is a state of W and $0 \leqq h \leqq 2^*P(|W|)$. If there is a transition t of W that takes (p, h) into (q, h') then the edge $(p, h) \to (q, h')$ labelled t is in G_W. Also if $(q, h) \vdash \cdots \vdash (q, h+d)$ (in W) as above, then G_W has the unlabelled edge $(q, h+d) \to (q, h)$. (Note, that adding these latter edges is what "precludes" G_W from being constructed in deterministic logspace.) It is easy to see that paths in G_W correspond in a natural way to computations of W. Now from the pair (W, G_W) one can construct, in deterministic logspace, an instance of FMCP that has a solution iff W fairly accepts some input. The proof of this claim is similar to the proof of Lemma 5.1, although perhaps somewhat more involved. The formal details are left to the reader. From this claim, Lemma 4.2 and Theorem 5.1 we obtain:

THEOREM 5.2. *For each $i = 1, 1', 2, 2'$, pairs and c-pairs, the fair NEP for ω-1CM's (over 1-letter or multiletter alphabets) with respect to i-acceptance is* PTIME-*complete.*

6. Conclusion. We have provided an alternate characterization of the alternating logspace hierarchy in terms of oracle Turing machines. Using this characterization we have given a thorough analysis of the fair and strongly fair nonemptiness problem for seven types of ω-FSM's (and seven types of ω-1CM's). The problems were shown to be complete for either NL, $A\Sigma_2^L$, $A\Sigma_3^L$, or PTIME. Hence, we have presented what seems to be the first natural examples of problems "between" NL and PTIME. Some of these problems have been shown to have immediate applications to problems concerned with the verification of concurrent finite state programs. See [32]. Some additional complexity results related to this topic can be found in [27]. Also, extensions of this work can be shown for certain liveness problems posed in [3] concerning networks of communicating finite state machines. (See also [10].) See [26] for a summary.

Acknowledgment. We thank E. Emerson, C. Lei and W. Ruzzo for their comments concerning an early version of this work. We also thank the anonymous referees for numerous comments which greatly improved the presentation of these results.

REFERENCES

[1] J. BENTLEY, T. OTTMANN AND P. WIDMAYER, *The complexity of manipulating hierarchically defined sets of rectangles*, in Advances in Computing Research 1, 1983, JAI Press, pp. 127-158.

[2] A. CHANDRA, D. KOZEN AND L. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[3] C. CHANG, M. GOUDA AND L. ROSIER, *Deciding liveness for special classes of communicating finite state machines*, Proc. 22nd Annual Allerton Conf. on Communication, Control, and Computing (1984), pp. 931-939.

[4] R. COHEN AND A. GOLD, *Theory of ω-languages. I: Characterizations of ω-context-free languages*, J. Comput. System Sci., 15 (1977), pp. 169-184.

[5] S. COOK, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing (1971), pp. 151-158.

[6] ———, *The classification of problems which have fast parallel algorithms*, in Fundamentals of Computation Theory, LNCS, 158 (1983), pp. 78-93.

[7] E. EMERSON AND C. LEI, *Modalities for model checking: branching time logic strikes back*, Sci. Comput. Programming 8 (1986), (some of these results were presented at the 18th Annual Hawaii International Conference on System Sciences and at the 12th Annual ACM Symposium on Principles of Programming Languages).

[8] M. FISCHER AND M. PATERSON, *Storage requirements for fair scheduling*, Inform. Process. Lett., 17 (1983), pp. 249-250.

[9] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[10] M. GOUDA AND C. CHANG, *A technique for proving liveness of communicating finite state machines with examples*, Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing (1984), pp. 38-49.

[11] M. GOUDA AND L. ROSIER, *On deciding progress for a class of communication protocols*, Proc. 18th Annual Conf. on Information Sciences and Systems (1984), pp. 663-667.

[12] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[13] O. IBARRA AND L. ROSIER, *On restricted one-counter machines*, Math. Systems Theory, 14 (1981), pp. 241-245.

[14] N. JONES AND W. LAASER, *Complete problems for deterministic polynomial time*, Theoret. Comput. Sci., 3 (1977), pp. 105-117.

[15] N. JONES, E. LIEN AND W. LAASER, *New problems complete for nondeterministic log space*, Math. Systems Theory, 10 (1976), pp. 1-17.

[16] R. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85-104.

[17] R. LADNER AND N. LYNCH, *Relativization of questions about log space computability*, Math. Systems Theory, 10 (1976), pp. 19-32.

[18] L. LANDWEBER, *Decision Problems for ω-automata*, Math. Systems Theory, 3 (1969), pp. 376-384.

[19] D. LEHMANN, A. PNUELI AND J. STAVI, *Impartiality, justice and fairness: The ethics of concurrent termination*, in Automata, Languages and Programming, Lecture Notes in Computer Science, 115 (1981), pp. 264-277.

[20] H. LEWIS AND C. PAPADIMITRIOU, *Symmetric space-bounded computation*, in Automata, Languages and Programming, Lecture Notes in Computer Science, 85, 1980, pp. 374-384.

[21] O. LICHTENSTEIN AND A. PNUELI, *Checking that finite state concurrent programs satisfy their linear specification*, Proc. 12th Annual ACM Symp. on Principles of Programming Languages (1985), pp. 97-107.

[22] N. LYNCH, *Log space machines with multiple oracle tapes*, Theoret. Comput. Sci., 6 (1978), pp. 25-39.

[23] C. PAPADIMITRIOU, *On the complexity of unique solutions*, J. Assoc. Comput. Mach., 31 (1984), pp. 392-400.

[24] C. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of facets (and some facets of complexity)*, J. Comput. System Sci., 28 (1984), pp. 244-259.

[25] J. REIF, *Symmetric complementation*, J. Assoc. Comput. Mach., 31 (1984), pp. 401-421.

[26] L. ROSIER AND H. YEN, *Logspace hierarchies, polynomial time and the complexity of fairness problems concerning ω-machines*, Rep. TR-85-08, Department of Computer Sciences, University of Texas at Austin, Austin, TX, 1985.

[27] ———, *On the complexity of deciding fair termination of probabilistic concurrent finite-state programs*, in Automata, Languages and Programming, Lecture Notes in Computer Science, 226, 1986, pp. 334-343.

[28] W. RUZZO, J. SIMON AND M. TOMPA, *Space-bounded hierarchies and probabilistic computations*, J. Comput. System Sci., 28 (1984), pp. 216-230.

[29] W. SAVITCH, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177-192.

[30] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1-22.

[31] L. VALIANT AND M. PATTERSON, *Deterministic one-counter automata*, J. Comput. System Sci., 10 (1975), pp. 340-350.

[32] M. VARDI, *Automatic verification of probabilistic concurrent finite-state programs*, Proc. 26th Annual Symposium on Foundations of Computer Science (1985), pp. 327-338.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987

© 1987 Society for Industrial and Applied Mathematics

DETERMINISTIC SIMULATION OF IDEALIZED PARALLEL COMPUTERS ON MORE REALISTIC ONES*

HELMUT ALT†, TORBEN HAGERUP‡, KURT MEHLHORN‡ AND FRANCO P. PREPARATA§

Abstract. We describe a nonuniform deterministic simulation of PRAMs on module parallel computers (MPCs) and on processor networks of bounded degree. The simulating machines have the same number n of processors as the simulated PRAM, and if the size of the PRAM's shared memory is polynomial in n, each PRAM step is simulated by $O(\log n)$ MPC steps or by $O((\log n)^2)$ steps of the bounded-degree network. This improves upon a previous result by Upfal and Wigderson. We also prove an $\Omega((\log n)^2/\log\log n)$ lower bound on the number of steps needed to simulate one PRAM step on a bounded-degree network under the assumption that the communication in the network is point to point.

As an important part of the simulation of PRAMs on MPCs, we use a new technique for dynamically averaging out a given work load among a set of processors operating in parallel.

Key words. parallel RAM, module parallel computer, processor network, deterministic simulation

AMS(MOS) subject classification. 68C25

1. Introduction and models of computation. Most parallel algorithms in the literature are designed to run on a PRAM (parallel RAM). The PRAM model was introduced by Fortune and Wyllie [FW]. A PRAM consists of some finite number n of sequential processors (RAMs), all of which operate synchronously on a common memory consisting of, say, m storage cells (also called "variables"), cf. Fig. 1. In every step of the PRAM, each of its processors executes a private RAM instruction. In particular, the processors may all simultaneously access (read from or write into) the common memory. Various types of PRAMs have been defined, differing in the conventions used to deal with read/write conflicts, i.e., attempts by several processors to access the same variable in the same step. In the most restrictive model, exclusive read-exclusive write or EREW PRAMs, no variable may be accessed by more than one processor in a given step. In contrast, CRCW (concurrent read-concurrent write) PRAMs allow simultaneous reading as well as simultaneous writing of each variable, with some rule defining the exact semantics of simultaneous writing.

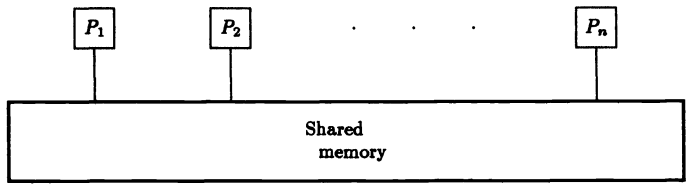


FIG. 1. *The PRAM model.* $P_1, \cdots, P_n$ *are processors.*

* Received by the editors April 28, 1986; accepted for publication (in revised form) November 14, 1986. This work was supported by the Deutsche Forschungegemeinschaft, SFB 124, TP B2, VLSI Entwurf und Parallelität, and by National Science Foundation grant ECS-84-10902. A preliminary and abridged version of this paper was presented at the 12th International Symposium on Mathematical Foundations of Computer Science, Bratislava, Czechoslovakia in August 1986.

† Fachbereich Mathematik, Freie Universität Berlin, Arnimallee 2-6, D-1000 Berlin 33, Federal Republic of Germany. Part of the research was done while the author was a member of the Mathematical Sciences Research Institute, Berkeley, California 94720.

‡ Fachbereich 10, Informatik, Universität des Saarlandes, D-6600 Saarbrücken, Federal Republic of Germany.

§ Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801.

PRAMs are very convenient for expressing parallel algorithms since one may concentrate on the problem of "parallelizing," i.e., decomposing the problem at hand into simultaneously executable tasks, without having to worry about the communication between these tasks. Indeed, any intermediate result computed by one of the processors will be available to all the others in the next step, due to the shared memory. Unfortunately, the PRAM is not a very realistic model of parallel computation. Present and foreseeable technology does not seem to make it possible for more than a constant number of processors to simultaneously access the same memory module. A model of computation that takes this problem into account is the MPC (*module parallel computer*, [MV]), cf. Fig. 2. An MPC consists of n processors (RAMs), each equipped with a memory module. Every processor may access every memory module via a complete network connecting the processors. However, the memory modules are sequential devices, i.e., able to satisfy only one request at a time. More precisely, a memory module M operates as follows: If several processors try in the same step to access a variable stored in M, exactly one of the processors is allowed to carry out its read or write instruction; the remaining access requests are discarded. All processors are informed of the success or failure of their access attempts. We make no assumptions about how the single successful processor is selected from among the processors competing to access M.

The MPC model is still not realistic for large n because of the postulated complete network connecting the processors. This leads us to consider a third model which we shall call the *network model*. Here the processors are connected via a network of bounded degree, i.e., each processor is linked directly to only a constant number of other processors, cf. Fig. 3. Since each step of a completely interconnected processor network may be simulated by $O(\log n)$ steps of a bounded-degree network ([AKS], [L]), efficient algorithms for the MPC model translate into asymptotically efficient algorithms for the network model.

The simulation of the idealized parallel machine, the PRAM, on the more realistic one, the MPC, has been considered in several previous papers. A naive approach represents each variable x of the PRAM by one variable $\psi(x)$ of the MPC. Now if a PRAM step accesses the variables $x_1, \cdots, x_l$, collisions may occur in the simulating

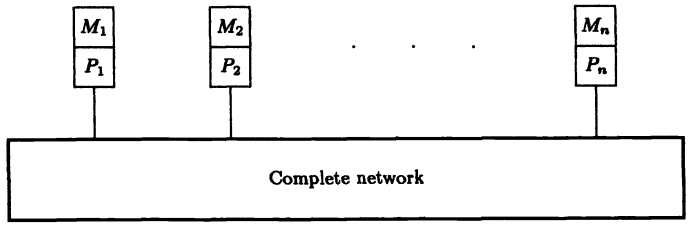


FIG. 2. *The MPC model.* $P_1, \cdots, P_n$ *are processors,* $M_1, \cdots, M_n$ *memory modules.*

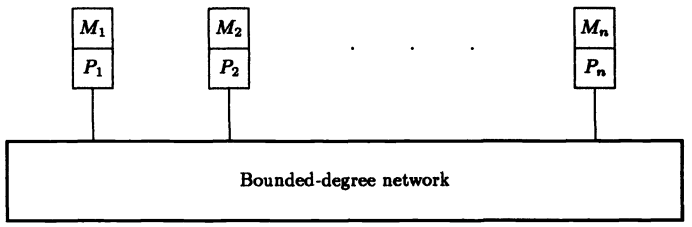


FIG. 3. *The network model.* $P_1, \cdots, P_n$ *are processors,* $M_1, \cdots, M_n$ *memory modules.*

machine because $\psi(x_1), \cdots, \psi(x_l)$ are not necessarily located in distinct memory modules. If $m \leqq n$, the m variables may be allocated to m different memory modules, and a trivial $O(1)$-time simulation is possible. However, we are concerned with the case in which m is considerably larger than n. Here a major problem is to find a memory correspondence between the PRAM and the MPC such that, for all possible access patterns of the PRAM, the maximum number of variables requested from a single MPC memory module is kept low. Note that, for specific PRAM algorithms such as matrix multiplication, there may be very efficient ways of assigning variables to modules; we refer the reader to § 4 of the survey paper by Kuck [K]. Here we are interested in universal simulations that work efficiently no matter which algorithm is executed by the PRAM.

Some results have been obtained previously using *probabilistic* methods: Mehlhorn and Vishkin [MV] used universal hashing to define the memory correspondence. They obtained several upper bounds, for example, an average of $O(\log n)$ MPC steps to simulate one PRAM step, with the total amount of memory used by the MPC larger than the PRAM memory by a factor of $O(\log n)$. Upfal [U] found a probabilistic simulation of $O((\log n)^2)$ average time for one PRAM step on a bounded-degree network; this was recently improved to $O(\log n)$ by Karlin and Upfal [KU].

This paper is concerned with *deterministic* simulations. We define the *slowdown* of a simulation as the number of steps needed by the simulating machine in the worst case to simulate one step of the simulated machine. Note that if $m \geqq n^2$, the simple scheme outlined above (x is represented by $\psi(x)$) performs poorly: An adversary could make the PRAM step access n variables $x_1, \cdots, x_n$ with $\psi(x_1), \cdots, \psi(x_n)$ all in the same module. Hence the slowdown is $\Omega(n)$. This reasoning shows that each PRAM variable must be represented by several "copies" stored in different modules. Mehlhorn and Vishkin [MV] showed that read instructions can be handled very efficiently using this idea. However, they did not know how to deal with write instructions. In a beautiful paper Upfal and Wigderson [UW] resolved this problem and exhibited a simulation which uses $\Theta(\log n)$ copies of each PRAM variable. If m is polynomial in n, the slowdown is $O(\log n(\log \log n)^2)$. They also showed an $\Omega(\log n/\log \log n)$ lower bound on the slowdown for a large class of simulations.

Using similar techniques, this paper improves the upper bound to $O(\log m)$. If m is polynomial in n, this is $O(\log n)$. Consequently, a PRAM step may be simulated in $O(\log n \log m)$ time on a bounded-degree network. On the other hand, we show that $\Omega(\log n \log m/\log \log m)$ time is necessary under certain assumptions on any bounded-degree network whose communication is restricted to be point to point. A similar result was also obtained by Karlin and Upfal [KU]. The assumption of point to point communication is not satisfied by our simulation algorithm which uses more general communication patterns.

The PRAM simulations which we consider will be based on *emulations* of the PRAM's shared memory. We conceptually retain the n PRAM processors while replacing (or, equivalently, implementing) the PRAM's (physically infeasible) shared memory by a (more feasible) suitably programmed MPC or bounded-degree network with n processors, called the emulating processors. Each PRAM processor, which was formerly connected to the shared memory, is now instead connected to one of the emulating processors called its *associated* processor, each emulating processor being associated with exactly one PRAM processor, cf. Fig. 4. We require the change to be completely transparent, i.e., all PRAM programs must run without change (though possibly slower) on the modified machine. Note that although the most direct PRAM simulation implied by a memory emulation as above uses a total of $2n$ processors, it

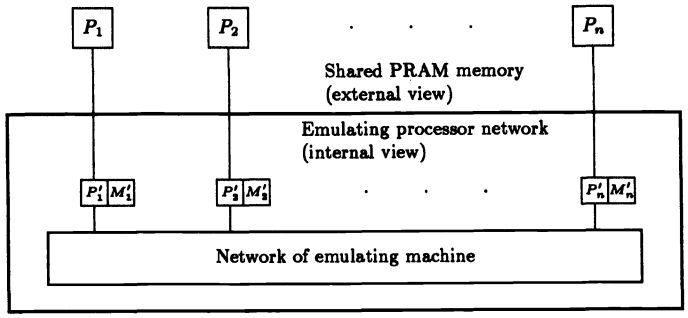


FIG. 4. *Emulation of the shared memory of a PRAM. For* $i = 1, \cdots, n$, P_i *is a PRAM processor and* P'_i *its associated emulating processor.*

is a trivial matter to reduce the number of processors to n by coalescing each pair of associated processors into a single processor. For expository reasons we prefer to keep the clean separation between PRAM processors and (emulated) shared memory.

Our simulation algorithms are nonuniform. This means that they are not given explicitly. Instead we merely prove that algorithms with the desired properties exist. For fixed values of n and m, such algorithms may be found by exhaustive search in a large but finite set. We return to this aspect in the concluding section.

It has been known since Adleman's work [A] that probabilistic algorithms may be converted into nonuniform deterministic ones. Hence the result by Karlin and Upfal [KU] automatically translates into a nonuniform deterministic simulation of PRAMs on a bounded-degree network. However, if the translation is based on Karlin and Upfal's analysis of their algorithm and uses known techniques, it introduces an $\Omega(n)$-increase in the product of time and number of processors [R, Thm. 6]. Since it is not difficult to devise a uniform deterministic simulation which uses $O(n^2/(\log n)^2)$ processors and has a slowdown of $O(\log n)$ (the construction is similar to one presented in the remark ending § 3), deterministic algorithms derived from Karlin and Upfal's probabilistic simulation are of little interest. The same is true of all other known probabilistic solutions.

The remaining part of the paper is structured as follows: In § 2 we describe our simulation of PRAMs on MPCs and show that its slowdown is $O(\log m)$. As part of the development of the algorithm, we define and solve a so-called "redistribution problem." Section 3 considers the simulation of PRAMs on bounded-degree networks and establishes upper and lower bounds of $O(\log n \log m)$ and $\Omega(\log n \log m/\log\log m)$, respectively. In § 4 we return to the redistribution problem and prove a stronger result than what was needed in § 2. Finally, § 5 addresses some interesting and important open issues.

2. Simulation of PRAMs on MPCs. In this section we describe an emulation with $O(\log m)$ slowdown of the shared memory of an EREW PRAM with n processors and m shared memory cells by an MPC with n processors. A standard method allows the emulation to be extended using only $O(\log n)$ extra time per simulated step to cover all common variants of CRCW PRAMs. We shall not give the details of this but refer the reader to [BH].

We will assume that $n < m \leqq 2^n$. This is no restriction: If $m \leqq n$, an $O(1)$-time solution is possible as argued in the previous section. And if $m > 2^n$, even the trivial sequential simulation on just one MPC processor is within the time bound of $O(\log m)$. We further allow our construction to fail for values of n and m smaller than some

(unspecified) constant number N. Again this is no restriction. Note that whenever we employ the "big-oh" notation, the limiting process implied is that of n and m tending to infinity while other parameters may take on arbitrary legal values.

For concreteness, let us assume that each PRAM processor P has two instructions for communicating with the shared memory: LOAD x, where x is a shared variable, replaces the contents of a distinguished register in P, called its accumulator, by the current value of x (which is held in the shared memory), and STORE x updates x with the contents of P's accumulator. We also assume that each PRAM variable x has a unique identification which we shall call its *name* and denote symbolically by "x".

We can now describe the computational task that must be performed by the emulating machine. In the beginning of each PRAM step, each emulating processor P' receives from its associated PRAM processor P as input one of the following:

(1) The name "x" of a PRAM variable x, meaning that P's current instruction is LOAD x. In this case we will say that P' *reads* x in the current (PRAM) step.

(2) A pair $\langle$"x", $a\rangle$, where "x" is the name of a PRAM variable x, meaning that P's current instruction is STORE x, and a is the contents of P's accumulator. We say that P' *writes* or *updates* x in the current step.

(3) A signal *NoOp*, meaning that P does not access the shared memory in the current step.

In cases (1) and (2), we will also say that P' is the *origin* of x and that x is *accessed* in the current PRAM step.

At the end of the simulation of each PRAM step, each emulating processor reading a variable x in that step must have computed the current value of x and must output it to the associated PRAM processor. The current value of x is defined in the obvious way as the second component of the pair $\langle$"x", $a\rangle$ with first component "x" most recently input by some emulating processor in a PRAM step prior to the current step (since we are considering only correct EREW PRAM programs, the pair $\langle$"x", $a\rangle$ is well-defined).

Our emulation algorithm is based on an idea introduced by Upfal and Wigderson [UW]:

Each variable x of the PRAM is represented by $2c-1$ memory cells $\psi_1(x), \cdots, \psi_{2c-1}(x)$ of the emulating MPC, where $c = \lfloor \log m \rfloor$.[1] These cells will be called the *copies* of x.

Whenever a processor of the PRAM executes the instruction STORE x, the MPC will access a majority, i.e., at least c, of x's copies and store the new value into them together with a time stamp indicating the number of PRAM steps simulated so far.

The simulation of the instruction LOAD x also consists of accessing at least c copies of x. Since this is again a majority, at least one of the copies must contain the current value of x, and the most recently updated copy is readily identified using the time stamps.

Remark. It suffices to count time modulo m if each PRAM variable is "cleaned" once every m simulated steps. Cleaning a variable means inspecting all its copies and replacing each of their time stamps by the appropriate one of two special values "invalid" and "oldest". Since this takes $O(c) = O(\log m)$ sequential time, one variable may be cleaned after each simulated step at only a constant-factor increase in simulation time. Furthermore, if variables are cleaned in a fixed cyclical order known to all MPC processors, the modified time stamps allow the MPC to determine the most recently

[1] "log" without subscript denotes the logarithm to base 2.

updated copy much as before. In the light of this observation, we shall assume that time stamps can be manipulated in unit time.

We partition the processors of the MPC into $k = \lceil n/(2c-1) \rceil$ *clusters* containing $2c-1$ processors each. (We assume that $2c-1$ divides n perfectly. If this is not the case, there will be an incomplete cluster. But if we let each MPC processor simulate two "virtual processors" at the price of increasing the execution time by at most a constant factor, there will be more than enough processors to fill the last cluster. Hence our assumption implies no loss of generality.) For $i = 1, \cdots, k$ and $j = 1, \cdots, 2c-1$, denote by $P_{i,j}$ the jth processor in the ith cluster, and for $r = 1, \cdots, 2c-1$, let $\mathcal{P}(r)$ be the set of the first r processors from each cluster, i.e., $\mathcal{P}(r) = \{P_{i,j} \mid 1 \leqq i \leqq k, 1 \leqq j \leqq r\}$. All processors within a cluster cooperate when an access to a variable is attempted by trying to access its different copies.

From now on consider the simulation of a single PRAM step called the *current step*. For each variable x accessed in the current step, there will be an MPC processor P *responsible* for x. Conversely, we shall say that x is in P's *custody* and sometimes think of x as residing in P. Initially, x is in the custody of its origin.

Responsibilities are changed during calls of the procedure REDISTRIBUTE to be described later. We maintain the invariant, however, that each processor at any given time is responsible for at most one variable.

Following is a description of the major subroutines used by our algorithm. In each case, all clusters operate in parallel.

ACCESS(j): ($1 \leqq j \leqq 2c-1$). Within the ith cluster, for $i = 1, \cdots, k$, all processors simultaneously try to access the different copies of the variable x_i which is in $P_{i,j}$'s custody. More specifically: For $l = 1, \cdots, 2c-1$, if $P_{i,l}$ has not previously been successful in an access to $\psi_l(x_i)$, it tries to access $\psi_l(x_i)$. If $P_{i,j}$ is currently not responsible for any variable, or if $P_{i,l}$ has already succeeded in accessing $\psi_l(x_i)$, $P_{i,l}$ remains idle. We shall say that the variables which are in the custody of some processor $P_{i,j}$ ($i = 1, \cdots, k$) are in the *range* of the call ACCESS(j).

Since processors from different clusters may attempt to access memory cells within the same module at the same time, in which case only one of them can be served, there will be successful and unsuccessful attempts. Each processor which is successful in accessing its copy records this fact in an internal table. If the copy is of a variable x which is read in the current step, the processor also records the value and the time stamp of the copy; if x is written in the current step, the processor updates the copy with the new value and time stamp.

By assumption the execution time of ACCESS is $O(1)$.

COUNT(r): ($1 \leqq r \leqq 2c-1$). For each variable x which is in the custody of some processor $P_{i,j}$ in $\mathcal{P}(r)$, this procedure counts the number of successful attempts by processors in the ith cluster to access copies of x. The variables for which this number is $\geqq c$ are declared *dead*, meaning that it is not necessary to access any more of their copies. COUNT uses for each cluster C a virtual complete binary tree T_C of processors, the processors of C being its leaves. Since T_C contains less than twice as many processors as C, the virtual processors in T_C may easily be simulated by the physical processors in C, each processor (except one) acting both as a leaf and as a particular internal node of T_C.

The counting is done for each variable x going up level by level in T_C, where C is the cluster containing the processor responsible for x. Initially C's processors acting as leaves of T_C transmit to their parents an integer, 1 if they have successfully accessed a copy of x, 0 otherwise (recall that this information was stored in an internal table

during the calls of ACCESS). Each internal node adds the two numbers received from its children and forwards the sum to its parent. After h steps, where h is the height of T_C, the root will have computed the total number of successful access attempts. If that number is $\geqq c$ ($<c$), the root sends the information that x is dead (alive) down the tree.

If x is read in the current step, the values and time stamps of copies of x are also propagated up the tree. Each internal node selects among the (value, time stamp) pairs of its children the one with the most recent time stamp. If x is declared dead the root is able to determine its valid value and to propagate it to the leaves of T_C where it will be stored internally by the processor responsible for x.

When the above procedure has been carried out for all variables concerned, each processor which is responsible for a now dead variable x read in the current step sends the value of x to the origin of x (this is the processor actually needing the value of x). Since the values transferred go to distinct destinations, this step may be carried out in constant time. At this point all memory accesses involving dead variables have been successfully simulated, allowing us to forget about variables that died in this call of COUNT. In particular, we shall no longer consider them to be in the custody of any processor.

Since all clusters operate simultaneously, doing the counting for one variable per cluster takes $O(h) = O(\log c) = O(\log \log m)$ time. We are actually doing the counting for r variables per cluster. However, since each variable uses only one level at a time in the corresponding tree T_C, it is possible to *pipeline* the propagation of information up and down the tree. Hence a call COUNT(r) may be executed in $O(r + \log \log m)$ time.

BROADCAST(r): ($1 \leqq r \leqq 2c-1$). Before all processors within a cluster can attempt to access the different copies of a variable x, the name of x and, if x is written in the current step, its new value must be broadcast by the processor responsible for x to all other processors in the cluster.

This is done using the same trees T_C as in COUNT. The processor responsible for x sends the information about x up the tree, the root sends copies down to both subtrees, etc., so that the information reaches all processors in the cluster and may be stored in a table within each processor. BROADCAST(r) operates only on variables in the custody of processors in $\mathcal{P}(r)$. Since a pipelining technique similar to the one adopted in COUNT may be used, the time bound of $O(r + \log \log m)$ again applies.

BROADCAST-ACCESS-AND-COUNT(s, r): ($s \geqq 1, 1 \leqq r \leqq 2c-1$). This is a simple combination of the above procedures defined by the program segment:

```
BROADCAST(r);
do s times
  for j := 1 to r do
    ACCESS(j);
  od;
  COUNT(r);
od;
```

A call BROADCAST-ACCESS-AND-COUNT(s, r) first broadcasts information about the live variables which are in the custody of some processor in $\mathcal{P}(r)$ (say again that these are in the range of the call). It then goes s times through a cycle in which it first attempts to access all live variables in the range and then calls COUNT to

determine which of the access attempts were successful. The execution time of BROADCAST-ACCESS-AND-COUNT(s, r) is $O(s(r+\log\log m))$.

REDISTRIBUTE. Even if (responsibilities for) the live variables are distributed about evenly among the clusters at some point in time, we cannot guarantee that this will still be true after several calls of ACCESS and COUNT since the processors in some clusters may be much more successful than processors in other clusters. It is crucial for efficiency to periodically average out the work load among the clusters. We do this by means of two variants of a procedure REDISTRIBUTE.

As we argue below, the redistribution problem is reducible to that of sorting the variables. This observation together with a fast parallel sorting algorithm [AKS] immediately gives us an $O(\log n)$-time procedure EXACT-REDISTRIBUTE for doing a perfect redistribution of the live variables. However, since we cannot always afford to spend $\Theta(\log n)$ time, we also need another redistribution procedure which is considerably faster but, in exchange, less accurate. Here the construction is more complicated. A comparatively short argument given below proves an upper bound on the time needed for an approximate redistribution. The bound, although weak, suffices for our purposes. Since we consider the redistribution problem to be interesting and important in its own right, however, we reconsider the redistribution problem in § 4 and obtain a much stronger result. Both of our proofs are inspired by the construction of ε-halvers in [AKS]. Let us first state the problem precisely. We distinguish between a weak and a strong form. The weak form will be used in the present section and the strong form in § 4.

DEFINITION. Consider n locations divided into k equal-size groups. Each location may hold a record, and given a set of n records, we call a particular association between the locations and the records a *configuration.* For integers R and y, the *weak* (*strong*) *redistribution problem* with n elements, upper bound R, and y allowable errors is as follows: Given a configuration of n records, each containing a key equal to either 0 or 1 and such that the set U of records with keys equal to 0 contains at most R elements, permute the records so that, in the resulting configuration for some set $V \subseteq U$ with $|U \setminus V| \leqq y$, there is no group whose locations contain more than $\lceil R/k \rceil$ ($\lceil |U|/k \rceil$, respectively) records belonging to V.

In our application the groups correspond to clusters of processors, and the records to names of variables together with any other relevant information (e.g., if a variable is read in the current step, an identification of its origin). A key equal to 0 indicates a live variable. Hence the strong redistribution problem asks for the live variables to be evenly redistributed among the clusters such that no more than y live variables "go wrong." The redistribution problem is closely related to that of approximate data compaction:

DEFINITION. Consider n locations arranged in a fixed order and each capable of holding a record. For integers R and y, the *weak* (*strong*) *approximate data compaction problem* with n elements, upper bound R, and y allowable errors is as follows: Given a configuration of n records, each with a key equal to either 0 or 1 and such that the number Q of records with keys equal to 0 is at most R, permute the records such that in the resulting configuration at most y records with keys equal to 0 are not in one of the first R (Q, respectively) locations.

We shall consider algorithms for the above problems consisting of a sequence of *parallel comparison-exchange steps.* A *comparison-exchange operation* compares the keys of two records A and B, with the location of A, say, preceding the location of B. If B's key is smaller than A's key, the records A and B are swapped. A parallel

comparison-exchange step on a set $\mathscr{L}$ of locations consists of a number of comparison-exchange operations executed in parallel on disjoint pairs of locations in $\mathscr{L}$. It is clear that if each of n MPC processors holds one of n locations, the MPC can execute any parallel comparison-exchange step on the n locations in constant time.

Suppose that we are given a redistribution problem. If we number the groups $1, \cdots, k$ and for $i = 1, \cdots, k$ denote the locations in the ith group by $L_{i,1}$, $L_{i,2}, \cdots, L_{i,n/k}$, then it is easy to see that solving the weak (strong) approximate data compaction problem with upper bound R and y allowable errors relative to the location order

$$L_{1,1}, L_{2,1}, \cdots, L_{k,1}, L_{1,2}, \cdots, L_{k,2}, \cdots, L_{1,n/k}, \cdots, L_{k,n/k}$$

also solves the weak (strong) redistribution problem with upper bound R and y allowable errors. Hence we need only consider the approximate data compaction problem which allows of a more convenient terminology. Note that data compaction is a special case of sorting in which there are only two distinct key values.

For all $p, q \in \mathbb{N}$, let

$$[p]_q = p(p-1)\cdots(p-q+1).$$

We shall make repeated use of the inequalities given in the proposition below of which one expresses a simple combinatorial fact and the other follows from Stirling's approximation of the factorial function.

PROPOSITION 1. (1) *Given sets A, A', B, B' with $A' \subseteq A$, $B' \subseteq B$ and $|A| \leqq |B|$, the fraction, among all injective functions $A \to B$, of those injective functions $A \to B$ which map each element of A' to an element of B' is*

$$\frac{[|B'|]_{|A'|}}{[|B|]_{|A'|}} \leqq \left(\frac{|B'|}{|B|}\right)^{|A'|}.$$

(2) *For all $p, q \in \mathbb{N}$ with $q \geqq 1$,*

$$\binom{p}{q} \leqq \left(\frac{ep}{q}\right)^q.$$

LEMMA 2.1. *For all n, R, and y with $1 \leqq y \leqq R \leqq n/2$, the weak approximate data compaction (or redistribution) problem with n elements, upper bound R, and y allowable errors is solved by an algorithm consisting of $\lceil 4(n/y)\log(n/y)\rceil$ parallel comparison-exchange steps on n elements.*

Proof. Let us consider the approximate data compaction problem and call it that of "compacting with at most y errors." Let the ordered sequence of locations be $L_1, \cdots, L_n$. We shall henceforth ignore the actual information to be permuted and consider each record to consist of a single bit, its key. Following [AKS], we translate the problem into a graph-theoretic setting and show the existence of certain expander-like graphs. Let $A = \{L_1, \cdots, L_R\}$, $B = \{L_{R+1}, \cdots, L_n\}$, and $\nu \in \mathbb{N}$. For each choice of ν injective functions $\pi_1, \cdots, \pi_\nu$ mapping A into B, we may construct a labeled bipartite graph on the node sets A and B by drawing an edge labeled l from a to $\pi_l(a)$ for all $l = 1, \cdots, \nu$, $a \in A$. Let $\mathscr{G}_\nu$ denote the set of all such labeled graphs obtained by varying $\pi_1, \cdots, \pi_\nu$.

To each graph in $\mathscr{G}_\nu$ there corresponds an algorithm consisting of ν parallel comparison-exchange steps on $L_1, \cdots, L_n$: Interpret an edge labeled l linking $L_i \in A$ and $L_j \in B$ as a comparison-exchange operation between L_i and L_j to be performed in the lth step. Note that by the ordering of the locations, the smaller value will be

placed in L_i. For $a \in A' \subseteq A$ and $b \in B' \subseteq B$, we will say that the edge (a, b) (if it exists) joins A' and B'. Now call a graph $G \in \mathcal{G}_\nu$ "good" if it has the following property:

> For all sets $A' \subseteq A$, $B' \subseteq B$ with $|A'| = |B'| = y$, there is at least one edge in G joining A' and B'.

We claim that the algorithm $\mathcal{A}$ corresponding to a good graph compacts with at most y errors. Suppose the contrary. Then there is a set $B' \subseteq B$, $|B'| = y$, such that all locations in B' contain a 0 after the execution of $\mathcal{A}$. Since the total number of 0's is at most R, there is also a set $A' \subseteq A$, $|A'| = y$, such that all locations in A' contain a 1 after the execution of $\mathcal{A}$. Since the graph is good, there is at least one edge joining A' and B'. Suppose that such an edge is labeled l and links $L_i \in A$ with $L_j \in B$. Then after the lth step of the algorithm, the value held by L_j is no less than the value held by L_i by the interpretation of the edge. On the other hand, L_i holds a 1 and L_j holds a 0 after the last step. But this is impossible since the values held by locations in A never increase, while those held by locations in B never decrease.

It still remains to show that $\mathcal{G}_\nu$ contains good graphs for all n for sufficiently small values of ν. If a graph in $\mathcal{G}_\nu$ is not good (call it "bad"), there are sets $A' \subseteq A$, $B' \subseteq B$, $|A'| = |B'| = y$, such that no edge joins A' and B'. Hence the fraction f_ν of bad graphs in $\mathcal{G}_\nu$ is bounded by

$$\binom{R}{y}\binom{n-R}{y}\left(\frac{[(n-R)-y]_y}{[n-R]_y}\right)^\nu.$$

Here $\binom{R}{y}$ is the number of possible choices of A', $\binom{n-R}{y}$ is the number of possible choices of B', and $[(n-R)-y]_y/[n-R]_y$, for fixed A' and B', is by Proposition 1 the fraction of injective mappings of A into B with the property that no element of A' is mapped to an element of B'. Using again Proposition 1, we get

$$f_\nu \leqq \left(\frac{eR}{y}\right)^y \left(\frac{e(n-R)}{y}\right)^y \left(\frac{(n-R)-y}{n-R}\right)^{\nu y} \leqq \left(\frac{e^2 n^2}{y^2}\left(1-\frac{y}{n}\right)^\nu\right)^y,$$

and it is clear that $f_\nu < 1$ provided that

$$\nu > \frac{2\log(en/y)}{-\log(1-y/n)}.$$

Finally observe that since

$$-\log\left(1-\frac{y}{n}\right) \geqq \frac{y}{n}\log e \quad \text{and} \quad \frac{n}{y} \geqq 2,$$

we have

$$\frac{2\log(en/y)}{-\log(1-y/n)} \leqq 2\frac{n}{y}\left(\log e + \log\frac{n}{y}\right)\frac{1}{\log e} = 2\frac{n}{y}\log\frac{n}{y}\left(\frac{1}{\log(n/y)} + \frac{1}{\log e}\right) < 4\frac{n}{y}\log\frac{n}{y}.$$

□

Let us denote a call of the MPC algorithm implied by Lemma 2.1 by APPROXIMATE-REDISTRIBUTE(R, y). As a technical point, define APPROXIMATE-REDISTRIBUTE(R, 0) for arbitrary R to be equivalent to EXACT-REDISTRIBUTE. Note that we may assume that a call APPROXIMATE-REDISTRIBUTE(R, y) moves all except y live variables to processors in $\mathcal{P}(\lceil R/k \rceil)$ and analogously for EXACT-REDISTRIBUTE, i.e., the live variables are moved to the *first* processors in each cluster.

The algorithm for simulating the memory accesses of one PRAM step consists of three parts: Part 1 decreases the number of live variables from at most n to $O(n \log\log m/\log m)$, Part 2 reduces it further to $O(m2^{-\log m/\log\log m})$, and Part 3 eliminates all remaining live variables.

The following program segment describes Part 1 of the algorithm. It uses a variable R that will be an upper bound on the number of live variables. a_1 and γ are design parameters to be determined in the subsequent analysis of the algorithm. Let us call an execution of lines (4)-(6) a stage. Part 1 of the algorithm consists of an initialization phase followed by $u = \lceil \log\log m - \log\log\log m \rceil$ stages.

Part 1

```
(1) For each origin P of a variable x,
      declare P to be responsible for x;
(2) R := n;
(3) do ⌈log log m − log log log m⌉ times
(4)   BROADCAST-ACCESS-AND-COUNT(a_1, ⌈R/k⌉);
(5)   R := ⌊R/2⌋;
(6)   APPROXIMATE-REDISTRIBUTE(R, ⌊γR⌋);
(7) od;
```

So far we have said nothing about how the $2c-1$ copies of each variable are to be distributed among the memory modules of the MPC. Any such arrangement may be represented by a Boolean $m \times n$ matrix whose rows correspond to variables and whose columns correspond to memory modules. The ijth entry is 1 if and only if the jth memory module contains a copy of the ith variable (i.e., each row in the matrix has exactly $2c-1$ entries equal to 1). Such an arrangement is called a *memory organization scheme* (MOS).

By a counting argument, Upfal and Wigderson proved the following lemma:

LEMMA 2.2 (Lemma 3.3 in [UW], slightly adapted). *There exists a constant η, $0 < \eta < 1$, such that for all n and m there is an* MOS *with the following property*:

> *For any call of BROADCAST-ACCESS-AND-COUNT(s, r), if the number of live variables in the range of the call before the call is w, then after the call it is at most $2\eta^s w$.*

Furthermore, the fraction of MOS*'s not having the above property among all possible* MOS*'s is $o(1/n)$.*

We are now in a position to analyze Part 1 of the algorithm. Choose a constant η and an MOS as given by Lemma 2.2. Then let a_1 be an integer large enough to make $2\eta^{a_1}$ strictly less than $\frac{1}{2}$ and put $\xi = 2\eta^{a_1}$, $\gamma = (\frac{1}{2} - \xi)/(1-\xi)$. Denote for $i = 0, 1, \cdots$ by R_i the value of the variable R after exactly i stages, and let $r_{i+1} = \lceil R_i/k \rceil$. We will show by induction that the invariant

At the beginning of the ith stage, there are at most R_{i-1} live variables

holds for $i = 1, 2, \cdots$. Since $R_0 = n$, the assertion is trivially true for $i = 1$. Thus let $i \geqq 1$ and assume that at the beginning of the ith stage, there are at most R_{i-1} live variables. We must show that the ith stage reduces the number of live variables to R_i.

If $i > 1$, the inductive hypothesis implies that the call of APPROXIMATE-REDISTRIBUTE which ended the previous stage was applied to a legal input. In particular, the first argument R is always at most $n/2$ as required by our construction. Hence we may conclude that $y \leqq \gamma R_{i-1}$, where y is the number of live variables outside the range of the call of BROADCAST-ACCESS-AND-COUNT in the ith stage. But

then the total number Q of live variables at the end of the ith stage is at most

$$y+\xi(R_{i-1}-y)=(1-\xi)y+\xi R_{i-1}\leqq((1-\xi)\gamma+\xi)R_{i-1}=R_{i-1}/2,$$

and since Q is an integer, we have in fact $Q\leqq\lfloor R_{i-1}/2\rfloor=R_i$, completing the inductive step. Thus Part 1 of the algorithm reduces the number of live variables to at most

$$n2^{-u}\leqq n\frac{\log\log m}{\log m}.$$

As for the running time $T(n, m)$ of Part 1, we have

$$T(n, m)=O\left(\sum_{i=1}^{u}(r_i+\log\log m+S_i)\right)$$

where S_i is the time spent in the ith call of APPROXIMATE-REDISTRIBUTE. Now

$$\sum_{i=1}^{u}r_i\leqq\sum_{i=1}^{u}\left(\frac{R_{i-1}}{k}+1\right)\leqq\frac{n}{k}\sum_{i=0}^{\infty}2^{-i}+u=O(\log m),$$

and since $n/(\gamma R_i)\leqq 2^{i+1}/\gamma=O(2^i)$, we have $S_i=O(i2^i)$ by Lemma 2.1 and

$$\sum_{i=1}^{u}S_i=O(u2^u)=O(\log m).$$

Hence $T(n, m)=O(\log m)$.

Part 2 of the algorithm is as follows:

Part 2

EXACT-REDISTRIBUTE;
BROADCAST-ACCESS-AND-COUNT($\lceil a_1\log m/\log\log m\rceil$, $\lceil 2\log\log m\rceil$);

The exact redistribution together with the fact that Part 1 leaves at most $n(\log\log m/\log m)\leqq 2k\log\log m$ live variables guarantees that all live variables are in the range of the call of BROADCAST-ACCESS-AND-COUNT. Hence the number of live variables is reduced to at most

$$n\frac{\log\log m}{\log m}2\eta^{a_1\log m/\log\log m}\leqq n2^{-\log m/\log\log m}.$$

The running time is $O(\log n+(\log m/\log\log m)\log\log m)=O(\log m)$. Summing up, we have shown:

LEMMA 2.3. *For a suitable choice of an MOS and of the constants a_1 and γ, Parts 1 and 2 together reduce the number of live variables to at most $n2^{-\log m/\log\log m}$. The execution time of each part is $O(\log m)$.*

Part 3 is described by the program segment:

Part 3

```
(1) EXACT-REDISTRIBUTE;
(2) BROADCAST(1);
(3) do ⌈a2 log n/log log m⌉ times
(4)   do ⌈a3 log log m⌉ times
(5)     ACCESS(1);
(6)   od;
(7)   COUNT(1);
(8) od;
```

Call an execution of lines (4)-(7) a stage. a_2 and a_3 are design parameters to be chosen later. Since the execution time of ACCESS is $O(1)$ and that of COUNT(1) is

$O(\log\log m)$, Part 3 clearly requires no more than $O(\log n)$ time. It remains to show that Part 3 eliminates all remaining live variables for suitable constants a_2 and a_3.

Notice that Part 3 uses a strategy different from the one adopted in Parts 1 and 2. A variable is accessed *several* (in fact, $\Theta(\log\log m)$) times before a COUNT is done to find out if it has died. Thus variables may be accessed even though they have fewer than c remaining live copies (copies that were not yet accessed). This means that Lemma 2.2 is not applicable. We will now show, however, that the idea of Lemma 2.2 still works as long as the variables being accessed have an average of at least c^q live copies each for some $q>0$.

LEMMA 2.4. *For all constants p, q and η with $0<p,\ q,\ \eta<1$ and $p+q>1$, there is a constant N such that for all n, m with $m>n\geqq N$, there exists an MOS with the following property*:

> *If a call of ACCESS attempts to access exactly $Q\leqq n2^{-(\log m)^p}$ variables and the total number of live copies of these variables before the call is $W\geqq Qc^q$, then after the call at most ηW of the copies are still alive.*

Furthermore, the fraction of MOS's not having the above property among all possible MOS's tends to 0 as n tends to infinity.

Proof. Call an MOS "good" if for any choice of $Q\leqq Q_{\max}=\lfloor n2^{-(\log m)^p}\rfloor$ variables and any choice of $W\geqq Qc^q$ live copies of these variables, the number of modules containing at least one of the W live copies is at least $(1-\eta)W$. It is clear that a good MOS guarantees the elimination of at least $(1-\eta)W$ live copies. Hence we need only show that there are sufficiently many good MOS's.

For each choice of Q and W ($1\leqq Q\leqq Q_{\max}$, $Qc^q\leqq W\leqq Q(2c-1)$) denote by $f_{Q,W}$ the fraction of MOS's that fail to have the above property because they map some set of W copies of Q variables to fewer than $(1-\eta)W$ modules. We now derive an upper bound on $f_{Q,W}$. There are

$\binom{m}{Q}$ ways to choose the set of Q variables,

$\binom{(2c-1)Q}{W}$ ways to choose the W live copies from the set of all copies of the Q variables, and

$\binom{n}{\lfloor(1-\eta)W\rfloor}$ ways to choose $\lfloor(1-\eta)W\rfloor$ modules to contain all the live variables.

For each choice of W live copies and $\lfloor(1-\eta)W\rfloor$ modules, the fraction of MOS's that map the chosen live copies to the chosen modules is by Proposition 1 at most

$$\left(\frac{\lfloor(1-\eta)W\rfloor}{n}\right)^W.$$

Hence

$$\begin{aligned}
f_{Q,W} &\leqq \binom{m}{Q}\binom{(2c-1)Q}{W}\binom{n}{\lfloor(1-\eta)W\rfloor}\left(\frac{\lfloor(1-\eta)W\rfloor}{n}\right)^W\\
&\leqq \left(\frac{em}{Q}\right)^Q\left(\frac{e(2c-1)Q}{W}\right)^W\left(\frac{en}{\lfloor(1-\eta)W\rfloor}\right)^{\lfloor(1-\eta)W\rfloor}\left(\frac{\lfloor(1-\eta)W\rfloor}{n}\right)^W\\
&\leqq [e^{2-\eta+Q/W}(1-\eta)^\eta m^{Q/W}Q^{1-Q/W}(2c-1)W^{\eta-1}n^{-\eta}]^W\\
&\leqq \left[2e^{2-\eta+Q/W}m^{Q/W}c\left(\frac{Q}{W}\right)^{1-\eta}Q^\eta n^{-\eta}\right]^W
\end{aligned}$$

$$\leqq \left[2e^3 m^{c^{-q}} c^{1-q(1-\eta)} \left(\frac{Q}{n}\right)^{\eta} \right]^W,$$

where the relation $Q/W \leqq c^{-q}$ was used in the last step. We will show that the quantity g_Q in square brackets is dominated by its last factor $(Q/n)^{\eta}$. Consider for this purpose its logarithm:

$$\log g_Q = 1 + 3 \log e + c^{-q} \log m + (1 - q(1-\eta)) \log c + \eta \log \frac{Q}{n}.$$

Now

$$c^{-q} \log m = O((\log m)^{1-q}) \quad \text{and} \quad (1 - q(1-\eta)) \log c = O(\log \log m)$$

while

$$\log \frac{Q}{n} \leqq -(\log m)^p.$$

Since $1 - q < p$, we may conclude that for all χ with $0 < \chi < \eta$,

$$\log g_Q \leqq -\chi (\log m)^p$$

and hence

$$f_{Q,W} \leqq 2^{-\chi W (\log m)^p}$$

for all sufficiently large values of m.

Now the fraction f of bad MOS's (for some values of Q and W) is bounded as follows:

$$f \leqq \sum_{Q=1}^{Q_{\max}} \sum_{W=\lceil Qc^q \rceil}^{(2c-1)Q} f_{Q,W} \leqq Q_{\max} \sum_{W=\lceil c^q \rceil}^{\infty} 2^{-\chi W (\log m)^p}$$

$$\leqq Q_{\max} 2^{-\chi c^q (\log m)^p} \sum_{W=0}^{\infty} (2^{-\chi (\log m)^p})^W \leqq Q_{\max} 2^{-\kappa (\log m)^{p+q}}$$

for some constant $\kappa > 0$ (and sufficiently large values of m). Using $Q_{\max} \leqq n$, we finally get

$$\log f \leqq -\kappa (\log m)^{p+q} + \log n,$$

and since $p + q > 1$ and $m \geqq n$, this shows that $f \to 0$ as $n \to \infty$. In particular, good graphs exist for all sufficiently large values of n. This concludes the proof of Lemma 2.4. □

For $i = 1, 2, \cdots$, let Q_i be the number of live variables immediately before the ith stage of Part 3 of the algorithm. By Lemma 2.3,

$$Q_1 \leqq n 2^{-\log m / \log \log m}.$$

This is less than $k = n/(2\lfloor \log m \rfloor - 1)$ and also less than $n 2^{-(\log m)^p}$ for any $p < 1$ for sufficiently large values of m. Hence all remaining live variables are in the range of the calls of ACCESS, and Lemma 2.4 is applicable. Let p, q and η be constants with $0 < p, q, \eta < 1$ and $p + q > 1$ and choose an MOS as given by Lemma 2.4.

We want the following assertion (1) to be true for all i:

(1) The number of live copies decreases in the ith stage from at most $Q_i(2c-1)$ to at most $Q_i c^q$.

As long as there are more than $Q_i c^q$ live copies, each call of ACCESS will decrease their number by a factor of at least η by Lemma 2.4. Hence s iterations of the inner loop are sufficient to guarantee (1) if

$$\eta^s Q_i(2c-1) \leqq Q_i c^q,$$

i.e., if

$$s \geqq \log_{1/\eta} \frac{2c-1}{c^q}.$$

Since $c \leqq \log m$, we have $\log_{1/\eta}((2c-1)/c^q) = O(\log \log m)$. Hence (1) holds given only that a_3 is chosen large enough.

By (1), the number of live copies at the beginning of the $(i+1)$st stage, for $i = 1, 2, \cdots$, is at most $Q_i c^q$. And each live variable has at least c live copies since the ith stage ended with a call of COUNT. But then

$$Q_{i+1} \leqq \frac{Q_i c^q}{c} = Q_i c^{q-1} \leqq Q_1 (c^{q-1})^i,$$

where the last step followed by induction. Hence t stages will eliminate all live variables if

$$Q_1 (c^{q-1})^t < 1,$$

i.e., if

$$t > \frac{\log Q_1}{(1-q)\log c}.$$

Since $Q_1 \leqq n$ and $\log c = \Omega(\log \log m)$, the smallest solution t is $O(\log n / \log \log m)$. In conclusion, we have proved:

LEMMA 2.5. *For a suitable choice of an MOS and of the constants a_2 and a_3, Part 3 eliminates all live variables remaining after the execution of Parts 1 and 2. Its execution time is $O(\log n)$.*

In order to combine Parts 1, 2 and 3 of the algorithm we must use an MOS that satisfies both Lemma 2.2 and Lemma 2.4 (i.e., has the properties described in the lemmas). But since the fraction of MOS's not satisfying Lemma 2.2 goes to 0 as n goes to infinity, as does the fraction not satisfying Lemma 2.4, the proportion of MOS's satisfying both conditions tends to 1, i.e., such MOS's certainly exist. Even a randomly chosen MOS for large n with high probability has the desired properties.

Summarizing our results about Parts 1, 2 and 3 as expressed in Lemmas 2.3 and 2.5, we have the following theorem:

THEOREM 1. *The shared memory of any* EREW PRAM *with n processors and m cells of shared memory may be emulated by an n-processor* MPC *with a slowdown of $O(\log m)$.*

Remark 1. As indicated in the first paragraph of § 2, Theorem 1 remains true even if we allow the simulated machine to be a CRCW PRAM.

Remark 2. It follows from the discussion towards the end of § 1 that for any PRAM program H running on an n-processor CRCW PRAM and using at most m cells of shared memory, there exists an equivalent MPC program which runs on an n-processor MPC and simulates the execution of H with a slowdown of $O(\log m)$.

3. Simulation of PRAMs on bounded-degree networks. This section considers emulations of PRAM memories by bounded-degree networks. We give an upper bound

on the slowdown which follows as a corollary to Theorem 1 and devote most of the section to the proof of a lower bound of $\Omega(\log n \log m/\log\log m)$. A similar lower bound was shown independently by Karlin and Upfal [KU].

For added clarity, let us use the word "cycle" instead of "step" when talking about network computations. In this section we will say that a data item is stored in a network processor meaning that it is stored in that processor's memory module.

THEOREM 2. *The shared memory of any CRCW PRAM with n processors and m cells of shared memory may be emulated by an n-processor bounded-degree network with a slowdown of* $O(\log n \log m)$.

Proof. This is a consequence of Theorem 1, the remark following it, and the general result that any step of an n-processor MPC may be simulated by $O(\log n)$ cycles of an n-processor bounded-degree network. The latter fact in turn follows from the existence of algorithms which sort on bounded-degree networks in $O(\log n)$ time ([AKS], [L]), together with the observation that a scheme for doing partial routing within the same time bound may be derived from any such algorithm.

Let us now turn to the lower bound and first describe our model and the assumptions made. We consider an n-processor bounded-degree network emulating the shared memory of an n-processor PRAM and identify the network processors with the nodes and the links between network processors with the edges of an undirected graph $G=(V, E)$. The distance $\text{dist}(u, v)$ between two network processors u and v is the number of edges on a shortest path in G from u to v.

Recall that when a PRAM processor executes the instruction STORE x, it presents to its associated network processor a pair $\langle$"x", $a\rangle$, where a is the new value of x. We call such a pair a *copy* of x. The copy is said to be *valid* until the next PRAM step in which x is updated (forever, if x is never again updated). We will assume that any network processor reading a PRAM variable x in a given PRAM step must contain a valid copy of x at the end of that step. We also assume that the network treats copies of variables as indivisible entities capable only of being input from or output to PRAM processors, of being routed through the network and of being stored in network processors (cf. Fig. 4). In particular, the network cannot "synthesize" copies of variables.

Regarding the cost of communication, we will assume that for each pair (u, v) of network processors, sending a copy of a variable from u to v needs at least $\text{dist}(u, v)$ *atomic actions* by the network, where an atomic action is a "processor-step," i.e., the amount of work associated with a single network processor executing a single RAM instruction. This is reasonable since each processor on a path in G from u to v must devote at least one cycle to passing on the copy.

We finally require all communication in the network to be *point to point.* Note that while our other assumptions were innocuous, this is a serious restriction. Point to point communication means that copies of variables are physical entities that cannot be subjected to duplication by the network. We allow a network processor writing a PRAM variable x to obtain an arbitrary number of copies of x from its associated PRAM processor; but any network processor which receives a copy of x from a network neighbor may forward the copy to at most one neighboring (PRAM or network) processor. As an example, if a network processor u writing a PRAM variable x wants to send a copy of x to each of h processors lying close together in the network but far from u, then u must obtain h separate copies from its associated PRAM processor and have them routed through the network. A less restrictive model would allow u to send just one copy that would be replicated by the network in the vicinity of the h destinations. We will use the assumption of point to point communication to conclude

that if during the simulation there are h instances of a processor sending a copy of a variable to a processor at a distance of at least q, then the network must perform a total of at least hq atomic actions.

Most of our assumptions may be visualized by imagining copies of variables to be contained in sealed envelopes which are carried by the network between processors of the PRAM.

Note that the assumption of point to point communication is not satisfied by our emulation algorithm, nor by the one by Upfal and Wigderson. Both algorithms use a form of load redistribution in which conceptually variables, not copies of variables, are transported through the network. This corresponds to "bundles" of $2c-1$ copies of the same variable being transported together at unit cost or, equivalently, to the replication of copies after they have been sent, something which is forbidden by the assumption of point to point communication. Thus the lower bound which we are about to derive does not apply to any of these cases. Partly due to the well-known difficulty of obtaining any sort of lower bounds for most natural problems, we nevertheless believe the result to be interesting.

Upfal and Wigderson showed in [UW] an $\Omega(\log n/\log\log n)$ lower bound on the slowdown associated with a simulation of a PRAM on an MPC. Our proof is a variation of their argument which we will therefore briefly review.

They start out with the observation (also made in [MV]) that if there is an average of r valid copies of each variable, then some PRAM steps in which all processors read need $\Omega((m/n)^{1/(2r)})$ cycles for their simulation. On the other hand, if an average of r valid copies is maintained, then the average cost of simulating a writing step is $\Omega(r)$. Thus one cannot do better than

$$\Omega\left(\min_r\left(\left(\frac{m}{n}\right)^{1/(2r)}+r\right)\right)=\Omega\left(\frac{\log n}{\log\log n}\right),$$

the desired lower bound. Of course, this informal argument ignores a lot of detail.

The main ideas behind our modified proof are as follows: Suppose that a processor u sends copies of some variable to a number of other processors. We will then (in the lower bound argument) count only those copies that go to processors at a distance of at least $q\approx\log n/(2\log d)$ from u, where d is the maximum degree of the network; transmission of all other copies is considered free. Since every counted copy requires $\Omega(\log n)$ atomic actions, the network will need $\Omega(r\log n)$ cycles to simulate a writing step updating n variables if an average of r counted copies per variable is to be kept.

What happens to reading steps? There is now for every variable an average of r counted copies and at most $\approx d^q\approx\sqrt{n}$ free copies. However, the free copies always cluster in a small "sphere" in the network, and the free copies of many variables cluster in the same sphere. This allows us to prove that the free copies are of little use in the sense that the simulation of a reading step may require $\Omega((m/(4n^2))^{1/(4r)})$ cycles. Thus the slowdown is

$$\Omega\left(\min_r\left(\left(\frac{m}{4n^2}\right)^{1/(4r)}+r\log n\right)\right)=\Omega\left(\frac{\log n\log m}{\log\log m}\right).$$

The details follow.

Before we state the lower bound, let us define a single instruction-multiple data (SIMD) program for a PRAM to be a program whose execution never causes two processors to carry out different types of instructions in the same step.

THEOREM 3. *If $m=\Omega(n^{2+\varepsilon})$ and $T\geqq(1+\varepsilon)(m/n)$ for some fixed $\varepsilon>0$, then the worst-case simulation time for a straight-line SIMD program running for T steps on an*

n-processor EREW PRAM with m cells of shared memory is

$$\Omega\left(T \min\left\{\sqrt{n \log n}, \frac{\log n \log m}{\log \log m}\right\}\right)$$

for any on-line emulation of the shared PRAM memory by an n-processor bounded-degree network that uses only point to point communication.

Remark 1. In particular, the (worst-case) slowdown is $\Omega(\min\{\sqrt{n \log n}, \log n \log m/\log \log m\})$, but the theorem is slightly stronger: There are arbitrarily long programs that cause an "average slowdown" of $\Omega(\min\{\sqrt{n \log n}, \log n \log m/\log \log m\})$.

Remark 2. Under the assumptions of Theorem 3, the simulation time is $\Omega((\log n)^2/\log \log n)$. If m is polynomial in n, this statement is equivalent to that of Theorem 3.

Remark 3. The emulation being on-line means that each PRAM step must be simulated without knowledge of memory requests in later steps. If this requirement is relaxed, it is possible in some cases to beat the lower bound; see [VW].

Proof. Let $\tau = \lfloor (T - \lceil m/n \rceil - 1)/2 \rfloor$ and observe that $\tau = \Omega(T)$. We consider PRAM programs in which the ith processor, for $i = 1, \cdots, n$, executes a program of the following form:

```
(1) LOADINDEX;
(2) for t := 1 to ⌈m/n⌉ do
(3)   STORE y_I^{t,i};
(4) od;
(5) for t := 1 to τ do
(6)   LOAD y_R^{t,i};
(7)   STORE y_W^{t,i};
(8) od;
```

Here the first instruction loads a well-defined value (the processor index, say) into the accumulator, and the various y's are PRAM variables. The above program as given of course does not have the straight-line format, but may be thought of as a compact representation of a straight-line program whose length is bounded by T.

We will call an execution of lines (6)-(7) a stage. The tth stage, for $t = 1, \cdots, \tau$, consists of a reading part which inspects all variables in the set $Y_R^t = \{y_R^{t,1}, \cdots, y_R^{t,n}\}$, followed by a writing part which updates all variables in the set $Y_W^t = \{y_W^{t,1}, \cdots, y_W^{t,n}\}$.

Let X be the set of the PRAM's m shared variables. Lines (2)-(4) simply assign values to all variables in X in some arbitrary fixed way.

Suppose now that programs of the above form are run on a PRAM whose shared memory is emulated by a bounded-degree network satisfying the assumptions of Theorem 3 and running for at most S cycles. We will demonstrate that S is large by exhibiting a particular choice of

$$(Y_R^1, Y_W^1, \cdots, Y_R^\tau, Y_W^\tau)$$

which forces the simulation to be slow. We fix the Y's one by one, choosing Y_R^t and Y_W^t depending on the state of the emulating network at the beginning of the tth stage. The salient parameters of the state of the network are the *redundancies* of variables in X as defined below.

Each network processor which writes a PRAM variable x may send copies of x to other processors. For $t = 1, \cdots, \tau + 1$ and $x \in X$, denote by Γ_x^t the set of network processors which have a valid copy of x at the beginning of the tth stage ("at the

beginning of the $(\tau+1)$st stage" should be interpreted to mean at the end of the τth stage). Identify the network with a graph $G=(V, E)$ as in the definition of our model, let $d \geqq 2$ be the maximum node degree of G, and for $u \in V$, let $B(u)$ be the sphere of radius $q=(\log n-2 \log \log n)/(2 \log d)$ around u, i.e.,

$$B(u)=\{v \in V \mid \operatorname{dist}(u, v) \leqq q\}.$$

For $t=1, \cdots, \tau+1$ and $x \in X$, the redundancy r_x^t of x at the beginning of the tth stage is defined as

$$r_x^t=\min_{u \in V}|\Gamma_x^t \backslash B(u)|;$$

i.e., take the sphere $B(u)$ best fitting the copies of x and count the number of copies outside $B(u)$. The total redundancy R^t and the average redundancy r^t at the beginning of the tth stage are

$$R^t=\sum_{x \in X} r_x^t$$

and

$$r^t=R^t/m.$$

The following Lemma 3.1 generalizes Theorem 4.1 in [UW] and § III.1 of [MV].

LEMMA 3.1. *Let $1 \leqq t \leqq \tau$ and suppose that $Y_R^1, Y_W^1, \cdots, Y_R^{t-1}, Y_W^{t-1}$ (and hence r^t) are fixed. Then it is possible to choose Y_R^t in such a way that the tth stage of the simulation (in fact, its reading part) must consist of $\Omega(g(r^t))$ cycles, where*

$$g(r)=\begin{cases}\min\left\{\sqrt{n} \log n, \dfrac{n}{r}, \left(\dfrac{m}{4n^2}\right)^{1/(4r)}\right\} & \text{if } r>0, \\ \sqrt{n} \log n & \text{if } r=0.\end{cases}$$

Proof. Let us drop the superscript t from Γ_x^t, r_x^t, R^t and r^t. Let $\{X_v \mid v \in V\}$ be a partition of X (empty sets allowed) such that

$$\forall x \in X_v: r_x=|\Gamma_x \backslash B(v)|.$$

Then there is an X_v which is not much smaller than average and whose redundancy is not much larger than average, i.e.,

Claim 1.

$$\exists v \in V: |X_v| \geqq \frac{m}{2n} \quad \text{and} \quad \sum_{x \in X_v} r_x \leqq 2r|X_v|.$$

Proof. Assume otherwise and let $U=\{v \in V \mid |X_v| \geqq m/(2n)\}$. Then $U \neq \varnothing$ and

$$R \geqq \sum_{v \in U} \sum_{x \in X_v} r_x > \sum_{v \in U} 2r|X_v| > 2r\frac{m}{2}=R$$

since

$$\sum_{v \notin U}|X_v|<n\frac{m}{2n}=\frac{m}{2}.$$

This is a contradiction. □

Choose $v \in V$ as given by Claim 1, i.e., $|X_v| \geqq m/(2n)$ and $\sum_{x \in X_v} r_x \leqq 2r|X_v|$. Then there is a set $A \subseteq X_v$ with $|A| \geqq |X_v|/2 \geqq m/(4n)$ such that $\forall x \in A$: $r_x \leqq 4r$. Let $f = |B(v)|$. A simple combinatorial argument shows that

$$f \leqq d^q + 1 = \frac{\sqrt{n}}{\log n} + 1.$$

Now for any set $W \subseteq V \setminus B(v)$, let

$$A_W = \{x \in A \mid \Gamma_x \subseteq B(v) \cup W\}.$$

Claim 2. For any k with $0 \leqq k \leqq n - f$, there is a subset W of $V \setminus B(v)$ with $|W| = k$ and

$$|A_W| \geqq |A| \binom{n - f - \lfloor 4r \rfloor}{k - \lfloor 4r \rfloor} \Big/ \binom{n-f}{k}.$$

Proof. Let $\mathcal{W} = \{W \subseteq V \setminus B(v) \mid |W| = k\}$. For any $x \in A$, there are exactly

$$\binom{n - f - r_x}{k - r_x}$$

sets $W \in \mathcal{W}$ with $x \in A_W$. Since $r_x \leqq \lfloor 4r \rfloor$ and hence

$$\binom{n - f - r_x}{k - r_x} \geqq \binom{n - f - \lfloor 4r \rfloor}{k - \lfloor 4r \rfloor},$$

we have

$$\sum_{W \in \mathcal{W}} |A_W| \geqq |A| \binom{n - f - \lfloor 4r \rfloor}{k - \lfloor 4r \rfloor}.$$

Since

$$\mathcal{W} = \binom{n-f}{k},$$

Claim 2 now follows by the pigeonhole principle. □

We can now finish the proof of Lemma 3.1. We consider only the case $r > 0$ and leave the verification for $r = 0$ to the reader. Let

$$k = \left\lceil 4r + n \left(\frac{m}{4n^2} \right)^{-1/(4r)} \right\rceil.$$

We may assume that $r \leqq n/8$ and $(m/(4n^2))^{1/(4r)} \geqq 3$ since otherwise what we are claiming is trivial. This allows us to also assume that n is large enough to make $n - f \geqq k$. Then

$$\frac{|A|}{n} \geqq \frac{m}{4n^2} \geqq \left(\frac{n}{k - 4r} \right)^{4r} \geqq \left(\frac{n}{k - \lfloor 4r \rfloor} \right)^{\lfloor 4r \rfloor}$$

$$\geqq \frac{[n-f]_{\lfloor 4r \rfloor}}{[k]_{\lfloor 4r \rfloor}} = \binom{n-f}{k} \Big/ \binom{n - f - \lfloor 4r \rfloor}{k - \lfloor 4r \rfloor},$$

and Claim 2 provides us with a subset W of $V \setminus B(v)$ with $|W| = k$ and $|A_W| \geqq n$. Choose Y'_R as any n-element subset of A_W. By construction, all valid copies of variables in Y'_R are contained in a total of $f + k$ processors. Hence the network cannot possibly

output copies of all variables in Y_R^t, i.e., simulate line (6), in fewer than $n/(f+k)$ cycles. But since

$$f+k \leqq \frac{\sqrt{n}}{\log n}+1+4r+n\left(\frac{m}{4n^2}\right)^{-1/(4r)}+1 = O\left(\max\left\{\frac{\sqrt{n}}{\log n}, r, n\left(\frac{m}{4n^2}\right)^{-1/(4r)}\right\}\right),$$

we have

$$\frac{n}{f+k} = \Omega\left(\min\left\{\sqrt{n}\log n, \frac{n}{r}, \left(\frac{m}{4n^2}\right)^{1/(4r)}\right\}\right). \qquad \square$$

We now continue the proof of Theorem 3 and first give a rule for choosing Y_R^t and Y_W^t. For $t=1,\cdots,\tau$, Y_R^t is chosen as in Lemma 3.1, and Y_W^t is chosen among all sets of n variables to maximize $\sum_{x\in Y_W^t} r_x^t$. Clearly $\sum_{x\in Y_W^t} r_x^t \geqq nr^t$. Note that this means that the writing part of stage t invalidates at least nr^t of those copies of variables which are counted in the total redundancy. On the other hand, writing such a copy costs $\Omega(q)=\Omega(\log n)$ atomic actions on the average, and each copy must be paid for separately by the assumption of point to point communication. To see this, consider a network processor u which writes a PRAM variable x in the tth stage, and let $t_1=\min(\{\tau+1\}\cup\{t'>t \mid x\in Y_W^{t'}\})$ (i.e., x is next updated in the t_1th stage). Assume that totalled over stages $t,\cdots,t_1-1$, exactly h valid copies of x leave $B(u)$. The cost of this is clearly $\Omega(hq)=\Omega(h\log n)$ by the assumption of point to point communication. On the other hand, $r_x^{t'}$, for $t<t'\leqq t_1$, can never exceed h since $r_x^{t'}$ by definition is the minimum number of valid copies of x outside any sphere $B(v)$, $v\in V$. Hence the average cost per counted copy is $\Omega(\log n)$.

Since $R^1=0$, $R^{\tau+1}\geqq 0$, and the total number of atomic actions performed by the network is at most Sn, we have the relation

$$\frac{Sn}{\log n}-\sum_{t=1}^{\tau} nr^t \geqq 0$$

or

$$S \geqq \tau\hat{r}\log n \tag{2}$$

where $\hat{r}=(1/\tau)\sum_{t=1}^{\tau} r^t$ is the value of r^t averaged over all stages.

Relation (2) expresses the cost of writing. In order to investigate the cost of reading as given by Lemma 3.1, we shall consider separately the various "cases" of the function g. Let

$$g_1(r)=\sqrt{n}\log n,$$

$$g_2(r)=\begin{cases} n/r & \text{if } r>0,\\ \infty & \text{if } r=0,\end{cases}$$

$$g_3(r)=\begin{cases} (m/(4n^2))^{1/(4r)} & \text{if } r>0,\\ \infty & \text{if } r=0,\end{cases}$$

and let $\{C_1, C_2, C_3\}$ be a partition of $\{1,\cdots,\tau\}$ such that for $i=1,2,3$,

$$\forall t\in C_i\colon\ g(r^t)=g_i(r^t).$$

Next choose $l\in\{1,2,3\}$ such that C_l is a biggest set in the partition, i.e., $|C_l|=\max_{1\leqq j\leqq 3}|C_j|$. Then there is a subset C' of C_l with

$$|C'|\geqq\frac{\tau}{6} \quad\text{and}\quad \forall t\in C'\colon\ r^t\leqq 6\hat{r}.$$

By Lemma 3.1 and (2), we have

$$S=\Omega\left(\max\left\{\tau\hat{r}\log n,\sum_{t\in C'}g_l(r^t)\right\}\right)=\Omega(T(\hat{r}\log n+g_l(6\hat{r}))).$$

Consider three cases depending on l. For $l\in\{1,2\}$, clearly

$$\min_{\hat{r}\geqq 0}(\hat{r}\log n+g_l(6\hat{r}))=\Omega(\sqrt{n\log n}).$$

As for the case $l=3$, observe first that $m=\Omega(n^{2+\varepsilon})$ implies $m/(4n^2)=\Omega(m^{\varepsilon/(2+\varepsilon)})$. Letting $\rho=\varepsilon\log m/(48(2+\varepsilon)\log\log m)$, we find

$$\min_{\hat{r}\geqq\rho}(\hat{r}\log n+g_3(6\hat{r}))\geqq\rho\log n=\Omega\left(\frac{\log n\log m}{\log\log m}\right)$$

and

$$\begin{aligned}\min_{0\leqq\hat{r}\leqq\rho}(\hat{r}\log n+g_3(6\hat{r}))\geqq g_3(6\rho)&=\left(\frac{m}{4n^2}\right)^{2(2+\varepsilon)\log\log m/(\varepsilon\log m)}\\&=\Omega((m^{\varepsilon/(2+\varepsilon)})^{(2+\varepsilon)2\log\log m/(\varepsilon\log m)})\\&=\Omega(2^{2\log\log m})=\Omega((\log m)^2)=\Omega\left(\frac{\log n\log m}{\log\log m}\right).\end{aligned}$$

Putting the various cases together finally yields

$$S=\Omega\left(T\min\left\{\sqrt{n\log n},\frac{\log n\log m}{\log\log m}\right\}\right).$$ □

Remark. The lower bound of Theorem 3 makes it natural to wonder whether a slowdown of close to $O(\sqrt{n})$ is achievable for arbitrary values of m. This is indeed the case. Assume for simplicity that n is a perfect square and consider the n emulating processors as arranged in a $\sqrt{n}$-by-$\sqrt{n}$ array. We use the following idea: When a PRAM processor P updates a variable x, its associated processor places a copy of x in each processor in its row (call this a *row operation*). When P wants to read the value of a variable x, the associated processor obtains a copy of x from each processor in its column which has a copy of x (a *column operation*). Clearly one of the copies will be valid, and it may be found using time stamps as in § 2.

It remains to determine the time needed for row and column operations. Observe that the procedure BROADCAST of § 2 actually performs a row operation if we consider each row as a cluster and execute a call BROADCAST($\sqrt{n}$). In the same way, a column operation is part of what happens in the execution of COUNT($\sqrt{n}$) with the columns considered as clusters. Hence if the emulating machine is an MPC, the slowdown is $O(\sqrt{n}+\log\sqrt{n})=O(\sqrt{n})$. Since COUNT and BROADCAST both use a fixed boundeded-degree interconnection pattern, namely a complete binary tree for each cluster, the same slowdown is attainable on a bounded-degree network. However, any such network must of course employ communication which is not point to point since otherwise it would violate the lower bound of Theorem 3 for sufficiently large values of m. On a bounded-degree network using only point to point communication, a slowdown of $O(\sqrt{n}\log n)$ is achievable. In order to see this, note that the necessary communication can be realized by a constant number of rounds, where in a round each processor sends at most one message to each processor in its row (or column). A round may clearly be implemented by $\sqrt{n}$ suitable permutations within each row

(column), e.g., by cyclic shifts by $1, 2, \cdots$, and finally by $\sqrt{n}$ positions, and each such permutation may be routed in $O(\log n)$ time if the processors within each row (column) are connected by, e.g., a cube-connected cycles network [PV].

4. An algorithm for approximate redistribution. This section establishes an upper bound on the parallel complexity of the strong redistribution problem. We also show how this result allows our simulation of PRAMs on MPCs to be somewhat simplified.

Our motivation for studying the redistribution problem is that it captures some aspects of a problem which is fundamental in parallel computing, namely that of distributing the work to be done evenly among the available processors in order to keep them all busy for as long as possible. We believe that the results and methods of this section may have applications outside of our simulation algorithm.

The main steps in the proof are as follows: We first show the existence of certain expander graphs and then indicate how these may be used to construct fast ε-halving procedures. This part of the proof parallels arguments in [AKS] except that we are being more explicit and want to correct a few inaccuracies. We then give an algorithm based on repeated ε-halving, bound its execution time and finally show that it solves the strong approximate data compaction problem.

Given an undirected graph $G=(V, E)$ and a set $U \subseteq V$ of nodes, let us write $\Gamma_G(U)$ or $\Gamma(U)$ for the set of nodes adjacent to a node in U, i.e., $\Gamma_G(U) = \{v \in V \mid (u, v) \in E \text{ for some } u \in U\}$.

DEFINITION. A bipartite graph on the node sets V_1 and V_2 is called an *expander graph* with parameters $(n_1, n_2, \lambda, \alpha, \nu)$ if $|V_1| = n_1$, $|V_2| = n_2$, and

(1) $\forall v \in V_1 \cup V_2$: $|\Gamma(\{v\})| \leqq \nu$,

(2) $\forall A \subseteq V_i$: $|A| \leqq \alpha n_{3-i} \Rightarrow |\Gamma(A)| \geqq \lambda |A|$, for $i = 1, 2$.

LEMMA 4.1. *For all $b > 0$, there is a constant K such that for all $\lambda \geqq 1$ ($\lambda \in \mathbb{R}$), $n \geqq 1$ and $\delta \in \{0, 1\}$ there exists an expander graph with parameters $(n+\delta, n, \lambda, 1/(\lambda+b), K\lambda^2)$.*

Proof. This is again a counting argument. We first consider the case $\delta = 0$. Let A and B be disjoint sets with $|A| = |B| = n$. Given ν bijections $\pi_1, \cdots, \pi_\nu$ from A to B, we may as in the proof of Lemma 2.1 construct a bipartite graph of maximum degree ν on the node sets A and B by drawing an edge from a to $\pi_l(a)$ for all $l = 1, \cdots, \nu$, $a \in A$. Let $\mathcal{G}_\nu$ denote the set of all such graphs and call a graph in $\mathcal{G}_\nu$ good if it is an expander graph with parameters $(n, n, \lambda, 1/(\lambda+b), \nu)$.

If a graph in $\mathcal{G}_\nu$ is not good, then there is a subset U of either A or B such that $1 \leqq |U| \leqq n/(\lambda+b)$ and $\Gamma(U) \subseteq V$ for some set V with $|V| = \lfloor \lambda |U| \rfloor$. For $z = 1, \cdots, \lfloor n/(\lambda+b) \rfloor$, let $f_{\nu,z}$ denote the fraction of graphs in $\mathcal{G}_\nu$ for which there is such a set U with $|U| = z$. As in the proofs of Lemmas 2.1 and 2.4 it is easy to see that (the factor 2 is due to the fact that one may choose $U \subseteq A$ or $U \subseteq B$)

$$f_{\nu,z} \leqq 2\binom{n}{z}\binom{n}{\lfloor \lambda z \rfloor}\left(\frac{[\lfloor \lambda z \rfloor]_z}{[n]_z}\right)^\nu \leqq 2\left(\frac{en}{z}\right)^z \left(\frac{en}{\lfloor \lambda z \rfloor}\right)^{\lfloor \lambda z \rfloor} \left(\frac{\lfloor \lambda z \rfloor}{n}\right)^{\nu z} \leqq 2\left[e^{\lambda+1}\frac{\lfloor \lambda z \rfloor^{\nu-\lambda}}{n^{\nu-\lambda-1}z}\right]^z.$$

Proceeding under the assumption $\nu \geqq \lambda + 1$, we further get

$$f_{\nu,z} \leqq 2\left[e^{\lambda+1}\left(\frac{\lambda z}{n}\right)^{\nu-\lambda-1}\lambda\right]^z \leqq 2\left[e^{\lambda+1}\left(\frac{\lambda}{\lambda+b}\right)^{\nu-\lambda-1}\lambda\right]^z,$$

where $z \leqq n/(\lambda+b)$ was used in the last step. Let

$$g_\nu = e^{\lambda+1}\left(\frac{\lambda}{\lambda+b}\right)^{\nu-\lambda-1}\lambda.$$

Then

$$\log g_\nu = (\lambda+1)\log e - (\nu-\lambda-1)\log\left(1+\frac{b}{\lambda}\right)+\log\lambda$$

$$\leqq (\lambda+1)\log e - \frac{(\nu-\lambda-1)}{\lambda}\log(1+b)+\log\lambda$$

which is $\leqq -2$ if

$$\nu \geqq \frac{\lambda(\lambda+1)\log e+\lambda\log\lambda+2\lambda}{\log(1+b)}+\lambda+1.$$

Hence if $\nu \geqq K\lambda^2$ for a suitable constant K depending only on b, we have

$$g_\nu \leqq \tfrac{1}{4}$$

and hence

$$f_{\nu,z} \leqq \frac{2}{4^z}.$$

The total fraction of bad graphs in $\mathcal{G}_\nu$ is then bounded by

$$\sum_{z=1}^{\lfloor n/(\lambda+b)\rfloor} f_{\nu,z} \leqq \sum_{z=1}^{\infty}\frac{2}{4^z} < 1$$

which proves the lemma for the case $\delta = 0$.

We now turn to the case $\delta = 1$. Suppose that we must produce an expander graph with parameters $(n+1, n, \lambda, 1/(\lambda+b), K\lambda^2)$ for given b, λ and n. First take an expander graph G with node sets A and B and parameters $(n+1, n+1, \lambda+b/2, 1/(\lambda+b), K\lambda^2)$ which exists by the proof for the case $\delta = 0$ for some constant K depending only on b. We may assume (increasing K as necessary) that extra edges have been added to achieve that each node in G has at least $2\lambda/b$ incident edges.

Now remove one node $v \in B$ and its incident edges. We claim that the resulting graph G' is an expander graph with parameters $(n+1, n, \lambda, 1/(\lambda+b), K\lambda^2)$. We must show that a sufficiently small subset U of either A or $B\backslash\{v\}$ has $|\Gamma_{G'}(U)| \geqq \lambda|U|$. Since this is trivial for $U \subseteq B$, we may assume that $U \subseteq A$, $1 \leqq |U| \leqq n/(\lambda+b)$. Then

$$|\Gamma_{G'}(U)| \geqq |\Gamma_G(U)| - 1 \geqq \left(\lambda+\frac{b}{2}\right)|U| - 1,$$

and if $|U| \geqq 2/b$, this allows the desired conclusion $|\Gamma_{G'}(U)| \geqq \lambda|U|$. On the other hand, if $|U| \leqq 2/b$, then an arbitrary node u in U by construction has a sufficient number of neighbors, i.e.,

$$|\Gamma_{G'}(U)| \geqq |\Gamma_{G'}(\{u\})| \geqq \frac{2}{b}\lambda \geqq \lambda|U|. \qquad \square$$

DEFINITION. A permutation π of $\{1, \cdots, n\}$ is said to be *ε-halved* if

(1) $\forall l \in \{1, \cdots, \lceil n/2\rceil\}$: $|\{i \mid 1 \leqq i \leqq l \text{ and } \pi(i) > \lceil n/2\rceil\}| \leqq \varepsilon l$;

(2) $\forall l \in \{\lceil n/2\rceil+1, \cdots, n\}$: $|\{i \mid l \leqq i \leqq n \text{ and } \pi(i) \leqq \lceil n/2\rceil\}| \leqq \varepsilon(n-l+1)$.

Consider again n locations arranged in a fixed order and each containing a record with a key. The configuration may be said to represent a permutation π of $\{1, \cdots, n\}$ if the record stored in the $\pi(i)$th location, for $i = 1, \cdots, n$, contains the key i. A procedure is called an ε-halver on n elements if it takes as input a configuration

representing an arbitrary permutation of $\{1, \cdots, n\}$ and produces as output a configuration representing an ε-halved permutation. An ε-halver thus approximately separates the small and the big keys.

LEMMA 4.2. *There is a constant K such that for all $n \geqq 1$ and all $\varepsilon > 0$, there exists an ε-halver on n elements consisting of at most $K(1/\varepsilon)^2$ parallel comparison-exchange steps on n elements.*

Proof. Assume without loss of generality that $\varepsilon \leqq \frac{1}{2}$. We consider two cases.

Case 1: $n \leqq 8/\varepsilon^2$. Here Batcher's well-known construction [B] gives us not only an ε-halver, but actually a sorting algorithm. For some constant K', it consists of at most $K'(\log n)^2 \leqq 25K'(\log(1/\varepsilon))^2 \leqq 25K'(1/\varepsilon)^2$ parallel comparison-exchange steps.

Case 2: $n > 8/\varepsilon^2$. Now let $\lambda = (1-\varepsilon)/\varepsilon$ and consider an ordered sequence $L_1, \cdots, L_n$ of n locations whose contents represent a permutation of $\{1, \cdots, n\}$. We assume that each record consists only of its key.

By Lemma 4.1, there is an expander graph G with parameters $(\lceil n/2 \rceil, \lfloor n/2 \rfloor, \lambda, 1/(\lambda + \frac{1}{2}), K\lambda^2)$ for some constant K. As in the proof of Lemma 2.1 we may identify the node sets of G with $A = \{L_1, \cdots, L_{\lceil n/2 \rceil}\}$ and $B = \{L_{\lceil n/2 \rceil + 1}, \cdots, L_n\}$ and consider each edge (L_i, L_j) as a comparison-exchange operation to take place between L_i and L_j. Since no node in G has more than $K\lambda^2$ incident edges, it is possible to carry out all comparison-exchange operations in at most $K\lambda^2 \leqq K(1/\varepsilon)^2$ parallel comparison-exchange steps (this is true without additional assumptions on G; see [O, Thm. 7.5.6]. However, the reader may also imagine G to have been constructed as in the proof of Lemma 4.1, in which case the claim becomes trivial).

Suppose that the resulting algorithm $\mathcal{A}$ is not an ε-halver. We will show that this leads to a contradiction. Assume without loss of generality (the other situation being similar) that for some $l \in \{1, \cdots, \lceil n/2 \rceil\}$, more than εl keys $\leqq l$ are in locations in B after the execution of $\mathcal{A}$. Call a key "small" exactly if it is $\leqq l$ and let B', $|B'| = \lceil \varepsilon l \rceil$, be a set of locations in B containing only small keys after the execution of $\mathcal{A}$. Then

$$|B'| \leqq \lceil \varepsilon \lceil n/2 \rceil \rceil \leqq \varepsilon \lfloor n/2 \rfloor + 2 \leqq \frac{\lfloor n/2 \rfloor}{\lambda + \frac{1}{2}},$$

where the last inequality may be shown correct by elementary manipulations using the relation $n > 8/\varepsilon^2$ and the definition of λ. But this means that B' is small enough to be "expanded" by G, i.e.,

$$|\Gamma(B')| \geqq \lambda |B'| \geqq (1-\varepsilon) l.$$

Since more than εl small keys are in locations in B after the execution of $\mathcal{A}$, fewer than $l - \varepsilon l = (1-\varepsilon) l$ are in locations in A. But then some location in B' is necessarily linked by an edge to a location in A containing a bigger key. As in the proof of Lemma 2.1, this is impossible. Hence $\mathcal{A}$ is an *ε-halver.* □

THEOREM 4. *There exists a constant K such that for all n, R, and y with $1 \leqq y < R \leqq n$, the strong approximate data compaction (or redistribution) problem with n elements, upper bound R, and y allowable errors is solved by an algorithm consisting of at most*

$$K\left(\frac{R}{y}\right)^2 \left\lceil \log \frac{n}{y} \right\rceil^3$$

parallel comparison-exchange steps on n elements.

Proof. If $y = 1$, we may use Batcher's sorting network. Hence assume that $y \geqq 2$. Let the locations $L_1, \cdots, L_n$ contain the records to be permuted, and assume as usual that each record is just a one-bit key. We will use repeated ε-halving, first on the whole set $L_1, \cdots, L_n$ of locations, then on its left and right halves, etc., until the pieces become smaller than y. More formally, when $I = \{a, \cdots, b\}$ is a finite set of consecutive

integers (call this an *interval*), we denote by I_l and I_r its left and right halves, i.e.,

$$I_l = \left\{a, \cdots, \left\lfloor \frac{a+b}{2} \right\rfloor\right\}, \qquad I_r = \left\{\left\lfloor \frac{a+b}{2} \right\rfloor + 1, \cdots, b\right\}.$$

We will also use the notation $L_I = \{L_i \mid i \in I\}$. Let $S_0 = \{\{1, \cdots, n\}\}$, and for $t = 1, 2, \cdots$, let

$$S_t = \{J \mid J = I_l \text{ or } J = I_r \text{ for some } I \in S_{t-1}\}.$$

Choose T minimal such that $|I| \leqq y$ for all $I \in S_T$ and let

$$\varepsilon = \frac{y}{3R(T+1)}.$$

The algorithm for approximate data compaction now is

```
(1) for t := 0 to T do
(2)   for all I ∈ S_t pardo (* do in parallel *)
(3)     Run an ε-halver on L_I;
(4)   odpar;
(5) od;
```

We call an execution of line (3) an ε-halving step. Each L_I should of course be considered ordered with $L_i < L_j \Leftrightarrow i < j$.

According to Lemma 4.2, the above algorithm may be implemented by $(T+1)K'(1/\varepsilon)^2$ parallel comparison-exchange steps for some constant K'. Since $T+1 \leqq \log(n/y)+2 \leqq 3\lceil \log(n/y) \rceil$ and hence $1/\varepsilon \leqq 9(R/y)\lceil \log(n/y) \rceil$, this is less than

$$K\left(\frac{R}{y}\right)^2 \left\lceil \log \frac{n}{y} \right\rceil^3$$

for a suitable constant K.

It remains only to show that the algorithm is correct, i.e., solves the strong approximate data compaction problem. We need a few definitions. First, denote by Q the total number of keys equal to 0. Then for $t = 0, \cdots, T+1$ and $I \in S_t$, let $\mu_t(I)$ be the number of 0's in (locations in) L_I after t ε-halving steps, let $\hat{\mu}(I)$ be the number of 0's that would result from a perfect data compaction, and let $\phi_t(I)$ be the "surplus" of 0's. More precisely,

$$\mu_t(I) = |\{i \in I \mid L_i \text{ contains a 0 after exactly } t \text{ } \varepsilon\text{-halving steps}\}|,$$

$$\hat{\mu}(I) = |I \cap \{1, \cdots, Q\}|, \text{ and}$$

$$\phi_t(I) = \max\{\mu_t(I) - \hat{\mu}(I), 0\}.$$

A global measure of the error accumulated after t ε-halving steps is given by

$$\Phi(t) = \sum_{I \in S_t} \phi_t(I).$$

We claim that Φ increases by at most εQ in a given ε-halving step, say the tth. Let $I \in S_{t-1}$. First it is obvious that if $Q \notin I$, then $\phi_t(I_l) + \phi_t(I_r) = \phi_{t-1}(I)$. Hence we need only consider the unique interval $I \in S_{t-1}$ with $Q \in I$ and show that

$$\phi_t(I_l) + \phi_t(I_r) \leqq \phi_{t-1}(I) + \varepsilon Q.$$

Note that ε-halvers were designed to sort permutations of $\{1, \cdots, n\}$. Since we consider algorithms consisting exclusively of comparison-exchange steps, ε-halving may also be applied to sequences of 0's and 1's. In order to deduce what happens, it is useful to imagine each one-bit key to have been augmented by a unique suffix and the comparisons to take place using the lexicographic order on the augmented keys. This would not change the resulting distribution of zero keys. Hence one may think of each

key as belonging in one particular position in the sorted sequence. It is now easy to see that the distribution of 0's between L_{I_l} and L_{I_r} achieved by any ε-halving of L_I may be obtained from the distribution that would result from a perfect sorting of L_I by moving at most εq 0's from L_{I_l} to L_{I_r} or vice versa, where q is the number of 0's in L_I. Furthermore, this would change $\phi_t(I_l)+\phi_t(I_r)$ by at most εq. On the other hand, if the tth ε-halving step happens to sort L_I perfectly, then $\phi_t(I_l)+\phi_t(I_r)=\phi_{t-1}(I)$. Hence

$$\phi_t(I_l)+\phi_t(I_r)\leqq\phi_{t-1}(I)+\varepsilon q\leqq\phi_{t-1}(I)+\varepsilon Q,$$

proving our claim.

Now for $t=0,\cdots,T+1$, let y_t be the number of errors in the configuration obtained after t ε-halving steps, i.e.,

$$y_t=\{i\in\mathbb{N}\,|\,Q<i\leqq n \text{ and } L_i \text{ contains a 0 after exactly } t \text{ } \varepsilon\text{-halving steps}\}.$$

If $I\in S_t$ is the interval containing Q, and $J\in S_t$ is an interval to the right of I (i.e., $i<j$ for all $i\in I, j\in J$), then $\phi_t(J)=\mu_t(J)$. Hence $y_t\leqq\Phi(t)+|I|$. In particular,

$$y_{T+1}\leqq\Phi(T+1)+\left\lceil\frac{y}{2}\right\rceil\leqq(T+1)\varepsilon Q+\left\lceil\frac{y}{2}\right\rceil\leqq(T+1)\varepsilon R+\left\lceil\frac{y}{2}\right\rceil\leqq\frac{y}{3}+\left\lceil\frac{y}{2}\right\rceil\leqq y. \qquad \square$$

Consider now the implications on the simulation algorithm in § 2. Armed with the result of Theorem 4, we are no longer constrained to run Part 1 of the simulation for just $O(\log\log m)$ stages. In fact, suppose that we increase the number of stages in Part 1 to $v=\lceil(\log m)^{1/4}\rceil$. Then the total time spent in calls of APPROXIMATE-REDISTRIBUTE is

$$\sum_{i=1}^{v}S_i=O\left(\sum_{i=1}^{v}(\log 2^i)^3\right)=O\left(\sum_{i=1}^{v}i^3\right)=O(v^4)=O(\log m),$$

and the same bound holds for the time spent in the remainder of Part 1. The number of live variables is reduced to at most

$$n2^{-v}\leqq n2^{-(\log m)^{1/4}}.$$

This suffices for an input to Part 3 of the simulation, as may be seen by reviewing the proof of Lemma 2.5. Hence Theorem 4 allows us to simplify the simulation algorithm by removing Part 2.

5. Conclusion. In this paper we achieved an upper bound of $O(\log m)$ on the slowdown incurred by an n-processor MPC emulating the shared memory of an n-processor PRAM with m shared memory cells. Our scheme leaves open a number of important problems.

The most troublesome unsolved problem is that of efficient (polynomial-time) construction of good MOS's. This computation needs to be carried out only once for given values of n and m. However, we know no method for doing this except systematically generating one MOS after another and testing "goodness" by brute force, and this is not a feasible approach for interesting values of n and m like $n=10^3$ and $m=10^6$.

Finding a polynomial-time construction algorithm appears to be very difficult because of the close connection to the problem of efficient construction of expander graphs. A polynomial-time algorithm was discovered for the latter problem after a considerable effort [GG]; the MOS construction problem, however, seems more difficult (cf. Lemma 2.4). In fact, the graph should maintain the expanding property even when an adversary is allowed to eliminate all but c^q of the $2c-1$ edges incident on each node representing a PRAM variable.

We assume, as do Upfal and Wigderson, that each MPC processor, when presented with the name of a variable x, is able to tell which of the n memory modules contains "its" copy of x (i.e., $\psi_l(x)$ if the processor is the lth in its cluster). If this mapping has no regularity to be taken advantage of, we need $\Omega(m \log n)$ extra bits per processor to store it in the form of a table, a very high overhead. Therefore it is important to find a good MOS with a short description.

There are two ways in which one might want to strengthen the lower bound of Theorem 3. First, we would like to get rid of the factor $\log \log m$ in the lower bound in order to obtain matching upper and lower bounds. In order to do this, it seems to be necessary to refine the analysis to not only count the routing steps necessary to bring copies to where they are needed, but also somehow take into account the difficulty of deciding where a valid copy may be found. Note that our emulation algorithm realizes a read instruction by accessing many copies and using time stamps to find a valid copy.

The other obvious improvement would be to obviate the need for the assumption of point to point communication. Even without this assumption, the cost of setting up r copies of some variable is at least proportional to the number of edges in a smallest Steiner tree spanning the processors containing the r copies. We believe that if the processors containing the copies are chosen to often make this quantity significantly smaller than $r \log n$, i.e., writing is cheap, then the copies necessarily cluster, in a sense still to be appropriately defined, and reading must be expensive.

REFERENCES

[A] L. ADLEMAN, *Two theorems on random polynomial time*, Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 75-83.

[AKS] M. AJTAI, J. KOMLÓS AND E. SZEMERÉDI, *An $O(n \log n)$ sorting network*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 1-9.

[B] K. E. BATCHER, *Sorting networks and their applications*, Proc. AFIPS Spring Joint Computer Conference, 32 (1968), pp. 307-314.

[BH] A. BORODIN AND J. E. HOPCROFT, *Routing, merging, and sorting on parallel models of computation*, Proc. 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 338-344.

[FW] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 114-118.

[GG] O. GABBER AND Z. GALIL, *Explicit constructions of linear size concentrators and superconcentrators*, Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, 1979, pp. 364-370.

[K] D. J. KUCK, *A survey of parallel machine organization and programming*, Comput. Surveys, 9 (1977), pp. 29-59.

[KU] A. R. KARLIN AND E. UPFAL, *Parallel hashing—an efficient implementation of shared memory*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 160-168.

[L] T. LEIGHTON, *Tight bounds on the complexity of parallel sorting*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 71-80.

[MV] K. MEHLHORN AND U. VISHKIN, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339-374.

[O] O. ORE, *Theory of Graphs*, American Mathematical Society, Providence, RI, 1962.

[PV] F. P. PREPARATA AND J. VUILLEMIN, *The cube-connected cycles: A versatile network for parallel computation*, Comm. ACM, 24 (1981), pp. 300-309.

[R] J. H. REIF, *On the power of probabilistic choice in synchronous parallel computations*, Proc. 9th Internat. Coll. Automata, Languages and Programming, 1982, pp. 442-450.

[U] E. UPFAL, *A probabilistic relation between desirable and feasible models of parallel computation*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 258-265.

[UW] E. UPFAL AND A. WIGDERSON, *How to share memory in a distributed system*, Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, 1984, pp. 171-180.

[VW] U. VISHKIN AND A. WIGDERSON, *Dynamic parallel memories*, Inform. and Control, 56 (1983), pp. 174-182.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987
© 1987 Society for Industrial and Applied Mathematics

A DNF WITHOUT REGULAR SHORTEST CONSENSUS PATH*

WENQI HUANG† AND XIANGDONG YU†

Abstract. A disjunctive normal form without regular shortest consensus path is given. The question Tseitin asked in 1968 and Galil mentioned again in 1977 is answered.

Key words. DNF, resolution, consensus

AMS(MOS) subject classification. 68Q15

As the dual of the resolution method, the consensus method can be used to determine the validity of DNF's. This method appeared over 20 years ago. But several problems on its complexity persisted for a long time. In 1968, Tseitin set a regular restriction on consensus paths, and showed that it takes steps amounting to an exponential function of the square root of the input length to determine the validity of DNF's by using the regular consensus method even most effectively. He posed in the same paper such a question: "For any valid DNF, is it necessary that some of its shortest consensus paths generating Λ are regular?" Obviously, a positive answer to this question would lead immediately to the conclusion that the examples in [4] would also be hard examples for the unrestricted consensus method. But he did not answer the question. Galil improved Tseitin's proof and result in 1977, yet still listed that question as an open one. Even when A. Haken and A. Urquhart sequentially proved the hard examples for the consensus method last year, they did not figure out the answer to Tseitin's question. In this paper we explore it and answer it negatively by giving a DNF as a counterexample.

The DNF is:

$$F = pab \vee \bar{p}ab \vee \bar{a}c \vee \bar{b}d \vee \bar{k}p \vee \bar{h}\bar{p} \vee \bar{c}kp \vee \bar{d}kp \vee \bar{c}h\bar{p} \vee \bar{d}h\bar{p}.$$

We claim that the consensus procedure represented by Fig. 1 is shorter than any regular consensus procedure. It contains only 10 consensus steps.

If two clauses of a DNF produce by consensus a new clause (not appearing in the present formula), then we call the operation adding the new one to the formula a consensus step (or a STEP). A series of steps is called a consensus procedure, or a PATH. In this paper, digraphs such as Fig. 1 will be used to represent paths.

More concretely, when a path from a DNF is given, we can construct the digraph as below. Take every clause in the DNF as a node, then, corresponding to each step, introduce a new node labeled with the new clause just generated, and draw an arrow to it from each of the two nodes (or clauses) generating it.

Hence, the indegrees of the original nodes are all 0. Call the original nodes LEAVES. The indegrees of the new nodes are all 2. Call the new nodes IN-NODES. Different nodes denote different clauses. No directed cycle exists in the digraph (because every arrow is introduced to point to a new node). If a digraph D denotes a path, and E is one of its clauses (or nodes), then D_E denotes such a subgraph consisting of all

* Received by the editors October 15, 1985; accepted for publication (in revised form) November 24, 1986. This work was supported by the Science Fund of the Chinese Academy of Science.

† Department of Computer Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei, China.

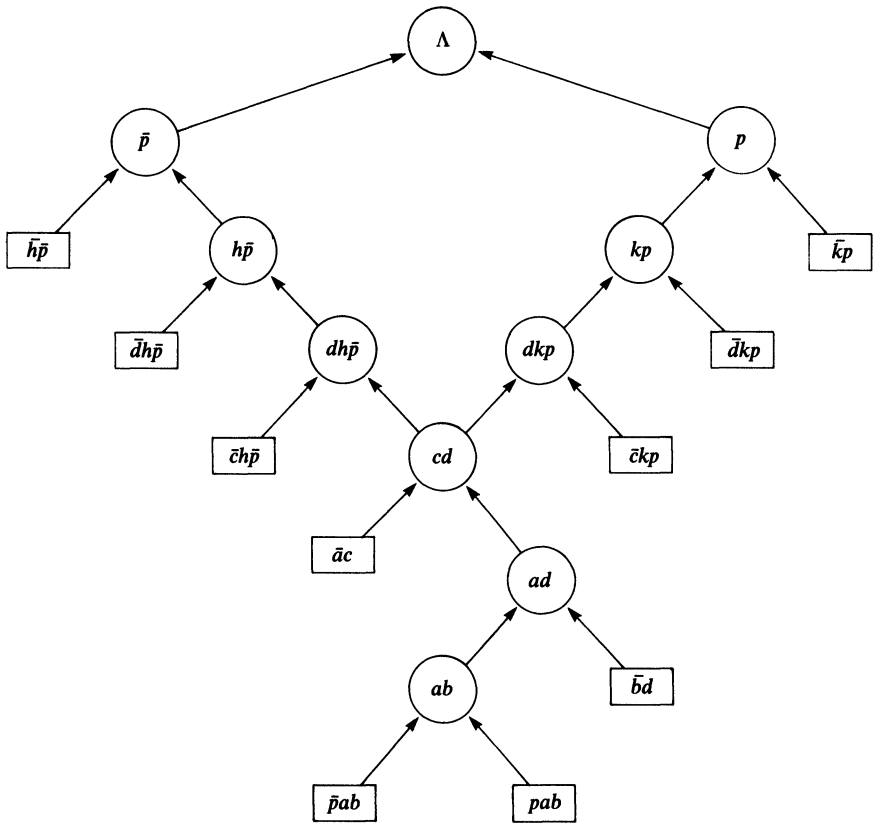


FIG. 1

nodes from which E can be arrived at and all arrows pointing to these nodes. Any node g in D_E is called a DESCENDANT of E, written as $g \in D_E$. If two nodes produce a node f by annihilating variable x, then we say that f ANNIHILATES x.

Define $N(D)$ as the number of in-nodes in digraph D. Consider D_E to be the path generating E, then $N(D_E)$ is its length. Since Λ can be generated only through some pair $(\bar{x}, x)$, it is sufficient to study the procedure generating $\bar{x}$ and x for any variable x. In the following we use $D_{(\bar{x},x)}$ to denote the digraph corresponding to one of the shortest regular paths generating $\bar{x}$ and x from F. Use $D_{\bar{x}}$ and D_x to denote the paths in $D_{(\bar{x},x)}$ generating $\bar{x}$ and x, respectively. Then

$$N(D_{(\bar{x},x)}) = N(D_{\bar{x}}) + N(D_x) - N(D_{\bar{x}} \cap D_x).$$

DEFINITION. A consensus procedure is regular if and only if its digraph does not contain a node E such that a descendant of E annihilates a variable in E.

In Fig. 1, $ab \in D_p$, but ab annihilates variable p, so the procedure corresponding to the figure is not regular.

We are going to prove that any regular path generating $(\bar{p}, p)$ or $(\bar{a}, a)$ or $(\bar{c}, c)$ or $(\bar{k}, k)$ contains at least 10 steps, i.e., $N(D_{(\bar{p},p)}) \geqq 10$, etc. Because of the symmetry between p, a, c, k and $\bar{p}, b, d, h$ in F, we may easily see that it is also the case for $(\bar{b}, b)$, $(\bar{d}, d)$ and $(\bar{h}, h)$. Then, any regular path generating Λ has length at least $10+1=11$, longer than that showed by Fig. 1. Thus Tseitin's question is answered negatively.

Let us show three lemmas as preparation.

LEMMA 1. *Let x be a variable in F. Suppose D_x has n leaves, and $d(E)$ denotes the outdegree of node E in D_x. Clearly, $d(E) \geqq 1$ for $E \neq x$. So,*

$$K = \sum_{\substack{E \in D_x \\ E \neq x}} (d(E) - 1) \geqq 0.$$

We have $N(D_x) = n + K - 1 \geqq n - 1$. An analogous conclusion is true for $\bar{x}$.

Proof. The total number of the arrows in D_x is

$$\sum_{\substack{E \in D_x \\ E \neq x}} d(E).$$

Each in-node absorbs two arrows and different in-nodes absorb different arrows. Thus

$$\sum_{\substack{E \in D_x \\ E \neq x}} d(E) = 2N(D_x).$$

So,

$$K = \sum_{\substack{E \in D_x \\ E \neq x}} d(E) - (N(D_x) + n - 1)$$

$$= 2N(D_x) - N(D_x) - (n - 1).$$

Then, $N(D_x) = n + K - 1 \geqq n - 1$. □

LEMMA 2. *Write F as $\bar{x}F_1 \vee xF_2 \vee F_3$, where F_1, F_2, F_3 are DNF's not involving x. Then, in $D_{(\bar{x},x)}$, no clause in xF_2 is $\bar{x}$'s descendant and no clause in $\bar{x}F_1$ is x's descendant.*

Proof. Assume some clause in xF_2 were $\bar{x}$'s descendant. According to regularity, the road leading to $\bar{x}$ from this node would not contain a node annihilating x, so the x in this node would be kept along the road until arriving at node $\bar{x}$. That is to say, $\bar{x}$ would contain x as a literal. This is a contradiction!

So $\bar{x}$ does not contain any xF_2's clause as its descendant. A similar proof applies to x. □

LEMMA 3. *Write F as $F_1 \vee F_2$, where F_1, F_2 are DNF's. Consider $D_{(\bar{x},x)}$, suppose $F_2 \subseteq D_x - D_{\bar{x}}$ (a subgraph produced from D_x by taking away nodes and arrows belonging to $D_{\bar{x}}$) and F_2 involves t variables different from x. Then, $N(D_x) - N(D_{\bar{x}} \cap D_x) \geqq t$.*

The lemma will be also true if we exchange the positions of $\bar{x}$ and x.

Proof. All variables in F_2 except x have to be annihilated somewhere when leading to node x from leaves in F_2. Notice the fact that any node annihilating some variable of F_2 (that is to say, the node annihilates the variable, and there is a road leading to the node from a F_2's node which involves the variable) cannot be a descendant of $\bar{x}$, or there would be a descendant of $\bar{x}$ in F_2, contradictory to the condition $F_2 \subseteq D_x - D_{\bar{x}}$. So, all nodes annihilating F_2's variables only appear in D_x. On the other hand, there are at least t such nodes; therefore $N(D_x) - N(D_{\bar{x}} \cap D_x) \geqq t$. □

When Λ is to be generated from a valid DNF, some clauses cannot be omitted, since the formula would no longer be valid without them. Similarly, when p, or a, c, etc. is to be generated from F, some clauses cannot be omitted. For example, $\bar{a}c$ is necessary for p. If it is taken away, we may find an assignment which guarantees $p = 1$ and makes the new formula 0. This formula is not able to generate p.

By the principle above, we will determine which clauses are necessary for a given literal. We give the assignment guaranteeing the literal 1 and the incomplete formula 0 in the following list.

Plain calculation illustrates that none of F's 10 clauses may be omitted to keep its validity.

When generating a literal x corresponding to some row, according to Lemma 2, those clauses marked ▨ never appear in D_x. (See Table 1.)

With the aid of our earlier lemmas, we are able to estimate $N(D_{(\bar{x},x)})$.

(1) $(\bar{p}, p)$. From the list we know, the leaves of $D_{\bar{p}}$ are:

$$\bar{p}ab,\ \bar{p}\bar{c}h,\ \bar{p}\bar{d}h,\ \bar{p}\bar{h},\ \bar{a}c,\ \bar{b}d.$$

Therefore, $N(D_{\bar{p}}) \geqq 6-1=5$. Similarly $N(D_p) \geqq 5$.

According to the above list, all in-nodes in $D_{\bar{p}} \cap D_p$ should come from $\bar{a}c$, $\bar{b}d$ only. But $\bar{a}c$, $\bar{b}d$ are not complemented to each other, so, $N(D_{\bar{p}} \cap D_p) = 0$. Thus, $N(D_{(\bar{p},p)}) = N(D_{\bar{p}}) + N(D_p) \geqq 10$.

(2) $(\bar{a}, a)$. The following must be leaves of D_a:

$$pab,\ \bar{p}ab,\ \bar{b}d,\ \bar{d}kp,\ \bar{d}h\bar{p},\ \bar{k}p,\ \bar{h}\bar{p}.$$

Hence, $N(D_a) \geqq 7-1=6$.

The list says, $\bar{a}c \notin D_a$. $\bar{a}c$ is the only clause containing literal c, so D_a does not involve literal $\bar{c}$, either, i.e., $\bar{c}kp$, $\bar{c}h\bar{p} \notin D_a$. This fact implies that $\bar{a}c$, $\bar{c}kp$, $\bar{c}h\bar{p} \in D_{\bar{a}}$. They involve 4 variables, c, k, h, p besides a. According to Lemma 3, we have

$$N(D_{\bar{a}}) - N(D_{\bar{a}} \cap D_a) \geqq 4,$$

$$N(D_{(\bar{a},a)}) = N(D_a) + N(D_{\bar{a}}) - N(D_{\bar{a}} \cap D_a) \geqq 6+4=10.$$

(3) $(\bar{c}, c)$. D_c's leaves are

$$pab,\ \bar{p}ab,\ \bar{a}c,\ \bar{b}d,\ \bar{d}kp,\ \bar{k}p,\ \bar{d}h\bar{p},\ \bar{h}\bar{p},$$

$$N(D_c) \geqq 8-1=7.$$

TABLE 1

Assignments \ Clauses in F		pab	$\bar{p}ab$	$\bar{a}c$	$\bar{b}d$	$\bar{c}kp$	$\bar{d}kp$	$\bar{k}p$	$\bar{c}h\bar{p}$	$\bar{d}h\bar{p}$	$\bar{h}\bar{p}$
Literals (truth values are listed in the order *pabcdkh*)	p	111 1111	▨	101 1111	110 1111	101 0111	110 1011	110 1000	▨	▨	▨
	$\bar{p}$	▨	011 1111	001 1111	010 1111	▨	▨	▨	001 0111	010 1011	010 1000
	a	111 1111	011 1111	▨	110 1111		110 1011	110 1000		010 1011	010 1000
	$\bar{a}$	▨	▨								
	c	111 1111	011 1111	101 1111	110 1111	▨	110 1011	110 1000	▨	010 1011	010 1000
	$\bar{c}$			▨							
	k							▨			
	$\bar{k}$		011 1111	001 1101	010 1101	▨	▨	110 1000	001 0101	010 1001	010 1000

$D_{\bar{c}} - D_c$ contains $\bar{c}kp$, $\bar{c}h\bar{p}$ involving 3 variables apart from c. According to Lemma 3, we have

$$N(D_{(\bar{c},c)}) = N(D_c) + N(D_{\bar{c}}) - N(D_{\bar{c}} \cap D_c) \geqq 7 + 3 = 10.$$

(4) $(\bar{k}, k)$. $D_{\bar{k}}$ and $D_k - D_{\bar{k}}$ contain $\bar{p}ab$, $\bar{a}c$, $\bar{b}d$, $\bar{k}p$, $\bar{c}h\bar{p}$, $\bar{d}h\bar{p}$, $\bar{h}\bar{p}$ and $\bar{c}kp$, $\bar{d}kp$, respectively.

There are two choices for pab:

(i) $pab \in D_{\bar{k}}$.

$$N(D_{\bar{k}}) \geqq 7, \qquad N(D_k) - N(D_{\bar{k}} \cap D_k) \geqq 3,$$

hence, $N(D_{(\bar{k},k)}) \geqq 10$.

(ii) $pab \in D_k - D_{\bar{k}}$.

$$N(D_{\bar{k}}) \geqq 6, \qquad N(D_k) - N(D_{\bar{k}} \cap D_k) \geqq 5,$$

hence, $N(D_{(\bar{k},k)}) \geqq 11$.

No matter what the choice is, $N(D_{(\bar{k},k)}) \geqq 10$ is always true.

Acknowledgments. We wish to thank Mr. Lai Susheng and Mr. Chen Zhixiang for helpful discussions, and Professor Zvi Galil for kind encouragement, apropos advice and references.

REFERENCES

[1] Z. GALIL, *On the complexity of regular resolution and Davis-Putnam procedure*, Theoret. Comput. Sci., 4 (1977), pp. 23-46.

[2] A. HAKEN, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297-308.

[3] J. A. ROBINSON, *A machine oriented logic based on the resolution principle*, J. Assoc. Comput. Mach., 12 (1965), pp. 23-41.

[4] G. S. TSEITIN, *On the complexity of derivation in the propositional calculus*, in Structures in Constructive Mathematical Logic, Part II, O. A. Slisenko, ed., 1968, pp. 115-125 (translated from Russian).

[5] A. URQUHART, *Hard Examples for Resolution*, to appear.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987

© 1987 Society for Industrial and Applied Mathematics
004

INDEPENDENT AND SEPARABLE DATABASE SCHEMES*

EDWARD P. F. CHAN†‡ AND ALBERTO O. MENDELZON†

Abstract. We propose and investigate the notion of separability to capture the design goal of independently updatable decompositions. We characterize separable schemes in the important case when the only constraints given are a set of functional dependencies and the join dependency $\bowtie\mathbf{R}$. This characterization is also applicable to cover embedding database schemes when a set of functional dependencies is given as constraint. As evidence in favor of separability as a natural concept of independence, we show that it is equivalent to a specialization of the abstract independent mappings defined by Bancilhon and Spyratos. Our characterization yields a polynomial-time algorithm for testing separability in these cases.

Key words. scheme design, the weak instance model, relational database, functional and join dependencies, independent updatable components

AMS(MOS) subject classification. H21

1. Introduction. Independent updatability of relations has long been recognized as an important goal in the design of a relational database scheme. For example, in a distributed database system, we would like to be able to update a relation stored at some node of the network without having to worry about the effect of this update on the integrity of relations stored at other nodes.

The work of Rissanen [14] and its generalization [5], [13] studied the properties of *lossless join* and *cover embedding* and their connection with independence. Later Vardi [17] examined independence under very general constraints. However, all this work deals only with *join-consistent* database states, that is, states that are projections of a common universal relation. In order to apply the results mentioned above, one has to ensure that the join-consistency assumption is maintained; this is a strong global constraint which goes against the goal of independent updatability.

Sagiv [15], [16], Graham [8], and Graham and Yannakakis [10] go a step further in using Honeyman's notion of *weak instance satisfaction* [11] to propose a new model of independence. In this context, a database state is said to satisfy a set of dependencies if there exists a satisfying universal relation whose projection contains each relation in the state. States that are not join-consistent are thus acceptable.

A database scheme is *independent* if local satisfaction of the projected dependencies at each relation guarantees weak instance satisfaction of the state as a whole. However, independence only goes so far in ensuring independent updatability, as is illustrated by the following example.

Example 1. $\mathbf{R} = \{R_1(Professor, Student), R_2(Student, Topic), R_3(Professor, Topic)\}$. $F = \{Student \rightarrow Topic\}$.

Consider the following database state.

Professor	*Student*	*Student*	*Topic*	*Professor*	*Topic*
Kant	John	John	Being	Kant	Being
Kant	Paul				

Intuitively, a professor is associated with a topic if he/she has a student pursuing that topic, and each student pursues a unique topic. Then inserting the tuple ⟨Paul,

* Received by the editors November 25, 1983; accepted for publication December 3, 1986. This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada.

† Computer Systems Research Institute, University of Toronto, Toronto, Canada M5S 1A4.

‡ Present address, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1.

Existence⟩ into the R_2 relation does not affect local satisfaction of the functional dependency *Student* → *Topic* in R_2. In fact, since this database scheme is independent, this insertion does not affect global satisfaction. However, the tuple ⟨Kant, Existence⟩ must be inserted in R_3 if we want the intuitive meaning of the relations to be preserved.

Similarly, if we delete the tuple ⟨John, Being⟩ from r_2, we would expect "Kant" is no longer associated with the topic "Being." Hence we should delete the tuple ⟨Kant, Being⟩ from r_3. Therefore although **R** is independent, it does not guarantee independent updatability.

The key to this problem is the distinction between *consistency* and *completeness* of a database state, as discussed in [9]. A state is *consistent* if its tuples do not contradict any of the existing integrity constraints; it is *complete* if any tuples that can be derived from the existing tuples and the integrity constraints are already explicitly in the state. In independent schemes, local consistency of relations implies global consistency of the state. We formulate in this paper the stronger notion of *separability.* A database scheme is *separable* if local consistency implies consistency and completeness of the state. The database scheme of Example 1 is independent, but not separable.

Bancilhon and Spyratos [6] have proposed a very general notion of independence (that we shall call *BS-independence*) in terms of database mappings. Their definition seems to capture very naturally the idea of a relation giving no information about another relation in a state. In the special case, when the mapping amounts to a decomposition of a single relation, they show that Rissanen-independence does not yield independent mappings and they conclude that only trivial decompositions provide true independence. In this paper, we show that separability does provide independence in the sense of [6], when the set of constraints considered is a set of functional dependencies plus the full join dependency $\bowtie\mathbf{R}$. The importance of this special case is discussed in [7]. Instead of considering completely defined universal relations and their projections, we take universal relations with null values. A null value is interpreted as an existentially quantified variable. The mappings used for decomposition are *total projections*, that is, projections that discard any tuple containing null values on the target of the projection. It is then straightforward to verify that a database scheme is separable if and only if the set of total projections associated with its relation schemes is BS-independent. We then turn our attention to the characterization of separable schemes when independence is assumed. Together with Graham and Yannakakis' algorithms [10] for testing independence, our characterization yields a polynomial-time algorithm for testing separability.

2. Definitions and notation.

2.1. Basics. We fix a finite set of attributes $U=\{A_1, \cdots, A_n\}$ and call it the *universe.* A *relation scheme* R is a subset of U. A *database scheme* $\mathbf{R}=\{R_1, \cdots, R_k\}$ is a collection of relation schemes such that the union of the R_i's is U. A *tuple* defined on $R=\{A_1, \cdots, A_j\}$ is a function μ that maps each A_i to a value, for all $1 \leqq i \leqq j$. The value can either be a constant or a variable taken from an infinite set of uninterpreted symbols. If μ is a tuple on R and X is a subset of R, $\mu[X]$ denotes the restriction of μ to X. We say $\mu[X]$ is *total* (or μ is *total* on X) if $\mu[A_i]$ is a constant, for all $A_i \in X$. A *tableau* on U is a set of rows or tuples defined on U. A *relation* on R is a tableau on R such that every tuple is total on R. A *database state* r defined on **R** is $\langle r_1, \cdots, r_k\rangle$, where each r_i is a relation defined on $R_i \in \mathbf{R}$.

The tableau for a *database scheme* $\mathbf{R}=\{R_1, \cdots, R_k\}$, denoted $T_{\mathbf{R}}$, is a tableau of k rows where each row corresponds to one R_i in **R**. The rows are composed of *distinguished variables* (dv's), denoted by subscripted a's and *nondistinguished variables*

(ndv's), denoted by subscripted b's. The entries in $T_{\mathbf{R}}$ are defined as follows: If R_i is the row in $T_{\mathbf{R}}$ for relation scheme R_i, $R_i[A_j] = a_j$, if $A_j \in R_i$, else $R_i[A_j] = b_{ij}$.

We associate with each database state r, a tableau T_r, and call it *the tableau for a database state r.* T_r contains exactly one tuple for each tuple in each relation $r_i \in r$. Each tuple t in T_r corresponds to exactly a tuple $s \in r_i$, and is constructed by letting $t[R_i] = s[R_i]$ and filling the rest of t with distinct subscripted symbols not appearing elsewhere in T_r. We shall call the values in $t[R_i]$ and $t[U - R_i]$ the *constants and null symbols* respectively. Also t is said to *originate* from r_i. Given a tableau T_r for a database state r, $\pi^{\downarrow}_{\mathbf{R}}(T_r)$ denotes the *total projection* of T_r onto the database scheme $\mathbf{R}$ and is defined as $\pi^{\downarrow}_{\mathbf{R}}(T_r) = \langle \pi^{\downarrow}_{R_1}(T_r), \cdots, \pi^{\downarrow}_{R_s}(T_r) \rangle$, where $\pi^{\downarrow}_{R_i}(T_r) = \{t[R_i] | t \in T_r$ and $t[R_i]$ is total$\}$, for all $R_i \in \mathbf{R}$. We associate with $\mathbf{R}$ a *project-join mapping*, denoted by $m_{\mathbf{R}}$, defined by $m_{\mathbf{R}}(I) = \bowtie_{i=1}^{k} \pi_{R_i}(I)$, where I is a universal relation defined on U, π is the usual projection operator and $\bowtie$ is the natural join operator. It should be noted that a tableau T can always be transformed into a universal relation by assigning constant values to symbols or variables in T.

2.2. Dependencies and chasing. The kinds of constraints considered here are *functional dependencies* (fd's) and *join dependencies* (jd's). Associated with each fd or jd is an F-rule or a J-rule respectively. Given a tableau T and a set of fd's and jd's Σ, we can use the J-rules to infer additional tuples that must be in T if it is to satisfy Σ, and the F-rules to infer equalities among symbols of T for the same reason. T can be a tableau for a database scheme or a tableau for a database state. These transformation rules are defined as follows and their properties are described in [12]:

> F-rule: For each fd $X \to A$, there is an F-rule corresponding to it. Suppose tableau T has rows t_1, t_2 that agree in all X-columns. Let ν_1, ν_2 be the values in the A-column of t_1, t_2 respectively. Furthermore, $\nu_1 \neq \nu_2$. Applying the F-rule corresponding to $X \to A$ to rows t_1, t_2 of T yields a transformed tableau T'. T' is the same of T except that ν_1, ν_2 are renamed as follows. If one of ν_1 or ν_2 is dv (or constant) and the other is not, then rename all occurrences of the other symbol by the dv (or constant respectively). If both are ndv's (or null symbols), then rename all occurrences of the variable (or null symbol) with the higher subscript to be the variable (or null symbol respectively) with the lower subscript. If both are distinct constants, the result of applying the rule is usually defined to be the empty tableau.
>
> J-rule: Let $\mathbf{S} = \{S_1, \cdots, S_k\}$, with the union of S_i's yielding U. Rows $t_1, \cdots, t_k$ of T (not necessarily distinct) are joinable on $\mathbf{S}$ if there exists a row $w \notin T$ that agrees with t_i on S_i, $1 \leqq i \leqq k$. Row w is the result of joining t_i's. The J-rule corresponding to the jd $\bowtie \mathbf{S}$ takes rows $t_1, \cdots, t_k$ of T that are joinable on $\mathbf{S}$ and adds their result w to T to form T'.

Let $\chi = \chi_1 \cdots \chi_n$ be a sequence of transformation rules, then $\chi(T) = \chi_n(\cdots(\chi_1(T))\cdots)$ denotes the application of transformation rules χ to the tableau T. Suppose Σ is a set of fd's and/or jd's; $\mathrm{CHASE}_{\Sigma}(T)$ is the result of applying the corresponding F-rules and/or J-rules exhaustively to T.

Given a set of dependencies Σ, there are additional dependencies implied by this set in the sense that any relation that satisfies this set must also satisfy the additional dependency. The set of dependencies that are logically implied by Σ is the *closure* of Σ, denoted by Σ^+. Ψ is said to be *equivalent* to Σ if $\Psi^+ = \Sigma^+$. If Σ is a set of fd's F, then F^+ can be found by the inference rules in [1]. G is a *cover* of F if G is equivalent to F. Given a set of attributes X, the *closure of* X with respect to F, denoted by X^+,

is the set of attributes $\{A | X \to A \in F^+\}$. The *projection* of a set of fd's F onto R_i, denoted by $F^+|R_i$, is the set of fd's $X \to A \in F^+$ such that $XA \subseteq R_i$. An fd $X \to A$ is said to be *embedded* in a relation scheme R if $XA \subseteq R$. A database scheme $\mathbf{R}$ is said to be *cover embedding* for a set of fd's F if there exists a cover G of F such that for each fd $X \to A \in G$, $X \to A$ is embedded in some $R_i \in \mathbf{R}$. A database scheme is said to be *dependency preserving* or to *preserve* F, if for any universal relation I defined on U, I satisfies F implies $m_{\mathbf{R}}(I)$ also satisfies F. By a theorem in [4], $\mathbf{R}$ is cover embedding implies $\mathbf{R}$ is dependency preserving. If $F^+ = G^+$ then $\text{CHASE}_F(T)$ and $\text{CHASE}_G(T)$ are identical up to renaming of variables, for any tableau T on U. Let $R_i \in \mathbf{R}$ and $X \subseteq R_i$, X is a *superkey* of R_i if $X \to R_i \in F^+$. Given a set of fd's F, F is said to be *nonredundant* if there is no $X \to A \in F$ such that $(F - \{X \to A\})^+ = F^+$.

2.3. Weak instances, independence and separability. Let r be a state for a database scheme $\mathbf{R} = \{R_1, \cdots, R_k\}$. Let Σ be a set of fd's plus the full jd $\bowtie\mathbf{R}$. Let I be a universal relation defined on U. Then I is a *weak instance* for r with respect to a set of dependencies Σ if

- $\pi_{R_i}(I) \supseteq r_i$ for each $1 \leqq i \leqq k$,
- I satisfies Σ.

A database state r is said to be *consistent* with a set of dependencies Σ if a weak instance exists for the state with respect to Σ [9], [11]. The set of all consistent states for a database scheme $\mathbf{R}$ with respect to Σ is denoted by

$$WSAT(\mathbf{R}, \Sigma) = \{r | r \text{ is a state of } \mathbf{R} \text{ and is consistent with } \Sigma\}.$$

A relation r_i over R_i is said to be *consistent* with $\Sigma^+|R_i$ if and only if there is a universal relation I satisfying Σ such that $\pi_{R_i}(I) \supseteq r_i$.[1] The *locally consistent states* of $\mathbf{R}$ are elements of the set

$$LSAT(\mathbf{R}, \Sigma) = \{r | r_i \text{ is consistent with } \Sigma^+|R_i \text{ for each } r_i \in r\}.$$

That is, r is locally consistent if no relation $r_i \in r$ contains any contradictory information. A database scheme $\mathbf{R}$ is said to be *independent* with respect to Σ when

$$LSAT(\mathbf{R}, \Sigma) = WSAT(\mathbf{R}, \Sigma).$$

A database state r is said to be *complete* if it is consistent with the given dependencies and

$$r = \bigcap \{\pi_{\mathbf{R}}(I) | I \text{ is a weak instance for } r\}$$

where the intersection of two database states defined on $\mathbf{R}$ is the result of the intersection of the pairs of the corresponding relations in the database state. In other words, if a tuple appears in the total projection of *every* weak instance for the state, the state must contain that tuple in order to be complete. As explained in [9], the idea is that a complete state contains explicitly all the tuples whose existence can be derived from other tuples and the dependencies. A database scheme $\mathbf{R}$ is said to be *everywhere-complete* if for any $r \in WSAT(\mathbf{R}, \Sigma)$, r is complete; otherwise it is said to be *somewhere-incomplete*. The completeness of a consistent state r can be tested by chasing T_r, as described by the following theorem.

THEOREM 1. *For any state* $r \in WSAT(\mathbf{R}, \Sigma)$, *the following are equivalent.*

(a) *r is complete with respect to Σ.*

(b) $r = \pi^{\downarrow}_{\mathbf{R}}(CHASE_{\Sigma}(T_r))$.

(c) $r = \bigcap \{\pi_{\mathbf{R}}(I) | I \text{ is a weak instance for } r\}$.

[1] In the case we are interested in, that is $\Sigma = F \cup \{\bowtie\mathbf{R}\}$, the containment can be replaced by equality [10].

Proof. See [9]. □

Last, our most important definition is: a database scheme **R** is *separable* under a set of dependencies Σ if every locally consistent state of **R** is complete. That is, **R** is separable if and only if **R** is independent and everywhere-complete.

3. BS-independence and separability. Bancilhon and Spyratos [6] have studied general mappings between database schemes and propose the following intuitive definition of independence:

> Two data base mappings f and g are independent if knowing the value of f does not affect our knowledge of the value of g, and vice versa.

They capture this idea formally as follows. Let $SAT(\mathbf{R}, D)$ be the set of all states of database scheme **R** that satisfy the set of "constraints" in D. We leave the constraints and the definition of satisfaction unspecified for now. A set of mappings $\{g_i\}_{i=1}^{k}$, $g_i : SAT(\mathbf{R}, D) \to SAT(S_i, \varnothing)$ is *BS-independent* if for all $\{\rho_1, \cdots, \rho_k\} \subseteq SAT(\mathbf{R}, D)$, there exists $\rho \in SAT(\mathbf{R}, D)$ such that $g_i(\rho_i) = g_i(\rho)$, for all $1 \leqq i \leqq k$. Intuitively, given any k points in the respective images of the $\{g_i\}$ mappings, we can find a single common state in the domain of the g_i's such that all the points are images of this single state. Thus we are free to change any one of the points independently of the others, as long as we make sure that it is still an image of some legal state and we are guaranteed that the single state ρ will still exist.

We are interested in a special case. Let $\mathbf{R} = \{U\}$ be the universal relation scheme. For the purposes of this section, a state of **R** is a tableau defined on U, rather than a universal relation. In other words, we allow "missing value" nulls to appear in our universal relations. S_i is a single-relation database scheme with one relation scheme $R_i \subseteq U$ in it. Let $\mathbf{S} = \{R_1, \cdots, R_k\}$ be a database scheme obtained from the S_i's. Let $D = \Sigma = F \cup \{\bowtie \mathbf{S}\}$, with the notion of satisfaction being weak instance satisfaction. That is, $SAT(\mathbf{R}, \Sigma) = WSAT(\mathbf{R}, \Sigma)$. For $T \in SAT(\mathbf{R}, \Sigma)$, $g_i(T) = \pi^{\downarrow}_{R_i}(T)$. The following theorem shows that separability of **S** coincides with BS-independence of the mappings $\{\pi^{\downarrow}_{R_i}\}$.

THEOREM 2. **S** *is separable if and only if the set of mappings* $\{\pi^{\downarrow}_{R_i}\}$ *is BS-independent.*

Proof. (If) Suppose the $\pi^{\downarrow}_{R_i}$ mappings are BS-independent. Let $\rho = \langle \rho_1, \cdots, \rho_k \rangle$ be a locally consistent state of **S**. Then there exists for each ρ_i a satisfying universal relation T_i such that $\pi^{\downarrow}_{R_i}(T_i) = \rho_i$. Hence, there is a universal relation T satisfying the dependencies such that $\pi^{\downarrow}_{R_i}(T) = \rho_i$, for all i. Generalizing slightly a theorem of Honeyman [11], we can show that $\pi^{\downarrow}_{R_i}(\mathrm{CHASE}_{\Sigma}(T_\rho)) \subseteq \pi^{\downarrow}_{R_i}(T)$, for any such T. Hence, $\pi^{\downarrow}_{R_i}(\mathrm{CHASE}_{\Sigma}(T_\rho)) \subseteq \rho_i$, for all i. Thus ρ is complete. We have showed that every locally consistent state of **S** is complete, so **S** is separable.

(Only if) Given $\pi^{\downarrow}_{R_1}(T_1), \cdots, \pi^{\downarrow}_{R_k}(T_k)$, where $T_1, \cdots, T_k$ are satisfying universal relations, let $\rho = \langle \rho_1, \cdots, \rho_k \rangle$, where $\rho_i = \pi^{\downarrow}_{R_1}(T_1), \cdots, \rho_k = \pi^{\downarrow}_{R_k}(T_k)$. The resulting state ρ is locally consistent by construction. Since **S** is separable, this implies that ρ is complete. That is, $\rho = \pi^{\downarrow}_{\mathbf{S}}(\mathrm{CHASE}_{\Sigma}(T_\rho))$. Now $\mathrm{CHASE}_{\Sigma}(T_\rho)$ is an element of the domain of each mapping $\pi^{\downarrow}_{R_i}$, showing that the mappings are BS-independent. □

The reason that separability does provide independent mappings while Rissanen-independence does not is that, by allowing null values in the universal relations, we can handle database states that are not join-consistent. So the statement in [6] that "it is not possible to give meaningful examples of independent decompositions using projections" is true if we restrict ourselves to the usual projection operator, but we solve the difficulty by using total projection. For example, the decomposition of a relation on *Employee, Department, Manager*, with *Employee* → *Department* and *Department* → *Manager*, into *Employee, Department* and *Department, Manager* [6, Ex. 10] is

indeed separable and thus the mappings for the corresponding total projections are independent.

4. A characterization of separable schemes. In this section, we shall characterize separable database schemes when the constraint given is $\Sigma = H \cup \{\bowtie \mathbf{R}\}$, where H is a set of fd's. Alternatively, the characterization can also be applied to database schemes that are cover embedding for a set of fd's H. The independence part was investigated and characterized by an algorithm in [10]. We will then identify the condition under which an independent database scheme is also everywhere-complete. Before we give the characterization, we need to know some properties of this class of independent database schemes.

THEOREM 3. *The database scheme* $\mathbf{R}$ *is independent with respect to* Σ *if and only if the following two conditions are satisfied*:

(1) $\mathbf{R}$ *embeds a cover* F *of* G, *where* G *is the set of fd's implied by* Σ.

(2) $\mathbf{R}$ *is independent with respect to* F.

Proof. See [10]. □

In [10] a polynomial-time algorithm is derived to find the embedded cover F if $\mathbf{R}$ is independent with respect to Σ, where $F = \cup F_i$ and F_i is an embedded cover for $F^+|R_i$, for all $R_i \in \mathbf{R}$. The following theorem states that $\mathbf{R}$ is independent exactly if for any r defined on $\mathbf{R}$, and if every relation $r_i \in r$ satisfies the projected fd's $F^+|R_i$ then r is a consistent state.

THEOREM 4. *Let* F *be a set of fd's embedded in* $\mathbf{R}$. *Then the following are equivalent.*

(1) $\mathbf{R}$ *is independent with respect to* $\Sigma = H \cup \{\bowtie \mathbf{R}\}$.

(2) $\mathbf{R}$ *is independent with respect to* F.

(3) *If for each* $r_i \in r$, r_i *satisfies* $F^+|R_i$, *then* r *is a consistent state, for any* r *defined on* $\mathbf{R}$.

Proof. See [10]. □

Let $\mathbf{R}$ be an arbitrary scheme and F be a set of fd's. We say $\mathbf{R}$ satisfies the *total derivation* condition if whenever there are R_i, R_j in $\mathbf{R}$, $i \neq j$, such that $R_i^+ \supseteq R_j$, then there exists a nonredundant cover F_j of $F^+|R_j$ and an fd $X \to A \in F_j$ such that X is a superkey of R_j. Our main result in this section will be that the total derivation condition is necessary and sufficient for separability under our assumptions.

Example 2. $\mathbf{R} = \{R_1(Employee, Department), R_2(Department, Manager)\}$, $F = \{Employee \to Department, Department \to Manager\}$.

It is easy to verify that $\mathbf{R}$ is independent. Given a locally consistent state r for this example, consider the chase performed on its tableau. For any tuple t_1 that originates from r_1 that has a constant in the *Manager*-component during the chase process, that value must come from a tuple t_2 that originates from r_2 using the fd *Department* $\to$ *Manager*, hence t_1 and t_2's *Department* and *Manager* components are equal. Therefore $t_1[R_2]$ is "totally" derived from a tuple in r_2, hence $\mathbf{R}$ is everywhere-complete. Therefore $\mathbf{R}$ is separable.

The following is another example to illustrate why the total derivation condition characterizes separability for an independent scheme.

Example 3. $\mathbf{R} = \{R_1(Course, Student, Instructor), R_2(Course, Student, Year)\}$, $F = \{Course \to Instructor\}$.

It is easy to verify that $\mathbf{R}$ is independent. However $\mathbf{R}$ is not everywhere-complete. Consider the following database state.

Course	*Student*	*Instructor*
c_1	s_1	i_1

Course	*Student*	*Year*
c_1	s_2	y_1

Since *Course* → *Instructor*, s_2 is taught by instructor i_1. Therefore after a chase is performed on the tableau for the state, the tuple $\langle c_1, s_2, i_1\rangle$ defined on R_1 can be derived from the state and will appear in every weak instance, but it is not explicitly in r_1 of the state. The reason is that **R** does not satisfy the total derivation condition and hence is somewhere-incomplete. $R_2^+ \supseteq R_1$ but the only nonredundant cover for $F^+|R_1$ is {*Course* → *Instructor*} and *Course* is not a superkey of R_1. Because of this, it is possible that in some state some tuple t_2 that originates from r_2 has its R_1-components become constants in the chase process but $t_2[R_1]$ does not equal any tuple in r_1.

In determining if **R** satisfies the total derivation condition, we need to find a nonredundant cover F_i for $F^+|R_i$, $R_i \in \mathbf{R}$ and a fd $X \to A \in F_i$ such that X is a superkey of R_i. There may be many nonredundant covers for $F^+|R_i$. However, as will be shown by a corollary below, it really does not matter which nonredundant cover F_i we choose in determining if **R** satisfies the total derivation condition. Before we prove the corollary, we need a lemma from [2].

LEMMA 1. *Let G_1 and G_2 be two nonredundant sets of fd's with $G_1^+ = G_2^+$. If $X \to A \in G_1$, then there exists an $Y \to B \in G_2$ with $Y \to X$ and $X \to Y$ in G_1^+.*

Proof. See [2]. □

COROLLARY 1. *Let* **R** *be a database scheme and F be a set of fd's. Let F_i be a nonredundant cover for $F^+|R_i$, for any $R_i \in \mathbf{R}$. If for all $X \to A \in F_i$, X is not a superkey of R_i, then for all nonredundant covers G_i for $F^+|R_i$, there does not exist a fd $Y \to B \in G_i$ such that Y is a superkey of R_i.*

Proof. Prove by contradiction. Suppose there exists such a nonredundant cover G_i such that $Y \to B \in G_i$ and Y is a superkey of R_i. By Lemma 1, there exists $Z \to C \in F_i$ such that $Y \to Z$ and $Z \to Y$ are in G_i^+. Since $G_i^+ = F^+|R_i$, $Z \to Y$ and $Y \to Z$ are in G_i^+ imply $Z \to Y$ and $Y \to Z$ are in F^+. But since Y is a superkey of R_i, $Y \to R_i \in F^+$ and hence $Z \to R_i \in F^+$. Therefore Z is also a superkey of R_i. This contradicts the assumption that for all fd's $X \to A \in F_i$, X is not a superkey of R_i. Therefore we conclude that Y is not a superkey of R_i. This completes the proof. □

The outline of the rest of this section is the following. Given an independent scheme **R** with respect to an embedded cover F, we will prove that **R** satisfies the total derivation condition if and only if **R** is everywhere-complete with respect to F. Then we will prove that **R** is everywhere-complete with respect to F if and only if **R** is everywhere-complete with respect to $\Sigma = H \cup \{\bowtie \mathbf{R}\}$, where H is a set of fd's and F is an embedded cover for the set of fd's implied by Σ.

Given a cover embedding database scheme **R**, the following theorem gives a sufficient condition for **R** to be somewhere-incomplete with respect to a set of fd's F.

THEOREM 5. *Let F be an embedded cover of fd's and let F_i be the fd's of F that are embedded in the relation scheme R_i. If there exists an R_j that is contained in some R_i^+, $i \neq j$, and no fd of F_j has a left-hand side that is a superkey of R_j, then* **R** *is somewhere-incomplete.*

Proof. Let **R** be a database scheme that satisfies the conditions stated in the theorem. We want to construct a consistent but incomplete state r. The state r is constructed from a satisfying universal relation I as follows.

For each $R_k \in \mathbf{R}$, $k \neq j$, the universal relation I has a tuple with 1's in the columns of R_k^+ and distinct constants elsewhere; this tuple is tagged with R_k. For each fd of F_j having a left-hand side X, there is a tuple having 1's in columns of X^+ and distinct constants elsewhere; this tuple is tagged with R_j. It is easy to verify that I is a satisfying relation.

Let r be $\langle r_1, \cdots, r_n\rangle$, where r_p is the projection onto R_p of the tuples of I with tag R_p. Since I is a satisfying relation, I is a weak instance of r and hence r is a

consistent state. However r is incomplete since at the end of the chase the tuple of r_i has 1's in all columns of $R_i^+ \supseteq R_j$, while r_j does not have a tuple with all 1's. □

We need the following lemmas to show that the total derivation condition is sufficient for an independent scheme to be everywhere-complete.

LEMMA 2. *Let r be a database state defined on a database scheme $\mathbf{R}$. Let t be a tuple in $\chi(T_r)$, χ is a sequence of transformation rules for a set of fd's F and t originates from some relation $r_i \in r$. If $t[A]$ is a constant in $\chi(T_r)$ then $R_i[A]$ in $\mathrm{CHASE}_F(T_{\mathbf{R}})$ is a dv.*

Proof. See [8]. □

LEMMA 3. *Let $\mathbf{R}$ be a database scheme that preserves F. Then $\mathrm{CHASE}_F(T_{\mathbf{R}})$ is obtained from $T_{\mathbf{R}}$ by replacing each row R_i in $T_{\mathbf{R}}$ by the row $\mathrm{cl}(R_i)$, where $\mathrm{cl}(R_i)$ has dv's exactly in the R_i^+-positions and the same ndv's as R_i in other positions.*

Proof. See [4]. □

The following lemma illustrates that each derived value in $\mathrm{CHASE}_F(T_r)$ is "uniquely" derived for an independent scheme. The reader may verify this fact for the examples we gave so far.

LEMMA 4. *Suppose $\mathbf{R}$ is independent with respect to a set of embedded fd's F. Then for any $r \in WSAT(\mathbf{R}, F)$, and for any nonredundant cover F_j of $F^+|R_j$, for any $R_j \in \mathbf{R}$ and for any fd $X \to A \in F_j$, $\pi^{\downarrow}_{XA}(\mathrm{CHASE}_F(T_r)) = \pi_{XA}(r_j)$.*

Proof. Suppose there exists $r \in WSAT(\mathbf{R}, F)$ and $X \to A \in F_j$, for some nonredundant cover F_j of $F^+|R_j$, for some $R_j \in \mathbf{R}$ such that $\pi^{\downarrow}_{XA}(\mathrm{CHASE}_F(T_r)) \neq \pi_{XA}(r_j)$. This means there exists a tuple t in $\mathrm{CHASE}_F(T_r)$ such that $t[XA]$ is total and $t[XA] \notin r_j[XA]$. This in turn implies $t[X] \notin r_j[X]$. Let us define a tuple t' on R_j as follows: Let X^* be the closure of X with respect to $F_j - \{X \to A\}$. Let $Q = \{B | B \in X^*$ and $t[B]$ is a constant in $\mathrm{CHASE}_F(T_r)\}$. Let $t'[Q] = t[Q]$, and assign a value not appearing anywhere else to $t'[C]$, for all $C \notin Q$. Let $r_j' = r_j \cup \{t'\}$. We claim that r_j' satisfies $F_j - \{X \to A\}$. Suppose it does not, then there exists $Z \to B \in F_j - \{X \to A\}$ such that t' and t'' violate it, where $t'' \in r_j$. This means $t'[Z] = t''[Z]$ but $t'[B] \neq t''[B]$. By the way t' is constructed, $t'[C]$ is assigned a value not appearing anywhere else, for any $C \notin Q$. Hence $Z \subseteq Q$. Since $Z \subseteq Q$, $B \in X^*$. Let us consider the following two possible cases for B.

Case 1: $B \in Q$. By the way we define t', $t'[B] = t[B]$. This implies $r_j \cup \{t[R_j]\}$ violates $Z \to B \in F$. Since we assume $r \in WSAT(\mathbf{R}, F)$, this case is impossible.

Case 2: $B \notin Q$. In this case, $t[B]$ is a null symbol in $\mathrm{CHASE}_F(T_r)$. But since $t''[Z] = t'[Z] = t[Z]$, $t''[B]$ is a constant, and $Z \to B \in F$, $t[B]$ should not be a null symbol in $\mathrm{CHASE}_F(T_r)$. Again, a contradiction.

Therefore we can conclude that r_j' satisfies $F_j - \{X \to A\}$. But $t[X] = t'[X] \notin r_j[X]$. Hence $X \to A$ cannot be violated by r_j'. Therefore r_j' satisfies F_j. By Theorem 4, $r \cup \{t'\} \in WSAT(\mathbf{R}, F)$. But $A \notin X^*$ (or else $X \to A$ is redundant in F_j). Therefore $t'[A]$ is assigned a distinct value from $t[A]$. Therefore t and t' violate $X \to A \in F$ when we chase the tableau for the new state $r \cup \{t'\}$. Hence $r \cup \{t'\} \notin WSAT(\mathbf{R}, F)$. This is a contradiction. □

With the above lemmas, the following lemma shows that the total derivation condition is sufficient for independent schemes to be everywhere-complete.

LEMMA 5. *Let $\mathbf{R}$ be an independent scheme with respect to an embedded cover F. If $\mathbf{R}$ satisfies the total derivation condition then $\mathbf{R}$ is everywhere-complete.*

Proof. Suppose we are given such a database scheme $\mathbf{R}$ and $\mathbf{R}$ satisfies the total derivation condition but $\mathbf{R}$ is somewhere-incomplete. It means there exists $r \in WSAT(\mathbf{R}, F)$ such that $\pi^{\downarrow}_{R_j}(\mathrm{CHASE}_F(T_r)) \supset r_j$, for some $R_j \in \mathbf{R}$. The extra tuple t_i in $\pi^{\downarrow}_{R_j}(\mathrm{CHASE}_F(T_r))$ has to originate from some relation $r_i \in r$, $i \neq j$. But we are going

to show that this is not possible. Since $t_i[R_j]$ is total, by Lemma 2, $R_i[R_j]$ in $\text{CHASE}_F(T_{\mathbf{R}})$ are dv's. By Lemma 3, $R_i^+ \supseteq R_j$. However, $\mathbf{R}$ satisfies the total derivation condition, so there exists a nonredundant cover F_j of $F^+|R_j$ and a fd $X \to A \in F_j$ such that X is a superkey of R_j. By Lemma 4, $\pi^{\downarrow}_{XA}(\text{CHASE}_F(T_r)) = \pi_{XA}(r_j)$. This implies that $\pi^{\downarrow}_{R_j}(\text{CHASE}_F(T_r)) = r_j$, for X is a superkey of R_j. This is a contradiction. Therefore $\mathbf{R}$ is in fact everywhere-complete. □

We are ready to give our main theorem, which characterizes separability in the special case that we are interested in.

THEOREM 6. *Let* $\mathbf{R}$ *be an independent scheme with respect to an embedded cover* F. *Then* $\mathbf{R}$ *is everywhere-complete with respect to* F *if and only if* $\mathbf{R}$ *satisfies the total derivation condition.*

Proof. (If) This proof follows from Lemma 5.

(Only if) Since $\mathbf{R}$ is cover embedding, this direction follows from Theorem 5. □

Sagiv [15], [16] identified a class of independent schemes that embed a set F of key dependencies. In this special case, we have a simpler characterization for separable schemes.

COROLLARY 2. *Suppose* $\mathbf{R}$ *is independent with respect to a set* F *of key dependencies. Then* $\mathbf{R}$ *is separable if and only if for each* R_i *contained in the closure of some* R_j, $i \neq j$, R_i *contains a nontrivial key.*

Proof. This follows directly from the definition of total derivation condition. □

The following corollary states that a database scheme generated by the algorithm of Biskup, Dayal and Bernstein [3] is separable, provided it is independent.

COROLLARY 3. *If* $\mathbf{R}$ *is independent and is a lossless, 3NF and cover embedding database scheme generated by Biskup, Dayal and Bernstein algorithm* [3], *then* $\mathbf{R}$ *is separable.*

Proof. This follows directly from the synthesizing algorithm and the total derivation condition. □

Before we bring the $\text{jd} \bowtie \mathbf{R}$ back into the picture, we need the following theorem. Let $\mathbf{R}$ be a dependency preserving database scheme and Σ be the union of the $\text{jd} \bowtie \mathbf{R}$ and a set of fd's F. The next theorem describes some properties of $\mathbf{R}$ and Σ.

THEOREM 7. *Let* $\mathbf{R}$ *and* Σ *be defined as above.*

(1) $WSAT(\mathbf{R}, \Sigma) = WSAT(\mathbf{R}, F)$.

(2) *For any* $r \in WSAT(\mathbf{R}, F)$, *for any* $X \subseteq R_i$, *and for any* $R_i \in \mathbf{R}$, $\pi^{\downarrow}_X(\text{CHASE}_\Sigma(T_r)) = \pi^{\downarrow}_X(\text{CHASE}_F(T_r))$.

Proof. (1) Clearly $WSAT(\mathbf{R}, \Sigma) \subseteq WSAT(\mathbf{R}, F)$ for $F \subseteq \Sigma$. Conversely if $r \in WSAT(\mathbf{R}, F)$, we want to show that $r \in WSAT(\mathbf{R}, \Sigma)$. We chase T_r with respect to Σ in the following ways:

(a) Obtain $\text{CHASE}_F(T_r)$ from T_r.

(b) Apply the $\text{jd} \bowtie \mathbf{R}$ to $\text{CHASE}_F(T_r)$ to obtain T_r^*. That is, $T_r^* = m_{\mathbf{R}}(\text{CHASE}_F(T_r))$. Since $\mathbf{R}$ preserves F, by a theorem in [4], $T_r^* = \text{CHASE}_\Sigma(T_r)$. Hence $r \in WSAT(\mathbf{R}, \Sigma)$.

(2) From (1) above, we know $\pi_{R_i}(\text{CHASE}_\Sigma(T_r)) = \pi_{R_i}(T_r^*) = \pi_{R_i}(\text{CHASE}_F(T_r))$, for all $R_i \in \mathbf{R}$. Hence $\pi_X(\text{CHASE}_\Sigma(T_r)) = \pi_X(\text{CHASE}_F(T_r))$ for any $X \subseteq R_i$. Hence $\pi^{\downarrow}_X(\text{CHASE}_\Sigma(T_r)) = \pi^{\downarrow}_X(\text{CHASE}_F(T_r))$ for any $X \subseteq R_i$. □

The next result brings the $\text{jd} \bowtie \mathbf{R}$ back into the picture, showing that our result still holds.

THEOREM 8. *Let* $\Sigma = H \cup \{\bowtie \mathbf{R}\}$, *where* H *is a set of fd's, and* $\mathbf{R}$ *is independent with respect to* Σ. *Let* F *be an embedded cover for the set of fd's implied by* Σ. *Then* $\mathbf{R}$ *is everywhere-complete with respect to* Σ *if and only if* $\mathbf{R}$ *is everywhere-complete with respect to* F *alone.*

Proof. As explained at the beginning of this section, we can always find the embedded cover F using an algorithm in [10]. Clearly $F \cup \{\bowtie \mathbf{R}\}$ is equivalent to Σ. Without loss of generality, we assume $\Sigma = F \cup \{\bowtie \mathbf{R}\}$. Since $\mathbf{R}$ is cover embedding and therefore dependency preserving, by Theorem 7, $WSAT(\mathbf{R}, \Sigma) = WSAT(\mathbf{R}, F)$. Therefore all we need to show is that for all $r \in WSAT(\mathbf{R}, F)$, r is complete with respect to F if and only if r is complete with respect to Σ. By Theorem 7, $\pi^{\downarrow}_{R_i}(\mathrm{CHASE}_{\Sigma}(T_r)) = \pi^{\downarrow}_{R_i}(\mathrm{CHASE}_{F}(T_r))$, for any $R_i \in \mathbf{R}$. Hence the theorem is proved. □

We have completely characterized an important class of separable database schemes when the only constraints on $\mathbf{R}$ are the union of the jd $\bowtie \mathbf{R}$ and a set of fd's H. Alternatively, the characterization can also be applied to cover embedding database schemes. The testing for separability is as follows. First determine if $\mathbf{R}$ is independent with respect to Σ using an algorithm in [10]. If it is, an embedded cover F can be found. Then $\mathbf{R}$ is separable with respect to Σ exactly when $\mathbf{R}$ satisfies the total derivation condition with respect to F. It is worth mentioning that all of this can be done in polynomial time. The following corollary summarizes the results.

COROLLARY 4. *Let $\Sigma = H \cup \{\bowtie \mathbf{R}\}$, where H is a set of fd's and $\mathbf{R}$ is independent with respect to Σ. Let F be an embedded cover for the set of fd's implied by Σ. Then the following are equivalent:*

(1) $\mathbf{R}$ *is everywhere-complete with respect to* Σ.

(2) $\mathbf{R}$ *is everywhere-complete with respect to* F.

(3) $\mathbf{R}$ *satisfies the total derivation condition.*

Proof. By Theorem 8, (1) and (2) are equivalent. By Theorem 4, $\mathbf{R}$ is independent with respect to Σ if and only if $\mathbf{R}$ is independent with respect to F. By Theorem 6, (2) and (3) are equivalent. □

5. Conclusion. We have proposed and investigated the notion of separability to capture the design goal of independently updatable decompositions. This notion is more realistic than the early work on independent decompositions, since it does not require the pure universal relation assumption. It is also stronger than the weak instance based notion described in the Introduction, since it assumes both consistency and completeness of the overall database when local consistency is maintained at each relation.

As evidence in favor of separability as a natural concept, we have shown it equivalent to a specialization of the abstract independent mappings defined by Bancilhon and Spyratos [6], when the constraints given are a set of fd's plus the full jd $\bowtie \mathbf{R}$.

We have finally characterized separable schemes when the constraints given are a set of fd's and the jd $\bowtie \mathbf{R}$. The characterization is also applicable to cover embedding schemes. Together with results in [10], our characterization yields a polynomial-time algorithm for testing separability in these cases.

Acknowledgment. The authors are grateful to the referees for their helpful comments. One of the referees' suggestions significantly simplified the proof of Theorem 5.

REFERENCES

[1] W. W. ARMSTRONG, *Dependency structures of data base relationships*, Proc. International Federation for Information Processing, 1974, North-Holland, Amsterdam-New York, pp. 580-583.

[2] P. A. BERNSTEIN, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. Database Systems, 1 (1976), pp. 277-298.

[3] J. BISKUP, U. DAYAL AND P. A. BERNSTEIN, *Synthesizing independent database schemas*, Proc. Special Interest Group for Management of Data, 1979, pp. 143-152.

[4] C. BEERI, A. O. MENDELZON, Y. SAGIV AND J. D. ULLMAN, *Equivalence of relational database schemes*, this Journal, 10 (1981), pp. 352-370.

[5] C. BEERI AND J. RISSANEN, *Faithful representations of relational database schemes*, IBM RJ2722, 1980.

[6] F. BANCILHON AND N. SPYRATOS, *Independent components of data bases*, Proc. Very Large Data Bases, 1981, pp. 398-408.

[7] R. FAGIN, A. O. MENDELZON AND J. D. ULLMAN, *A simplified universal relation assumption and its properties*, ACM Trans. Database Systems, 7 (1982), pp. 343-360.

[8] M. H. GRAHAM, *Satisfying database states*, Computer Systems Research Group TR-137, Univ. of Toronto, December, 1981.

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

[10] M. H. GRAHAM AND M. YANNAKAKIS, *Independent database schemas*, J. Comp. System Sci., 28 (1984), pp. 121-141.

[11] P. HONEYMAN, *Testing satisfaction of functional dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 668-677.

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

[13] D. MAIER, A. O. MENDELZON, F. SADRI AND J. D. ULLMAN, *Adequacy of decompositions of relational databases*, J. Comput. System Sci., 21 (1980), pp. 368-379.

[14] J. RISSANEN, *Independent components of relations*, ACM Trans. Database Systems, 2 (1977), pp. 317-325.

[15] Y. SAGIV, *Can we use the universal instance assumption without using nulls*? Proc. Special Interest Group on Management of Data, (1981), pp. 108-120.

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

[17] M. T. VARDI, *On decomposition of relational databases*, Proc. 23rd Annual Symposium on the Foundation of Computer Science, 1982, pp. 176-185.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987
© 1987 Society for Industrial and Applied Mathematics
005

A NOTE ON RANDOMIZED POLYNOMIAL TIME*

STUART A. KURTZ†

Abstract. We show that if A and B are independent random sets, then $\mathbf{P}^A \cap \mathbf{P}^B = \mathbf{BPP}$ with probability one.

Key words. random polynomial time, **BPP**, random oracles

AMS(MOS) subject classifications. 03D10, 03D15, 68C25, 68C40

1. Introduction. In this paper we answer affirmatively a question of Babai and Sipser, who wanted to know whether $\mathbf{P}^A \cap \mathbf{P}^B = \mathbf{BPP}$ with probability one for independent random sets A and B. The class **BPP** is defined in [Gil77].

In this note, L will denote a language, i.e., a subset of Σ^*. The set of all languages is $\mathscr{P}(\Sigma^*)$, or $\mathscr{P}$ for short. We will use $\mu^{\mathscr{P}}$ to denote the usual Lebesgue measure on $\mathscr{P}$, and $\mu^{\mathscr{P}\times\mathscr{P}}$ to denote the product measure on $\mathscr{P}\times\mathscr{P}$. If A is an oracle, than $\lambda x.\, A[x]$ denotes the characteristic function of A.

2. A lemma. The following lemma is known in the folklore as the 0-1 law for **BPP**.

LEMMA 2.1. *Let L be a language such that for some polynomial time Turing machine M, $\mu^{\mathscr{P}}\{A \mid L(M^A) = L\} > 0$. Then $L \in \mathbf{BPP}$.*

Proof. By a well-known result of measure theory (the metric density theorem (cf. [Oxt70, p. 17]))[1], if a set L has positive measure, then for every $\delta < 1$, there is a basic open interval Δ such that L has density at least δ in Δ. A basic open interval in the context of languages is simply the set of extensions of a finite function. Therefore, let Δ be a finite function such that $\mu^{\mathscr{P}}\{A \supset \Delta \mid L(M^A) = L\} > 3/4\mu(\Delta)$. (Here $\mu(\Delta)$ denotes the measure of the set of extensions of Δ in $\mathscr{P}$.)

Let f be the function from oracles to oracles given by

$$f(A)[x] = \begin{cases} \Delta[x] & \text{if } x \in \mathrm{dom}(\Delta), \\ A[x] & \text{otherwise.} \end{cases}$$

Let M' be the Turing machine such that $M'^A(x) = M^{f(A)}(x)$. It is easy to see that $\mu^{\mathscr{P}}\{A \mid L(M'^A) = L\} \geqq \frac{3}{4}$, as all oracles have in effect been translated into an interval relative to whose elements M accepts L at least three-quarters of the time.

We now convert M' to a **BPP**-machine M'' accepting L. The idea is to use a random coin to simulate a random oracle. It is not enough, however, to simply replace each oracle query by a coin toss, as M' may have queried the oracle about certain strings more than once. A random oracle would necessarily give the same question each time a given query was made, whereas a random coin toss might not. Fortunately, this is an easy difficulty to avoid: M'' maintains a table of query-response pairs. Whenever the simulation of M' makes a query, M'' checks whether this query was made before. If so, it repeats its original response, and if not, it generates its response by flipping its coin (and it then stores the query-response pair in its table).

* Received by the editors January 22, 1986; accepted for publication December 4, 1986.

† Department of Computer Science, The University of Chicago, Chicago, Illinois 60637.

[1] The intuition behind the metric density theorem is the first of Littlewood's three principles [Lit44, p. 26]: every measurable set is almost the union of finitely many intervals.

It is clear that M'' is a **BPP**-machine, as the table manipulation is not expensive. Moreover, it is clear that for every x, the probability that $M''(x) = L[x]$ is at least $\frac{3}{4}$. (The probability that we will be correct about a single string being at least as large as the probability that we will be correct about all strings.) □

The following corollary is a direct consequence of this lemma.

COROLLARY 2.2. *If L is a language such that $\mu^{\mathscr{P}}\{A \mid L \in \mathbf{P}^A\} > 0$, then $L \in \mathbf{BPP}$.*

Proof. If L is as hypothesized, then by the countable additivity of Lebesgue measure there is a polynomial time Turing machine M such that $\mu^{\mathscr{P}}\{A \mid L = L(M^A)\} > 0$. The corollary follows immediately from Lemma 2.1. □

3. The results.

THEOREM 3.1. *Let A and B be two independent random oracles. Then $\mathbf{P}^A \cap \mathbf{P}^B = \mathbf{BPP}$ with probability one.*

Proof. We know [BG81, p. 104] that for almost all oracles A, $\mathbf{P}^A \subset \mathbf{BPP}$. Therefore with probability one we must have $\mathbf{P}^A \cap \mathbf{P}^B \subset \mathbf{BPP}$. It suffices to demonstrate the reverse containment.

If the reverse containment fails, then there is a language in $\mathbf{P}^A \cap \mathbf{P}^B$ which is not in **BPP** for almost all A and B. In notation:

$$\mu^{\mathscr{P}\times\mathscr{P}}\{(A, B) \mid (\exists M, N)[L(M^A) = L(N^B) \notin \mathbf{BPP}]\} > 0.$$

As there are only countably many pairs (M, N) of Turing machines, there must be a fixed pair (M_0, N_0) for which

$$\mu^{\mathscr{P}\times\mathscr{P}}\{(A, B) \mid L(M_0^A) = L(N_0^B) \notin \mathbf{BPP}\} > 0.$$

By Fubini's Theorem [Roy68, p. 269], we know that there is a fixed A' such that

$$\mu^{\mathscr{P}}\{B \mid L(M_0^{A'}) = L(N_0^B) \notin \mathbf{BPP}\} > 0.$$

Indeed, Fubini's Theorem tells us that the set of such A' must have positive measure.

As A' and M_0 are fixed, $L(M_0^{A'})$ is a fixed language L'. Thus

$$(\exists L' \notin \mathbf{BPP})[\mu^{\mathscr{P}}\{B \mid L' = L(N_0^B)\} > 0].$$

This, however, contradicts Lemma 2.1. □

REFERENCES

[BG81] C. J. BENNETT AND J. GILL, *Relative to a random oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq \mathbf{coNP}^A$ with probability 1*, this Journal, 10 (1981), pp. 96–113.

[Gil77] J. GILL, *Computational complexity of probabilistic turing machines*, this Journal, 6 (1977), pp. 675–695.

[Lit44] J. E. LITTLEWOOD, *Lectures on the Theory of Functions*, Oxford University Press, New York, 1944.

[Oxt70] J. OXTOBY, *Measure and Category*, Springer-Verlag, New York, 1970.

[Roy68] H. L. ROYDEN, *Real Analysis*, second ed., MacMillan, New York, 1968.

SIAM. J. COMPUT.
Vol. 16, No. 5, October 1987

© 1987 Society for Industrial and Applied Mathematics
006

DOUBLY LEXICAL ORDERINGS OF MATRICES*

ANNA LUBIW†

Abstract. Every matrix has a *doubly lexical ordering*: an ordering of the rows and columns so that the row vectors are lexically (or "lexicographically") increasing and the column vectors are lexically increasing. Every graph has a *lexical ordering*: a vertex ordering making the neighbourhood matrix doubly lexical. An almost linear time doubly lexical ordering algorithm is given. Doubly lexical orderings unify the orderings characterizing certain classes of matrices and graphs, including totally balanced matrices, subtree matrices and chordal graphs.

Key words. lexical ordering, lexicographic ordering, totally balanced matrices, subtree matrices, chordal graphs, strongly chordal graphs, acyclic hypergraphs, acyclic data bases

AMS(MOS) subject classifications. 05C50, 68R05

1. Introduction.

A *doubly lexical ordering* of a matrix is an ordering of the rows and of the columns of the matrix so that the rows, as vectors, are lexically increasing and the columns, as vectors, are lexically increasing. *Lexical ordering* of vectors is the standard dictionary ordering (often called "lexicographic"), except that vectors will be read from highest to lowest coordinate. Thus row vectors will be compared from right to left, and column vectors from bottom to top. The reason for this "backward and upside down" convention is to relate doubly lexical orderings more conveniently to other previously defined orderings. The concept of doubly lexical orderings arose in a discussion [EKLLS] of [Lu1] and a preliminary version of [HSK].

The first theorem is that every matrix has a doubly lexical ordering. This result, which will be proved in § 2, holds for any matrix with entries from a totally ordered set, but attention will be restricted without loss of generality to matrices of nonnegative integers.

For symmetric matrices, symmetry should be preserved. A matrix M is *symmetric* if its rows and columns are indexed by the same set S and $M(s, t) = M(t, s)$ for all $s, t \in S$. A *symmetric ordering* of such an M is an ordering of S. It is not true that every symmetric matrix has a symmetric doubly lexical ordering. But it will be shown that the symmetric matrix M has a symmetric doubly lexical ordering if it has a *dominant diagonal*, meaning that $M(s, s) \geqq M(s, t)$ for all $s, t \in S$.

The concept of doubly lexical orderings extends to graphs via *neighbourhood matrices*. The *neighbourhood matrix* of a graph is a symmetric 0-1 matrix with rows and columns indexed by the set of vertices of the graph and with an entry of 1 iff the corresponding two vertices are equal or adjacent in the graph. Neighbourhood matrices have dominant diagonals. Thus any graph has a *lexical ordering*, which is an ordering of the vertices making the correspondingly ordered neighbourhood matrix doubly lexical.

* Received by the editors July 23, 1986; accepted for publication (in revised form) December 10, 1986. This work was done at the Universities of Toronto and Waterloo, in Canada, with financial support from the Natural Sciences and Engineering Research Council of Canada. It appeared in preliminary form in the Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 1985.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

The proof of the existence of doubly lexical orderings is algorithmic, but not efficient. Section 3 contains an efficient algorithm to find a doubly lexical ordering of any matrix. The algorithm can be specialized to find symmetric doubly lexical orderings of symmetric matrices with dominant diagonals, and hence to find lexical orderings of graphs. Section 4 contains an implementation of the algorithm which runs in time $O(L \log^2 L)$, $L = e + m + n$, on $m \times n$ matrices of nonnegative integers with e nonzeros. This implementation can be used to find a lexical ordering of a graph in time $O(L \log^2 L)$ where L is the number of edges plus the number of vertices of the graph. An improved (and optimal) $O(L \log L)$ doubly lexical ordering algorithm has recently been developed [PT].

The remaining sections of this paper concern applications of doubly lexical orderings. Doubly lexical orderings will be shown to generalize the characterizing orderings of *totally balanced matrices*, *subtree matrices* and *plaid matrices*: it will be shown that a doubly lexical ordering of a matrix from one of these classes has the ordering properties characteristic of the class. Lexical orderings of graphs will be shown to generalize the characterizing orderings of *chordal graphs* and of *totally balanced graphs* (also called *strongly chordal graphs*). These classes have been studied for their optimization, combinatorial and data base properties.

One advantage of the unification provided by doubly lexical orderings is that theorems proved separately for the various classes can now be proved in a common way using results on the structure of doubly lexical matrices. Proved or reproved in this way will be the theorems characterizing the various classes in terms of orderings. Another advantage is algorithmic: an algorithm to find doubly lexical orderings finds all of the special orderings. Furthermore, as long as the special orderings are easy to recognize—and it will be shown that they can all be recognized in linear time—a doubly lexical ordering algorithm can be used to recognize matrices and graphs in the special classes. The resulting almost-linear-time recognition algorithms are faster than previously known algorithms for 3 of the 5 classes.

A *totally balanced matrix* is a 0-1 matrix with no cycle submatrices. A *cycle matrix* is a 0-1 $n \times n$ matrix, $n \geqq 3$, with exactly two 1's in each row and in each column and such that no proper submatrix has this property. Totally balanced matrices can be characterized as the 0-1 matrices which have Γ*-free orderings* [HSK], [Lu1], [AF]. (These orderings will be defined later.) Intrinsic interest aside, Γ*-free orderings* are useful in optimization problems: For M in Γ*-free order* greedy linear programming algorithms will find 0-1 valued optimum solutions x to the linear programs $\min\{wx: Mx \geqq 1, x \geqq 0\}$ and $\max\{wx: Mx \leqq 1, x \geqq 0\}$ for any w [HSK], [Lu1], [Fa1]. The existence of such 0-1 valued solutions was first proved (for a more general class than totally balanced matrices) in [FHO]. In § 5 it will be proved that any doubly lexical ordering of a totally balanced matrix is Γ*-free.* Doubly lexical orderings will be used in § 6 to give an easy proof of the theorem from [Lu1] that if a matrix M is totally balanced then so is the matrix $M^T \cdot M$, the Boolean matrix product of the transpose of M with M.

A *subtree matrix* is a 0-1 matrix with rows indexed by vertices of a tree T and columns indexed by some subtrees of T and with an entry of 1 iff the corresponding vertex is in the corresponding subtree. The existence of *supported* Γ *orderings* (to be defined) characterizes subtree matrices. They are a superset of totally balanced matrices, retaining some, though not all, of the nice optimization properties. In § 7 it will be proved that a doubly lexical ordering of a subtree matrix is *supported* Γ.

A *plaid matrix* is a totally balanced matrix with no submatrix M_3 where M_3 is a 3×3 0-1 matrix with exactly two 0's, not in the same row or column. One ordering of

M_3 is:

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

These three classes of matrices—totally balanced, subtree, and plaid matrices—have been studied for their properties as acyclic data base schemes under the names "β-acyclic," "α-acyclic" and "γ-acyclic" matrices, respectively [Fg], [BFMY]. Once a matrix is in doubly lexical order it can be tested in linear time whether the matrix is α-, β- or γ-acyclic. To use Fagin's terminology, the "degree of acyclicity" can be found. Furthermore, doubly lexical orderings of these matrices are useful for further data base algorithms [Fg].

A graph is *chordal* (or *triangulated*) if every cycle of 4 or more vertices has a *chord*, which is an edge joining two vertices not consecutive in the cycle. Chordal graphs are a well-studied class of perfect graphs [G], [B], [Du]. They are relevant for Gaussian elimination on sparse symmetric matrices [Ro].

This section concludes with some definitions. Let the matrix M have rows indexed by a set R, columns by a set C. Matrices will be considered to be unordered unless otherwise specified. Note that this is not conventional. Ordering the matrix M means ordering R and C. For pictorial purposes the first row of an ordered matrix goes at the top and the first column at the left. An *entry* of M is a pair (r, c), $r \in R$, $c \in C$. The *value* of the entry (r, c) is the number in row r and column c of M, denoted $M(r, c)$. Matrices will contain only nonnegative integers. A *nonzero entry* will mean an entry whose value is not zero. For $R' \subseteq R$, $C' \subseteq C$, $M(R', C')$ denotes the submatrix of M determined by rows R' and columns C'. A submatrix of an ordered matrix inherits an ordering in the obvious way.

A matrix M will be specified for algorithmic purposes by: listing R and C; listing for each row the nonzero entries in the row; and recording the row, column and value of each nonzero entry. Note that the list of nonzero entries in each column can easily be generated. To specify an *ordered* matrix, order the lists R and C (but not the lists of entries). For 0-1 matrices the nonzero entries are simply the 1's of the matrix. In this case the list of nonzero entries in row $r \in R$ can be thought of as a list of columns $c \in C$ where $M(r, c) = 1$.

Vectors from $\{0, 1\}^E$ will be identified with the subsets of E whose characteristic vectors they are, in order to define *containment* and *intersection* of vectors from $\{0, 1\}^E$. The *clique matrix* of a graph is a 0-1 matrix with rows indexed by the *cliques* of the graph and columns indexed by the vertices, and an entry of 1 iff the corresponding vertex is in the corresponding clique. A *clique* of a graph is a maximal (by containment) subset of vertices of the graph inducing a complete subgraph.

2. Existence.

THEOREM 2.1. *Every matrix has a doubly lexical ordering.*

Associate with the ordered matrix M, whose rows are indexed by $\{1, \cdots, m\}$ and columns by $\{1, \cdots, n\}$, the ordered $m \cdot n$-vector $d(M)$ formed by ordering the entries $M(i, j)$ by $i+j$ and then by j: $d(M) = (M(1, 1), M(2, 1), M(1, 2), \cdots, M(m, n))$. See Fig. 2.1.

CLAIM 2.2. *Exchanging two rows (columns) which are lexically out of order in M causes the vector $d(M)$ to improve lexically.*

Proof. Let $k < l$ index rows of M with row k lexically greater than row l. Let $j \in \{1, \cdots, n\}$ be the largest index for which $M(k, j) \neq M(l, j)$. Then $M(k, j) > M(l, j)$.

$$\begin{bmatrix} d_1 & d_3 & d_6 & \\ d_2 & d_5 & & \cdot \\ d_4 & & \cdot & \\ & \cdot & & d_{m \cdot n} \end{bmatrix}$$

FIG. 2.1. *The vector d.*

After exchanging rows k and l the component of $d(M)$ which was $M(l, j)$ becomes $M(k, j)$ and the components beyond that retain their old values. The symmetric argument works for column. □

Proof of Theorem 2.1. By Claim 2.2 an ordering of a matrix M which makes $d(M)$ lexically largest is a doubly lexical ordering of M. □

For an ordered 0-1 matrix the following property is equivalent to the condition that the columns of the matrix be in lexical order: for any row r with a 1 in some column c and a 0 in some column $k > c$ there is a row $s > r$ with a 0 in column c and a 1 in column k.

Note that there is no restriction on the submatrices of an ordered doubly lexical matrix: If M is any ordered 0-1 matrix then the ordered matrix $\left[\begin{smallmatrix} M & I_2 \\ I_1 & J \end{smallmatrix}\right]$ is doubly lexical, where I_1 and I_2 are appropriately dimensioned identity matrices, and J is a matrix of all 1's.

Consider now the case of symmetric matrices. It is not true that every symmetric matrix has a symmetric ordering which is doubly lexical: An example is $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

THEOREM 2.3. *Every symmetric matrix with a dominant diagonal has a symmetric ordering which is a doubly lexical ordering.*

Recall that the symmetric matrix M with rows and columns indexed by S has a *dominant diagonal* if $M(s, s) \geqq M(s, t)$ for all $s, t \in S$. It is not necessary for a symmetric matrix to have a dominant diagonal in order for it to have a symmetric ordering which is doubly lexical: An example is $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right]$.

Proof of Theorem 2.3. In a symmetric ordered matrix M with a dominant diagonal the exchange of two columns which are lexically out of order leaves the corresponding two rows lexically out of order. Thus by Claim 2.2 the exchange of two columns of M which are lexically out of order and subsequent exchange of the two corresponding rows both lexically improve $d(M)$. A symmetric ordering of M which makes $d(M)$ lexically largest must then be a symmetric doubly lexical ordering of M. □

The neighbourhood matrix of any graph is symmetric and has a dominant diagonal. Thus Theorem 2.3 guarantees that every graph has a lexical ordering.

The condition that the ordering $v_1, v_2, \cdots, v_n$ of the vertices of a graph G be a lexical ordering is equivalent to the following condition, where $\simeq$ indicates adjacent or equal vertices:

For any vertices v_i, v_j, v_k with $i < j$, $v_k \simeq v_i$, $v_k \not\simeq v_j$ there is a vertex v_l, $l > k$ with $v_l \not\simeq v_i$, $v_l \simeq v_j$. ($l = j$ is allowed.)

3. Doubly lexical ordering algorithm. This section contains a doubly lexical ordering algorithm and a proof that the algorithm works. It also contains a modification of the algorithm which finds symmetric doubly lexical orderings of symmetric matrices with dominant diagonals.

An *ordered partition* of a finite set E is an ordered list $(E_1, E_2, \cdots, E_k)$ where the sets E_i, called the *parts*, partition E. The ordered partition $(F_1, F_2, \cdots, F_l)$ is a *refinement* of the ordered partition $(E_1, E_2, \cdots, E_k)$ if there are indices $1 = i_1 < i_2 \cdots <$

$i_{k+1} = l+1$ such that $(F_{i_j}, \cdots, F_{i_{j+1}-1})$ is an ordered partition of E_j. A total ordering of E is *consistent* with an ordered partition of E if it is a refinement of the ordered partition. For a matrix M with ordered partition $(R_1, \cdots, R_k)$ of its row set R, and ordered partition $(C_1, \cdots, C_l)$ of its column set C, each pair (R_i, C_j) determines a *block* $B = M(R_i, C_j)$ of $M \cdot B$ is *constant* if $M(r, c)$ is the same for all $r \in R_i$, and all $c \in C_j$. The *largest entry* in B is $\max(B) \equiv \max\{M(r, c): r \in R_i, c \in C_j\}$ A *splitting row* of B is an $r \in R_i$ s.t. $\{c \in C_j: M(r, c) = \max(B)\}$ is a nonnull proper subset of C_j, and a *splitting column* of B is a $c \in C_j$ s.t. $\{r \in R_i: M(r, c) = \max(B)\}$ is a nonnull proper subset of R_i. Every nonconstant block has a splitting row or a splitting column.

At any stage of the algorithm there is an ordered partition of R and an ordered partition of C. Initially each ordered partition has just one part. The algorithm proceeds by successively refining these ordered partitions, and it ends when every block formed by the ordered partitions is constant. Any row and column orderings consistent with the final ordered partitions provide a doubly lexical ordering (to be proved).

At each stage of the algorithm choose from amongst the blocks formed by the current ordered row and column partitions, a nonconstant block $B = M(R^*, C^*)$ for which all blocks below and/or to the right are constant. Pick a splitting row or a splitting column of B (not both). If a splitting row $r \in R^*$ is chosen, replace C^* in the ordered column partition by the ordered pair $\{c \in C^*: M(r, c) \neq \max(B)\}$, $\{c \in C^*: M(r, c) = \max(B)\}$. Otherwise a splitting column was chosen; split R^* in the analogous way. This completes the description of the algorithm.

A proof that the algorithm works is given below. R. E. Tarjan contributed suggestions for this algorithm. An $O(L \log^2 L)$ implementation of the algorithm will be given in § 4; here $L = e+m+n$ for $m \times n$ matrices of nonnegative integers with e nonzeros. An improved $O(L \log L)$ implementation has recently been given in [PT]. There is an algorithm in [HSK] to produce a doubly lexical ordering of any 0-1 matrix. Its running time is $O(n^2 m)$ on $m \times n$ matrices.

Every doubly lexical ordering of a 0-1 matrix can be realized by the above algorithm: if the 0-1 matrix M is in doubly lexical order to begin with, and splitting rows (columns) are always chosen to be bottom-most (right-most) then the ordering of M is not changed by the algorithm. However, it is not true that every doubly lexical ordering of every matrix can be realized by the algorithm. An example is $\left[\begin{smallmatrix} 2 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. This matrix is doubly lexical, but the algorithm would put the 2 in the bottom right corner.

For a symmetric matrix with a dominant diagonal and with rows and columns indexed by S it is consistent with the above algorithm to keep the row and column partitions identical by following every split in the row partition by the corresponding split in the column partition and vice versa: For if at some stage of the algorithm the row and column partitions are identical and a block $B = M(T, U)$, $T \neq U$ is chosen then any split of the row set T (or the column set U) does not affect the block $M(U, T)$ so it may be chosen next and the corresponding split of the column set T (or the row set U) may be made. On the other hand if a block $B = M(T, T)$ is chosen and a splitting row $t \in T$ is used to split the column set T into T^1, T^2, this splits B into B^1, B^2. The block B^2 may be chosen next. Since M has a dominant diagonal $M(t, t) \geqq M(t, s)$ for all $s \in T$ and so t is in T^2. Thus in B^2 the splitting column t may be used to split the row set T, making the row and column partitions once more identical.

With this modification the algorithm can be used to find symmetric doubly lexical orderings of symmetric matrices with dominant diagonals. This can be done in time $O(L \log^2 L)$ using the implementation in § 4.

In particular the algorithm can be used to find lexical orderings of graphs and this can be implemented in time $O(L \log^2 L)$ where L is the number of edges plus the

number of vertices. The doubly lexical ordering algorithm, acting on the neighbourhood matrix of a graph $G=(V, E)$ may be interpreted as follows:

At each stage of the algorithm there is an ordered partition of V. Initially there is just the single part V. The algorithm proceeds by successively refining this ordered partition, and it ends when each part X of the ordered partition induces a complete subgraph of G, and for each pair of (distinct) parts X, Y either there are no edges or all possible edges between X and Y. Any vertex ordering consistent with the final ordered partition provides a lexical ordering of G.

At each stage of the algorithm choose a part or pair of parts violating the above conditions. A part X can be chosen only if all parts beyond X and all pairs of (distinct) parts Z, W, $Z \geqq X$, $W \geqq X$, satisfy the above conditions. A pair of parts X, Y can be chosen only if all parts $Z \geqq X$, Y and all other pairs of (distinct) parts Z, W, $Z \geqq X$, $W \geqq Y$, satisfy the above conditions. If a part X is chosen there must be a vertex $x \in X$ adjacent to not all of X; replace X in the ordered partition by the ordered pair $\{v \in X: (v, x) \notin E\}, \{x\} \cup \{v \in X: (v, x) \in E\}$. If a pair of parts X, Y is chosen there must either be a vertex $x \in X$ adjacent to some but not all of Y—in which case replace Y in the ordered partition by the ordered pair $\{v \in Y: (v, x) \notin E\}, \{v \in Y: (v, x) \in E\}$—or there must be a vertex $y \in Y$ adjacent to some but not all of X—in which case replace X in the ordered partition by the ordered pair $\{v \in X: (v, y) \notin E\}, \{v \in X: (v, y) \in E\}$. This completes the description of the algorithm working on a graph.

Proof that the algorithm works. Consider two columns of M indexed by $d, e \in C$, with $d < e$ in the final column ordering. It must be shown that, relative to the final row ordering, column d is lexically no greater than column e. This, together with the analogous result for rows, suffices.

By contradiction, suppose that in the final ordered matrix the lowest row t where columns d and e differ satisfies $M(t, d) > M(t, e)$. Since the two columns are not equal, they must be put into different parts of the column partition at some first stage S_1 in which, say, block $B^1 = (R^1, C^1)$ and splitting row $s \in R^1$ are chosen. See Fig. 3.1. Then columns d and e are equal on rows in parts of the row partition below R^1 in stage S_1, and $M(s, d) < \max(B^1) = M(s, e)$. Suppose C^1 is split into $C^{1.1} = \{c \in C^1: M(s, c) < \max(B^1)\}$, $C^{1.2} = C^1 - C^{1.1}$. Since columns d and e differ on row t, and row t is below row s in the final row ordering, therefore $t \in R^1$. Thus $M(t, d) \leqq \max(B^1)$; and so $M(t, e) < M(t, d) \leqq \max(B^1) = M(s, e)$.

Suppose rows s and t are put in different parts of the row partition at some first stage S_2 (after S_1), in which, say, block $B^2 = M(R^2, C^2)$ and splitting column $k \in C^2$

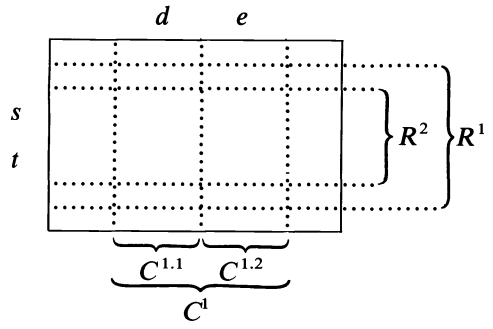


FIG. 3.1

are chosen. Then rows s and t are equal on columns in parts of the column partition to the right of C^2 in stage S_2, and $M(s, k) < \max(B^2) = M(t, k)$. In stage S_1 column k cannot be in a part of the column partition to the right of C^1, else $M(t, k) = M(s, k)$. At the end of stage S_1 $k \notin C^{1.2}$, else $M(s, k) = \max(B^1) \geqq M(t, k)$. Thus at the end of stage S_1 column k is in a part of the column partition to the left of the part of the column partition containing e. This will also be true in the later stage S_2. Therefore $M(s, e) = M(t, e)$. Contradiction. □

4. Implementing the algorithm. Let M be an $m \times n$ matrix of nonnegative integers with rows indexed by R and columns by C. Let e be the number of nonzeros in M, and let $L = m + n + e$. Details of how to implement the doubly lexical ordering algorithm on M in time $O(n + m + e \log^2 L)$ will now be described. It will be assumed that two values from the matrix can be compared in constant time. The matrix will be given to the algorithm as described in the introduction.

The term "part of the column (row) partition" will be abbreviated "column (row, respectively) part." The *block overlay* for block B and value $z \in \mathbb{N}$ consists of the entries of B whose value is z. The *row slice* for row r and column part C^* consists of the nonzero entries (r, c), $c \in C^*$. *Column slice* is defined analogously. The *row (column) slice overlay* for row (column) slice t and value $z \in \mathbb{N}$ consists of the entries of t whose value is z.

It will be assumed that every row and every column has at least one nonzero entry: zero rows and columns must go first in the final ordering, and can be deleted in time $O(L)$. Impose some arbitrary initial ordering on R and C.

The following doubly linked lists will be preserved during the course of the algorithm: an ordered list of row parts and an ordered list of column parts; for each row (column) part a list of blocks ordered by column (row) part, and an ordered list of rows (columns); for each block a list of block overlays ordered by value; for each block overlay a list of row slice overlays ordered by row, and a list of column slice overlays ordered by column; for each row (column) slice overlay a list of nonzero entries ordered by column (row); for each row (column) a list of row (column) slices ordered by column (row) part; for each row (column) slice a list of row (column) slice overlays ordered by value.

Pointers from any list element to its appearance in other lists, and to the lists' owners will also be kept; for example, every row slice overlay will point to the block overlay and the row slice containing it, and to its appearance in their lists. The hierarchy of these constructs is shown in Fig. 4.1. The number of rows (columns) in a row (column) part, and the number of nonzero entries in a row (column) slice overlay will be preserved. A *current block*—the one to be processed next—will also be preserved.

The algorithm as described in § 3 leaves considerable freedom in the choice of the next block to be processed and the choice of splitting rows/columns. In the implementation blocks will be processed in the order of their column parts, then their row parts.

The crucial idea of how to effect a split of a row or column part and account for the cost of doing so is Hopcroft's "relabel the smaller half" idea (applied in this context by Tarjan): when a column part is split into two new column parts $O(\log L)$ steps will be done for each nonzero entry in the new column part with the fewest columns. A nonzero entry can be in the smaller of a pair of column parts formed by a split at most $\log n$ times. The analogous cost accounting applies to row part splits. Thus the total cost of splitting row and column parts over the course of the algorithm is $O(e \log^2 L)$. Note that $\log L$ is $O(\log(\max\{n, m\}))$, since $e \leqq n \cdot m$.

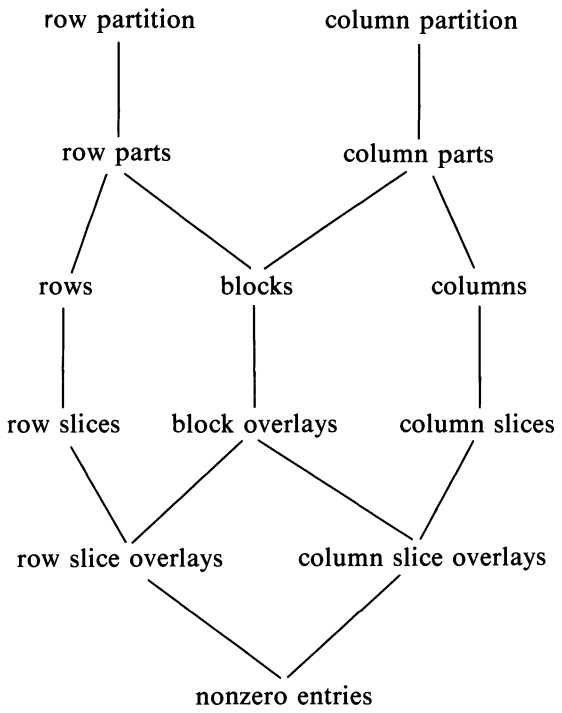


FIG. 4.1

At each stage of the algorithm let $B = M(R^*, C^*)$ be the current block. All blocks below and/or to the right of B will be constant. Suppose R^* has k rows and C^* has l columns. Let V be the last block overlay in B's list of block overlays. Go through V's list of row slice overlays and V's list of column slice overlays alternately looking for a row slice overlay with $l' < l$ entries (which will provide a splitting row), or a column slice overlay with $k' < k$ entries (which will provide a splitting column).

If neither is found B is constant. The current block becomes the block preceding B in C^*'s block list—or, if that does not exist, the last block in the block list of the column part preceding C^*—and if that does not exist the algorithm is done. The cost of discovering B to be constant is accounted for by allowing $O(1)$ steps for each nonzero entry when it is first discovered to be in a constant block, for a total during the course of the algorithm of $O(e)$ steps.

Now suppose that a row slice overlay t with $l' < l$ entries was found. Suppose t belongs to row $r \in R^*$. C^* will be split into $C_1^* = C^* - C_2^*$, $C_2^* = \{c \in C^*$: row slice overlay t has an entry whose column is $c\}$, of $l - l'$ and l' columns, respectively. The cost of finding t is bounded by a constant times the number of row slice overlays in V with l entries. Each such row slice overlay has at least one entry in each of C_1^*, C_2^*. Therefore no matter which of C_1^*, C_2^* is smaller, the cost of finding t is absorbed by the cost of the split.

Denote by $\bar{C}$ the smaller of C_1^*, C_2^*. Suppose $\bar{C}$ has p columns and f nonzero entries. Note $f \geqq p$. Create a new column part to be $\bar{C}$ and modify C^* to become the other of C_1^*, C_2^*. Once $\bar{C}$'s column list is known, C^*'s can be updated by deleting the

columns of $\bar{C}$ in time $O(p)$ which is absorbed by the cost of the split. If $\bar{C} = C_2^*$ then $\bar{C}$'s columns are simply the columns of t's entries. If $\bar{C} = C_1^*$ then it takes $O(l)$ steps to create $\bar{C}$'s column list by deleting the columns of t's entries from a copy of C^*'s column list. $O(p)$ steps disappear into the cost of the split. The other $O(l-p)$ steps are accounted for by allowing $O(1)$ steps for each nonzero entry when it is first put in a block where its row slice is constant (and symmetrically, $O(1)$ steps for each nonzero entry when it is first put in a block where its column slice is constant) for a total during the course of the algorithm of $O(e)$ steps.

The operation of merging will be used to update the remainder of the data structures. N ordered (nonempty) lists containing a total of K elements can be merged in time $O(K \log N)$ by maintaining N elements (originally the first element of each list) in a heap, and successively removing the smallest element a from the heap and adding to the heap a's successor in its list.

To create $\bar{C}$'s list of blocks, merge the column slice lists of the columns in $\bar{C}$ to get a list of column slices of $\bar{C}$ ordered by row part and then by column and create a new block for each group of column slices with a common row part. Each new block must be added to the block list of its row part in the appropriate place. Altogether this takes time $O(f \log p)$. If $\bar{C} = C_2^*$ then the new block with row part R^* becomes the current block.

To create a block overlay list for each new block $\bar{B}$: merge the column slice overlay lists of the column slices in $\bar{B}$ to get a list of column slice overlays of $\bar{B}$ ordered by value and then by column; and create a new block overlay $\bar{V}$ for each group of column slice overlays with a common value. $\bar{V}$'s column slice overlay list is immediately available. Remove each of $\bar{V}$'s column slice overlays from its old block overlay. Summed over all new blocks this takes time $O(f \log p)$.

To create the row slice overlay list for each new block overlay $\bar{V}$: merge the entry lists of the column slice overlays in $\bar{V}$ to get a list of nonzero entries of $\bar{V}$ ordered by row and then by column; and create a new row slice overlay $\bar{t}$ for each group of entries with a common row. $\bar{t}$'s list of entries is immediately available. Remove each of $\bar{t}$'s entries from its old row slice overlay. Summed over all new block overlays this takes time $O(f \log p)$.

Merge the row slice overlay lists of the block overlays in each new block $\bar{B}$ to get a list of the row slice overlays in $\bar{B}$ ordered by row and then by value. For each group of row slice overlays with a common row, create a new row slice $\bar{s}$ with that row slice overlay list. Insert $\bar{s}$ in the appropriate place in its row's list of row slices. Since there are at most f new block overlays and at most f new row slice overlays this last merge operation summed over all new blocks takes time $O(f \log f)$.

This completes the data updates for one step of the algorithm. In effecting the column split $O(\log L)$ time has been spent for each of the f nonzero entries in $\bar{C}$.

5. Totally balanced matrices and Γ-free orderings. A Γ is an ordered 0-1 valued 2×2 matrix with exactly one 0, in the bottom right corner: $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. An ordered 0-1 matrix is Γ-*free* if it has no (ordered) submatrix which is a Γ. A Γ-*free ordering* of a 0-1 matrix is an ordering of the rows and columns which makes the matrix Γ-free.

THEOREM 5.1. *A doubly lexical ordering of a totally balanced matrix is* Γ*-free.*

This is proved by:

THEOREM 5.2. *Let M be an ordered doubly lexical* 0-1 *matrix with rows R and columns C. Any 2×2 submatrix of M formed by $r_1 < r_2 \in R$ and $c_1 < c_2 \in C$ with $M(r_1, c_2) = M(r_2, c_1) = 1$, $M(r_2, c_2) = 0$ is, for some $k \geq 3$, embedded in a $k \times k$ submatrix of M*

formed by $r_1 < r_2 < \cdots < r_k \in R$ *and* $c_1 < c_2 < \cdots < c_k \in C$ *with* $M(r_i, c_{i+1}) = M(r_{i+1}, c_i) = 1$ *for* $i = 1, \cdots, k-1$, $M(r_k, c_k) = 1$, *and* $M(r_i, c_j) = 0$ *for other* i, j *except possibly* $i = j = 1$. *In particular any* Γ *submatrix is embedded in a cycle submatrix. See Fig.* 5.1.

Proof. An inductive construction beginning with $i = 2$: $M(r_i, c_i)$ is 0. Let c_{i+1} be the rightmost column of M where rows r_i and r_{i-1} differ. Since $M(r_{i-1}, c_i) = 1$, $M(r_i, c_i) = 0$, and M is lexical, c_{i+1} exists, $c_{i+1} > c_i$, and $M(r_{i-1}, c_{i+1}) = 0$, $M(r_i, c_{i+1}) = 1$. For $j = i, \cdots, 3$ consecutively, since c_j was by induction the rightmost column where r_{j-1} and r_{j-2} differed, $M(r_{j-1}, c_{i+1}) = 0$ implies $M(r_{j-2}, c_{i+1}) = 0$.

Let r_{i+1} be the bottom-most row of M where columns c_i and c_{i-1} differ. Since $M(r_i, c_{i-1}) = 1$, $M(r_i, c_i) = 0$, and M is lexical, r_{i+1} exists, $r_{i+1} > r_i$, and $M(r_{i+1}, c_{i-1}) = 0$, $M(r_{i+1}, c_i) = 1$. For $j = i, \cdots, 3$ consecutively, since r_j was by induction the bottom-most row where c_{j-1} and c_{j-2} differed, $M(r_{i+1}, c_{j-1}) = 0$ implies $M(r_{i+1}, c_{j-2}) = 0$. If $M(r_{i+1}, c_{i+1})$ is 1 the theorem is proved with $k = i+1$, and otherwise the construction applies once more. □

Observation 5.3. If a 0-1 matrix has a cycle submatrix then for any ordering of the matrix there is a Γ submatrix (formed by a topmost, leftmost 1 of the cycle submatrix; the other 1 in its row in the cycle; and the other 1 in its column in the cycle).

Observation 5.3, Theorem 5.1, and the existence of doubly lexical orderings imply:

THEOREM 5.4 ([HSK], [Lu1], [AF]). *A* 0-1 *matrix has a* Γ*-free ordering iff it is totally balanced.*

The definition of totally balanced matrices is a coNP description: it is easy to verify, given a cycle submatrix, that a matrix is *not* totally balanced. The existence of a Γ-free ordering is an NP description. So Theorem 5.4 gives the equivalence of an NP and a coNP description of the class of totally balanced matrices. Good recognition algorithms will be discussed below.

COROLLARY 5.5 ([BK], [AF], [D'AM]). *Every totally balanced matrix* M *has a row* r *with the property that the columns* c *with* $M(r, c) = 1$ *are totally ordered by containment.*

Proof. The first row of a Γ-free matrix has this property. □

The above results apply to graphs: Farber [Fa2] defined a graph to be *strongly chordal* if it has a *strong elimination ordering*, which is an ordering of the vertices making the neighbourhood matrix of the graph Γ-free. Strongly chordal graphs are a

	c_1	c_2	c_3	c_4			c_i			c_k	
r_1	?	1	0	0							
r_2	1	0	1	0							
r_3	0	1	0	1			0				
r_4	0	0	1	0	·						
				·	·	·					
					·	0	1				
r_i						1	0	·			
			0				·	·	·		
								·	0	1	
r_k									1	1	

FIG. 5.1

class of graphs for which the dominating set problem is well solved [Fa1]. Theorem 5.1 implies:

THEOREM 5.6. *A lexical ordering of a graph with a totally balanced neighbourhood matrix is a strong elimination ordering.*

Observation 5.3, Theorem 5.6 and the existence of lexical graph orderings provide a new proof of:

THEOREM 5.7 ([Fa2]). *A graph is strongly chordal iff its neighbourhood matrix is totally balanced.*

On the other hand, results on strongly chordal graphs can be used to give an alternative proof of Theorem 5.4; see [AF].

Totally balanced matrices are a subclass of subtree matrices. Lehel [Le] gives an interesting characterization of them in terms of special families of subtrees of trees.

Totally balanced matrices are also a subclass of *balanced matrices*, defined by Berge [B] to be 0-1 matrices without $n \times n$ cycle submatrices for n odd. It is a major open problem to find a polynomial time recognition algorithm for balanced matrices, or to find a balanced matrix analogue of Theorem 5.4, which would put the class of balanced matrices in **NP**. Balanced matrices are significant for the existence of integral optimal solutions to various linear programs involving them [FHO], [B]. In particular the linear programs $\max\{wx: Mx \leqq 1, x \geqq 0\}$ and $\min\{wx: Mx \geqq 1, x \geqq 0\}$ have 0-1 valued optimum solution vectors for any w and any balanced matrix M. For totally balanced matrices these existence results have been realized by polynomial time algorithms [Lu1], [HSK], [Fa1] which use in a crucial way a Γ-free ordering.

The combinatorial optimization properties of balanced matrices were applied by Berge to graphs with balanced clique matrices, which he called *balanced graphs*. Similarly a graph is *totally balanced* if its clique matrix is totally balanced. A consequence of a result in the following section is that totally balanced and strongly chordal graphs are the same.

Totally balanced matrices also arise in the study of relational data bases [Fg]. In this context they are called "β-acyclic hypergraphs."

The remainder of this section is about recognizing totally balanced matrices. To test whether a 0-1 matrix is totally balanced apply the doubly lexical ordering algorithm and then test if the resulting ordered matrix is Γ-free. It will be shown that any ordered 0-1 $m \times n$ matrix M with e 1's can be searched for Γ's in time $O(L)$, $L = e + m + n$. Thus totally balanced matrices can be recognized in time $O(L \log^2 L)$.

Previous algorithms to recognize totally balanced matrices [Fg], [Lu1], [HSK], [AF] had time bounds worse than $O(L \log^2 L)$. The algorithm in [HSK] took time $O(n^2 m)$ on $m \times n$ matrices and the others took at least as long.

An ordered 0-1 matrix will be specified for algorithmic purposes as described in the introduction.

It is not possible to test all the potential Γ's in an ordered 0-1 matrix in time $O(L)$ because there are too many of them. The following lemma shows that it suffices to test a set of e potential Γ's having distinct upper left 1's.

LEMMA 5.8. *If an ordered* 0-1 *matrix M has a Γ submatrix then it has a Γ submatrix formed by rows $r_1 < r_2$ and columns $c_1 < c_2$ such that $M(r_1, c) = 0$ for all c, $c_1 < c < c_2$, and $M(r, c_1) = 0$ for all r, $r_1 < r < r_2$.*

Proof. Choose in M a Γ formed by rows $r_1 < r_2$ and columns $c_1 < c_2$ which minimizes $s(r_1, r_2, c_1, c_2) \equiv |\{c \in C: c_1 < c < c_2\}| + |\{r \in R: r_1 < r < r_2\}|$. If there is a $c \in C$, $c_1 < c < c_2$ with $M(r_1, c) = 1$ then if $M(r_2, c) = 0$, a Γ with smaller s is formed by r_1, r_2, c_1, c, and otherwise $M(r_2, c) = 1$ and a Γ with smaller s is formed by r_1, r_2, c, c_2. Similarly any $r \in R$, $r_1 < r < r_2$ satisfies $M(r, c_1) = 0$. □

In an ordered 0-1 matrix the *right neighbour* of a 1 is the first 1 to the right in the same row. *Bottom neighbours* are defined similarly. To test if an ordered 0-1 matrix has a Γ it suffices by Lemma 5.8 to test for each 1 in the matrix whether it forms a Γ together with its right neighbour and its bottom neighbour. Right neighbours can be identified in time $O(L)$ as follows: go through the columns from last to first; for each column go through its list of 1's; keep a record of the current leftmost 1 in each row; then when a new 1 is examined its right neighbour is the current leftmost 1 in its row. Bottom neighbours can be found in the same way. Using right and bottom neighbours create a list l_c for each column c of the rows r for which $M(r, c)$ must be 1 in order for the matrix to be Γ-free. To test this create an m-vector z initialized to 0. For each column c: Set $z(r)$ to 1 for each row r in c's list of 1's; test if $z(r)$ is 1 for each row r in the list l_c; then reset $z(r)$ to 0 for each row r in c's list of 1's. This can all be done in time $O(L)$.

A Γ in a 0-1 doubly lexical matrix can be expanded to a cycle submatrix as in the proof of Theorem 5.2 in time $O(L)$: In order to find, given two columns c_1 and c_2 the last row r where $M(r, c_1) = 0$ and $M(r, c_2) = 1$, use an m-vector z initialized to 0. Set $z(r)$ to 1 for each row r in c_1's list of 1's; go through the rows in c_2's list of 1's finding the maximum row, r, for which $z(r) = 0$; then reset $z(r)$ to 0 for each row r in c_1's list of 1's. The analogous thing can be done for a pair of rows. In the construction given in the proof of Theorem 5.2 it is also necessary to test given r and c whether $M(r, c)$ is 0 or 1. Since this test is needed at most once for each c it can be done by going through c's list of 1's.

To test whether a graph has a strong elimination ordering find a symmetric doubly lexical ordering of the neighbourhood matrix and search for Γ's as described above. Farber [Fa2] contains a polynomial time algorithm to recognize strongly chordal graphs. No running time is given. The present algorithm is faster than the natural implementation of that algorithm.

6. The Boolean power of a totally balanced matrix.

THEOREM 6.1 ([Lu1]). *If the matrix M is totally balanced then so is $M^T \cdot M$, the Boolean matrix product of the transpose of M with M.*

The theory of doubly lexical orderings provides a simple proof of this result since by Theorems 5.1 and 5.4 it suffices to prove:

LEMMA 6.2. *If the ordered* 0-1 *matrix M is doubly lexical and Γ-free then $M^T \cdot M$ is Γ-free.*

Proof. By contradiction. Suppose $M^T \cdot M$ has a Γ formed by rows $a < b$ and columns $c < d$, $a, b, c, d \in C$, the set of column indices of M. Assume without loss of generality $a \leqq c$. Since the entry in row b and column c of $M^T \cdot M$—denoted $(M^T \cdot M)(b, c)$—is 1, M has a bottom-most row r_1 with $M(r_1, b) = M(r_1, c) = 1$. $M(r_1, d) = 0$ since $(M^T \cdot M)(b, d) = 0$.

$(M^T \cdot M)(a, c) = 1 = (M^T \cdot M)(a, d)$ implies that M has a row with 1's in columns $a \leqq c$ and a row with 1's in columns $a < d$. This gives the configuration

a	c	d
1	1	?
1	?	1

where the two rows may be in either order. Some row, call it r_2, must have 1's in a, c, and d, else a Γ is forced in M. $M(r_2, b) = 0$, since $(M^T \cdot M)(b, d) = 0$. Now $r_1 < r_2$, else rows r_2, r_1 and columns c, d form a Γ in M. Then $b < c$, else columns c, b and rows r_1, r_2 form a Γ in M. Since M is doubly lexical and $M(r_2, a) = 1$, $M(r_2, b) = 0$,

$a < b$, M has a row $r_3 > r_2$ with $M(r_3, a) = 0$, $M(r_3, b) = 1$. Row r_1 was the bottom-most row with 1's in columns b and c, and $r_1 < r_3$ so $M(r_3, c) = 0$. But then columns b, c and rows r_1, r_3 form a Γ in M. Contradiction. □

Both hypotheses of Lemma 6.2 are necessary: in Fig. 6.1 M is doubly lexical but $M^T \cdot M$ is not Γ-free; and in Fig. 6.2 M is Γ-free but $M^T \cdot M$ is not. Nor is it true that M doubly lexical and Γ-free implies $M^T \cdot M$ doubly lexical; see Fig. 6.3.

Frank [Fr1] proved the special case of Theorem 6.1 when the matrix M has rows indexed by vertices of a directed rooted tree T and columns indexed by directed paths in T and an entry of 1 iff the corresponding vertex is in the corresponding path. It is easy to show that such a matrix M is totally balanced.

Theorem 6.1 has application to graphs: For a graph G with neighbourhood matrix N, the *k*th-*power* of G is the graph whose neighbourhood matrix is N^k, the kth Boolean power of N. The kth-power of G has the same vertex set as G and an edge between any two vertices joined by a path of k or fewer edges in G.

THEOREM 6.3. *Let G be a graph with neighbourhood matrix N, and let C_k be the clique matrix of the kth power of G. If C_1 is totally balanced then so is N. If N is totally balanced then so is the matrix $[N N^2 \cdots N^k C_1^T \cdots C_k^T]$ for any k.*

LEMMA 6.4 ([An]). *If the matrix M with columns c_1 and c_2 (among others) is totally balanced then so is the matrix formed by adding to M a new column c where $M(r, c) = M(r, c_1) \cdot M(r, c_2)$. c is called the intersection of c_1 and c_2.*

Proof. By virtue of Theorem 5.4 this is proved by the observation that if M is ordered to be Γ-free and $c_1 < c_2$ then adding c just before c_1 makes a Γ-free matrix. □

Proof of Theorem 6.3. Suppose C_1 is totally balanced. $N = C_1^T \cdot C_1$ so by Theorem 6.1 N is totally balanced.

Suppose N is totally balanced. Then so is $[I N]$. Suppose by induction that $[I N N^2 \cdots N^k]$ is totally balanced. Then by Theorem 6.1 the following matrix is

M

a	b	c	d		
0	1	1	0	1	0
1	0	1	1	0	1
1	0	0	0	1	1
0	1	0	0	1	1
0	0	1	0	1	1
0	0	0	1	1	1

$M^T \cdot M$ contains

	c	d
c	1	1
b	1	0

FIG. 6.1

M

a	b	c	d
0	1	1	0
1	0	1	1

$M^T \cdot M$ contains

	c	d
a	1	1
b	1	0

FIG. 6.2

M

a	b	c
1	1	0
0	0	1
0	1	1

$M^T \cdot M$

	a	b	c
a	1	1	0
b	1	1	1
c	0	1	1

FIG. 6.3

totally balanced:

$$\begin{bmatrix} I \\ N \\ \vdots \\ N^k \end{bmatrix} [IN \cdots N^k] = \begin{bmatrix} * & * & & * \\ N & N \cdot N & \cdots & N \cdot N^k \\ \vdots & \vdots & & \vdots \\ * & * & & * \end{bmatrix}.$$

Therefore $[I\,N\,N^2 \cdots N^{k+1}]$ is totally balanced.

It remains to prove that for any graph G with clique matrix C and neighbourhood matrix N, and for any matrix M with rows index by vertices of G, $[MN]$ totally balanced implies $[MNC^T]$ totally balanced. This follows from Lemma 6.4 since a column of C^T indexed by clique c is the intersection of the columns of N indexed by vertices in c. □

COROLLARY 6.5 ([Lu1], [Fa2]). *Totally balanced and strongly chordal graphs are the same.*

COROLLARY 6.6 ([Lu1]). *Powers of totally balanced graphs are totally balanced.*

This was conjectured in [CN].

In a graph $G=(V, E)$ a *star* with radius r and centre $v \in V$ consists of the vertices of G within distance r of v. A *star matrix* of a graph is a 0-1 matrix with rows indexed by some stars of the graph and columns by vertices, and a 1 exactly when a vertex is in a star. A star matrix of a graph G with neighbourhood matrix N is a submatrix of the transpose of $[INN^2 \cdots N^k]$ for some k, so Theorem 6.3 implies:

COROLLARY 6.7. *Any star matrix of a totally balanced graph is totally balanced.*

This was proved in [Gs] for trees. (Note that trees are obviously totally balanced.)

Theorem 6.1 then implies:

COROLLARY 6.8. *Any* 0-1 *matrix with rows and columns indexed by stars of a totally balanced graph and a* 1 *exactly when the two stars intersect is totally balanced.*

This was proved for trees in [Tm].

Polynomial time algorithms for finding integral optimal solutions to certain linear programs involving totally balanced matrices provide, by virtue of the above results, polynomial time algorithms to solve a variety of optimization problems on totally balanced graphs. A main instance is the R-domination problem: given a graph G with vertex set V, weights $w: V \to \mathbb{R}$ and numbers $R: V \to \mathbb{N}$, find a minimum weight subset S of V such that for every $v \in V$ there is a $u \in S$ within distance $R(v)$ of v. This problem can be expressed as the integer linear program $\min\{wx: Mx \geqq 1, x \geqq 0, x \text{ integer-valued}\}$, where M is a star matrix of G. As mentioned in § 5 there are polynomial time algorithms in [Lu1], [HSK] and [Fa1] to solve integer linear programs like the one above for any totally balanced matrix M. These algorithms solve the R-domination problem on totally balanced graphs in polynomial time. Further discussion of the R-domination problem can be found in [Lu1]. Prior to this work, good algorithms had been available to solve the R-domination problem on trees with w identically 1 [S]; on totally balanced graphs with R identically 1 [Fa1]; and on totally balanced graphs with R constant and w identically 1 [CN].

7. Subtree matrices. The first lemma in this section characterizes subtree matrices in terms of the existence of *supported* Γ *orderings*. This characterization and the definition are both **NP** descriptions. After that results analogous to those in § 5 will be given: it will be shown that a doubly lexical ordering of a 0-1 matrix with no *fence* submatrix is *supported* Γ; and as a consequence that a 0-1 matrix is supported Γ iff it has no *fence* submatrix. This provides a co**NP** description of the class.

An ordered 0-1 matrix M is *supported* Γ if for every pair $r_1 < r_2$ of rows and pair $c_1 < c_2$ of columns which form a Γ there exists a row $r_3 > r_2$ with $M(r_3, c_1) = M(r_3, c_2) = 1$. r_3 *supports* the Γ. Matrices with supported Γ orderings generalize totally balanced matrices but not within the class of balanced matrices.

LEMMA 7.1. *Subtree matrices are exactly the matrices with supported* Γ *orderings.*

Proof. Let M be a subtree matrix for tree T. Pick a vertex r of T and order the vertices of T by decreasing distance from r. Break ties arbitrarily. Also order the subtrees of T by decreasing distance from r. The distance of a subtree from r is the minimum distance of a vertex of the subtree from r. This is a supported Γ ordering of M, for suppose vertices $v_1 < v_2$ and subtrees $t_1 < t_2$ form a Γ in M: Let r_i be the vertex of t_i closest to r $(i = 1, 2)$. Then $r_1 \geqq v_2$. Since $t_1 < t_2$, r_2 is at least as close to r as r_1. But t_1 and t_2 contain a common vertex v_1. Thus r_1 is in t_2, and supports the Γ.

If the ordered $m \times n$ matrix M is supported Γ, create a tree T on vertex set $\{1, \cdots, m\}$ by setting for $i = 1, \cdots, m-1$, $f(i) = \min\{k: M(i, k) = 1$ and there exists $j > i$, $M(j, k) = 1\}$, and $b(i) = \max\{j: M(j, f(i)) = 1\}$ and creating the edge $(i, b(i))$ if $f(i)$ exists and (i, m) otherwise. This makes a tree since $b(i) > i$ when it exists. To show that each column of M is the incidence vector of a subtree of T it suffices to show that $M(i, k) = 1 = M(j, k)$, $i < j$ implies $M(b(i), k) = 1$. If this were not true then $f(i) < k$, and rows $i, b(i)$ and columns $f(i), k$ would form an unsupported Γ. □

An *identity complement* matrix is a 0-1 $n \times n$ matrix, $n \geqq 3$, with exactly one 0 in each row and in each column. A submatrix M' of the 0-1 matrix M is *column induced* if for every row r of M there is a row r' of M' with $M(r, c) \leqq M(r', c)$ for every column c of M', i.e., M' is formed by deleting some columns and then deleting from the resulting submatrix rows which are contained in other rows. A column induced submatrix which is a cycle or an identity complement will be called a *fence submatrix.* For example in Fig. 7.1 the submatrix formed by the last three rows and the last three columns is a 3×3 identity complement (this is the same as a 3×3 cycle) which is *not* column induced; but the submatrix formed by rows 1 through 4 and columns 2 through 5 is a fence submatrix.

	1	2	3	4	5
1	0	0	1	1	1
2	0	1	1	0	1
3	0	1	0	1	1
4	1	1	1	1	0
5	1	0	1	0	1

FIG. 7.1

THEOREM 7.2. *If an ordered doubly lexical* 0-1 *matrix has an unsupported* Γ *then it has a fence submatrix.*

Proof. Let M be an ordered 0-1 doubly lexical matrix. Suppose that columns $c_1 < c_2$ and rows $r_1 < r_2$ form an unsupported Γ—call it G—with c_1 rightmost, then c_2 rightmost, then r_1 and r_2 bottom-most. Row r_2 is then the bottom-most row with a 1 in column c_1. As in the proof of Theorem 5.2 construct a cycle submatrix K of M with rows $r_1 < r_2 < \cdots < r_k$ and columns $c_1 < c_2 < \cdots < c_k$, $k \geqq 3$, $M(r_i, c_{i+1}) = M(r_{i+1}, c_i) = 1$ for $i = 1, \cdots, k-1$, $M(r_1, c_1) = M(r_k, c_k) = 1$, $M(r_i, c_j) = 0$ for other i, j,

and where c_i for $i=3, \cdots, k$ is the rightmost column where rows r_{i-1} and r_{i-2} differ, and r_i for $i=3, \cdots, k$ is the bottom-most row where columns c_{i-1} and c_{i-2} differ. By induction on $i=2, \cdots, k$, r_i is the bottom-most row of M with a 1 in column c_{i-1}.

If K is column induced the theorem is proved; otherwise there is a row r^* of M such that for every r_i, $i \in \{1, \cdots, k\}$ there is a $c_{j(i)}, j(i) \in \{1, \cdots, k\}$ with $M(r^*, c_{j(i)}) = 1$, $M(r_i, c_{j(i)}) = 0$. Let c_l be the leftmost column of K with a 1 in row r^*. Row r_{l+1} has a 1 in column c_l and lies below r^*. Rows r^*, r_{l+1} and columns $c_l, c_{j(l+1)}$ form an unsupported Γ. See Fig. 7.2. By the choice of G, l is 1. $M(r^*, c_j)$ is 0 for all $j \geqq 4$, else r^*, r_2 and c_1, c_j make a better choice for G. Then $M(r^*, c_2) = M(r^*, c_3) = 1$. k is 3, otherwise r^*, r_3 and c_2, c_3 make a better choice for G.

Rename $s_1 \leftarrow r_3$, $s_2 \leftarrow r_2$, $s_3 \leftarrow r_1$. Rows s_3, s_2, s_1 and columns c_1, c_2, c_3 form an identity complement submatrix which is not column induced. Larger identity complement submatrices will be constructed until a column induced one is reached. Note that for $c > c_3$, $M(s_3, c) = 1$ implies $M(s_1, c) = 1$; otherwise s_3, s_1, c_2, c make a better choice for G. Recall that c_3 is the rightmost column where s_2 and s_3 differ and s_3 is the lowest row with 1's in c_1 and c_2.

Begin an inductive construction with $i=3$. See Fig. 7.3. There is a row with 1's in columns $c_1, \cdots, c_i$. Let s_{i+1} be the bottom-most such. Since, by induction, s_i was the bottom-most row with 1's in columns $c_1, \cdots, c_{i-1}$, and $M(s_i, c_i)$ is 0, therefore $s_{i+1} < s_i$. Let c_{i+1} be the rightmost column where rows s_{i+1} and s_i differ. Since $M(s_{i+1}, c_i) = 1$, $M(s_i, c_i) = 0$ and M is doubly lexical, c_{i+1} exists, $c_{i+1} > c_i$ and

	c_l	$c_{j(l+1)}$
r^*	1	1
r_{l+1}	1	0
	0	
	.	
	0	

FIG. 7.2

	c_1	c_2	c_3	c_4			c_i	c_{i+1}
s_{i+1}	1	1	1	1	·	·	1	0
s_i	1	1	1	1			0	1
.	.							
.	.							
s_4	1	1	1	0			1	1
s_3	1	1	0	1			1	1
s_2	1	0	1	1			1	1
s_1	0	1	1	1			1	1

FIG. 7.3

$M(s_{i+1}, c_{i+1}) = 0$, $M(s_i, c_{i+1}) = 1$. For $j = i+1, i, \cdots, 4$ consecutively, since c_{j-1} was by induction the rightmost column where rows s_{j-1} and s_{j-2} differed, $M(s_{j-1}, c_{i+1}) = 1$ implies $M(s_{j-2}, c_{i+1}) = 1$. Then $M(s_1, c_{i+1}) = 1$ too. Rows $s_{i+1}, \cdots, s_1$ and columns $c_1, \cdots, c_{i+1}$ form an identity complement. If it is column induced the theorem is proved, and otherwise the construction applies once more. □

LEMMA 7.3. *If a* 0-1 *matrix M has a fence submatrix then any ordering of the matrix has an unsupported* Γ.

Proof. Order *M*.

Suppose M has a column induced identity complement submatrix M'. Let c be the leftmost column of M' and let r be the bottom-most row of M with a 1 in column c. There must be a column c' of M' with $M(r, c') = 0$, and a row r' of M' with $M(r', c) = M(r', c') = 1$. Then $r > r'$ and r, r', c, c' form an unsupported Γ in M.

Suppose M has a column induced cycle submatrix with columns C'. Let R' be the bottom-most rows making a cycle submatrix M' with columns C'. Choose a topmost leftmost 1 of M'—suppose it is in position (r, c). There is a row $r' \in R'$, $r' > r$ with $M(r', c) = 1$, and a column $c' \in C'$, $c' > c$ with $M(r, c') = 1$. Then $M(r', c') = 0$, and r, r', c, c' form a Γ. This Γ is unsupported, for if row $s > r'$ supports the Γ, then since M' is column induced, s is identical to r on C', and would have been preferred over r. □

Theorem 7.2 and Lemma 7.3 together with the existence of doubly lexical orderings imply:

THEOREM 7.4. *A* 0-1 *matrix has a supported* Γ *ordering iff it has no fence.*

Theorem 7.2, Lemma 7.3, and Lemma 7.1 imply:

THEOREM 7.5. *A doubly lexical ordering of a subtree matrix is supported* Γ.

COROLLARY 7.6 ([BFMY], [BEW]). *Every subtree matrix M has a column with a single* 1 *or a row contained in another row.*

Proof. If the first column c of M in supported Γ order has more than one 1 then the last row with a 1 in column c contains the other rows with a 1 in column c. □

Subtree matrices retain some, though not all of the integrality properties of totally balanced matrices: the linear program $\max\{wx: Mx \leqq 1, x \geqq 0\}$ has 0-1 valued optimum solution vectors for any w and any subtree matrix M. This result can be derived from a result on chordal graphs in [Fr2]. Direct algorithmic proofs can be found in [BEW] (using the subtree characterization) and [Lu2] (using supported Γ orderings).

Subtree matrices have been studied in the context of relational data-base schemes under the name "acyclic hypergraphs" or "α-acyclic hypergraphs" [BFMY], [Fg], [Ma], [TY], [D'AM]. In [BFMY] the authors prove the equivalence for a 0-1 matrix of the following three conditions (among others more relevant to data-bases). The equivalence of conditions (1) and (3) was previously proved by Duchet [Du]. The equivalence of conditions (1) and (2) is shown in [BEW].

(1) M is a subtree matrix.

(2) There is some sequence of applications of the following two operations which reduces M to nothing:

(A) Delete a column containing a single 1.

(B) Delete a row contained in another row.

(3) M satisfies the *Helly property for columns* and its *intersection graph* is chordal.

A 0-1 matrix M with rows R and columns C satisfies the *Helly property for columns* if for any $C' \subseteq C$ such that every pair of columns in C' intersect there is a row $r \in R$ such that $M(r, c) = 1$ for all $c \in C'$. The *intersection graph* of a 0-1 matrix is a graph with a vertex for each column of the matrix and an edge between two vertices exactly when the corresponding two columns intersect.

Condition 3 ties together the theory of subtree matrices with that of chordal graphs. In fact the equivalence of conditions 2 and 3 is proved in [BFMY] using a classic **NP** characterization of chordal graphs [FG] (see Theorem 9.1 in this paper).

Condition 2 is equivalent to the existence of a supported Γ ordering: row and column orderings which are supported Γ are the same as deletion orderings produced by applying operations (A) and (B). Condition 3 is equivalent to the absence of fence submatrices: chordless cycles of the intersection graph of M correspond to column induced cycle submatrices of M, and minimal sets of columns of M violating the Helly property correspond to column induced identity complement submatrices. Thus Theorem 7.4 can be derived from known results. Alternatively, the equivalence of conditions 1, 2 and 3 can be derived from the present results.

The remainder of this section is about recognizing subtree matrices. This can be done in time $O(L \log^2 L)$ using the doubly lexical ordering algorithm together with an $O(L)$ algorithm (below) to test if an ordered 0-1 matrix is supported Γ. This method of recognizing subtree matrices is not as fast as the linear time algorithm of [TY], which will be described below, but has the advantage that once a matrix is in doubly lexical order it is possible to test in linear time whether it is a subtree matrix, whether it is totally balanced, and whether it is a plaid matrix. Using Fagin's data-base terminology, the "degree of acyclicity" can be found.

An ordered 0-1 matrix will be specified for algorithmic purposes as described in the introduction.

The following lemma restricts the set of potential unsupported Γ's which must be examined in testing whether an ordered matrix is supported Γ.

LEMMA 7.7. *If an ordered* 0-1 *matrix* M *with rows* R *and columns* C *has an unsupported* Γ *then it has one formed by rows* $r_1 < r_2$ *and columns* $c_1 < c_2$ *such that the following* 2 *conditions hold.* (See Fig. 7.4.)

$$M(r, c_1) = 0 \quad \forall r \in R, \qquad r > r_2,$$

$$M(r, c) = 0 \quad \forall r \in R, \quad c \in C \text{ s.t. } r > r_1, \quad c < c_1, \quad M(r_1, c) = 1.$$

Proof. Let the Γ formed by rows $r_1 < r_2$ and columns $c_1 < c_2$ be an unsupported Γ of M with c_1 as far to the left as possible and, subject to this, r_2 as far down as possible, and then r_1 as far down as possible. This Γ satisfies the conditions of the lemma, for suppose not: If there is an $r > r_2$ such that $M(r, c_1) = 1$ then since the Γ is unsupported $M(r, c_2)$ must be 0 and replacing r_2 by r makes an unsupported Γ with a lower bottom row, a contradiction. So $M(r, c_1) = 0$ for all $r > r_2$. If there is a $c < c_1$ such that $M(r_1, c) = 1$ and an $r > r_1$ such that $M(r, c) = 1$ then let r be the lowest such. If $r > r_2$

	c	c_1	c_2
r_1	1	1	1
	0		
	·		
r_2	·	1	0
	·	0	
	·	·	
	0	0	

FIG. 7.4

then $M(r, c_1) = 0$ so r_1, r, c, c_1 form an unsupported Γ with first column further to the left than the original, a contradiction. Similarly if $r = r_2$ then r_1, r_2, c, c_2 form a better unsupported Γ, and if $r < r_2$ then if $M(r, c_1)$ or $M(r, c_2)$ is 0 then r_1, r, c, and c_1 or c_2 form a better unsupported Γ, and if $M(r, c_1) = M(r, c_2) = 1$ then r, r_2, c_1, c_2 form a better unsupported Γ. □

Note that the potential Γ's described in the lemma have distinct upper right 1's. The *base* of a 1 in M is the bottommost 1 in the same column. The base of each 1 can be identified in time $O(L)$ by going through the 1's list of each column to find the bottommost 1. The *trunk* of a 1 in M is defined to be the leftmost 1 in the same row whose base is not itself. The trunk of each 1 in M can be identified in time $O(L)$ by going through the 1's list of each row to find the leftmost 1 whose base is not itself. By Lemma 7.7 M is supported Γ iff there is no Γ in M formed by a 1, its trunk t, and t's base. Using trunks and bases a list l_c can be created for each column c of the rows for which $M(r, c)$ must be 1 in order for the matrix to be supported Γ. These lists can be tested in time $O(L)$ exactly as described in § 5: create an m-vector z initialized to 0. For each column c: set $z(r)$ to 1 for each row r in c's list of 1's; test if $z(r)$ is 1 for each row r in the list l_c; then reset $z(r)$ to 0 for each row r in c's list of 1's.

If an unsupported Γ is found in a doubly lexical 0-1 matrix the construction given in the proof of Theorem 7.2 can be used to find a fence submatrix in polynomial time. Implementing this construction in linear time seems hard.

Tarjan and Yannakakis [TY] gave a linear time subtree matrix recognition algorithm which orders any 0-1 matrix and then tests for a property that characterizes subtree matrices. Although they do not use the idea of supported Γ's it can be proved that their algorithm does in fact produce supported Γ orderings of subtree matrices. Their ordering algorithm is:

Maximum cardinality search on a matrix. To order the matrix number the rows from m to 1 and the columns from n to 1. At any stage number an as-yet-unnumbered row with 1's in as many numbered columns as possible, and then number (in any order) the as-yet-unnumbered columns with 1's in this row.

This algorithm does not produce doubly lexical orderings even of subtree matrices, nor does it produce Γ-free orderings of totally balanced matrices. For example the ordered matrix

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

can result from the [TY] algorithm; it is totally balanced; but it is not Γ-free and not doubly lexical.

8. Plaid matrices. A *weave ordering* of a 0-1 matrix is an ordering of the rows and columns such that for any *bottom right submatrix*—i.e., any submatrix formed by taking all rows below some row and all columns to the right of some column—either all the columns with a 1 in the first row are consecutive and identical or all the rows with a 1 in the first column are consecutive and identical. For example the matrix in Fig. 8.1 is not weave ordered because of the submatrix formed by deleting the first row and the first column. A weave ordering is a Γ-free ordering.

THEOREM 8.1. *A doubly lexical ordering of a plaid matrix is a weave ordering.*

Proof. Let M be a doubly lexical ordered plaid matrix with rows $1, \cdots, m$ and columns $1, \cdots, n$. By Theorem 5.1 M is Γ-free. It must be proved that either the rows of M with 1's in the first column are identical (hence consecutive in a doubly lexical matrix) or the columns of M with 1's in the first row are identical (hence consecutive).

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

FIG. 8.1

Case 1. $M(1,1)=1$. Suppose that the columns with 1's in row 1 are not identical. Then there is a column k with 1 in row 1, not identical with column 1. Also suppose that the rows with 1's in column 1 are not identical. Then there is a row r with 1 in column 1, not identical with row 1. $M(r,k)=1$ otherwise a Γ is formed. Since columns 1 and k are not identical there is some last row s with 0 in column 1, 1 in column k. It will be shown that $s>r$. Similarly there is some last column l with 0 in row 1, 1 in row r; and $l>k$. See Fig. 8.2. But then $M(s,l)$ must be 1 (otherwise as Γ is formed by r, s, k, l), and then the three rows and columns form an M_3 submatrix.

Consider now the possibility that $s<r$. Since M is doubly lexical there is a column t with 0 in row 1, 1 in row s. If $M(r,t)=1$ then the three rows and the columns $1, k, t$ form an M_3 submatrix. Hence $M(r,t)=0$. This forces $t<k$, otherwise s, r, k, t form a Γ. See Fig. 8.3. Now there must be a row $u>r$ with a 0 in column 1, and a 1 in column t. Then $M(u,k)=1$ to avoid a Γ s, u, t, k. But then u was better choice for s.

Case 2. $M(1,1)=0$. Suppose that the columns with 1's in row 1 are not identical. Let column j be the leftmost column with a 1 in row 1. Consider the submatrix formed by deleting columns $1, \cdots, j-1$. By Case 1 all the rows with 1's in column j must be consecutive, and identical on columns beyond j.

Any row i with a 1 in some column $k<j$ has a 1 in column j: for otherwise there must be a row $s>i$ with a 0 in column k and a 1 in column j; but then rows 1 and s are 1 on column j, and row i which lies between them is 0 on column j, in contradiction to the fact that the rows with 1's in column j are consecutive.

Let $i_1<i_2$ be two rows with 1's in column 1. Then they both have 1's in column j and cannot differ beyond column j. On the other hand if they differ on column $l<j$ then if l is 1 on row i_1 and 0 on row i_2 a Γ is formed (by rows i_1, i_2 and columns $1, l$)

	1	k	l
1	1	1	0
r	1	1	1
s	0	1	

FIG. 8.2

	1	t	k
1	1	0	1
s	0	1	1
r	1	0	1

FIG. 8.3

and if l is 0 on row i_1 and 1 on row i_2 then there is a row $i_3 > i_1$ which is 0 on column 1, 1 on column l, and hence 1 on column j. But then an M_3 submatrix is formed by rows i_1, i_2, i_3 and columns $1, l, j$. Thus the rows with 1's in column 1 are identical. □

Observation 8.2. If a 0-1 matrix has an M_3 submatrix then it has no weave ordering.

By Observation 5.3 a 0-1 matrix with a cycle submatrix has no Γ-free ordering, hence no weave ordering. Thus Theorem 8.1, Observation 8.2 and the existence of doubly lexical orderings imply:

THEOREM 8.3. *A 0-1 matrix is a plaid matrix iff it has a weave ordering.*

This theorem gives the equivalence of a co**NP** description of plaid matrices (having no cycle or M_3 submatrix) with an **NP** description (having a weave ordering). A previous **NP** description was given by Fagin. This, and good recognition algorithms, will be discussed below.

COROLLARY 8.4 ([Fg]). *Any plaid matrix has a singleton or duplicate row or column.*

A *duplicate* row (column) is a row (column) identical to another. A *singleton* row (column) has one 1.

Since plaid matrices are a subclass of the totally balanced matrices the same integrality properties hold for the linear programs $\max\{wx: Mx \leqq 1, x \geqq 0\}$ and $\min\{wx: Mx \geqq 1, x \geqq 0\}$. In [Lu2] a good algorithm is given for the Boolean basis problem on plaid matrices, with application to the problem of covering rectilinear polygons by a minimum number of rectangles. Integrality results obtained there do not extend to totally balanced matrices.

Plaid matrices have been introduced previously in the context of acyclic database schemes by Fagin [Fg], who used the name "γ-acyclic hypergraphs." Fagin uses the definition of plaid matrices given here. In addition to proving the equivalence of this definition with a number of desirable properties of database schemes, he proves that a matrix is a plaid matrix iff the following algorithm (due to D'Atri and Moscarini) succeeds in reducing the matrix to nothing: delete any singleton or duplicate row or column; repeat. (An equivalent characterization of plaid matrices in terms of a row ordering is given in [D'AM].) It can be proved that the success of the above algorithm is equivalent to the existence of a weave ordering; thus Theorem 8.3 can be derived from known results.

To test whether a 0-1 matrix is a plaid matrix, apply the doubly lexical ordering algorithm and then test if the result is a weave ordering. It will be shown that it is possible to test whether an ordered 0-1 $m \times n$ matrix with e 1's is weave ordered in time $O(L)$ where $L = e + m + n$. Thus plaid matrices can be recognized in time $O(L \log^2 L)$.

An ordered 0-1 matrix M will be specified for algorithmic purposes as described in the introduction.

Let M have rows $R = \{1, \cdots, m\}$ in order and columns $C = \{1, \cdots, n\}$ in order. For $r \in R$ let $d_C(r)$ be the minimum $c \in \{0, \cdots, n-1\}$ such that the columns $k > c$ with 1's in row r are consecutive, and identical in rows $s > r$. Symmetrically, for $c \in C$ let $d_R(c)$ be the minimum $r \in \{0, \cdots, m-1\}$ such that the rows $s > r$ with 1's in column c are consecutive, and identical in columns $k > c$. Finally, for $r \in R$ define $D_C(r) = \max_{s \leqq r} d_C(s)$, and let $D_C(0) = 0$.

The matrix M is *not* weave ordered iff there is a row r and a column c with $d_C(r) \geqq c$ and $d_R(c) \geqq r$; iff there is a column c with $D_C(d_R(c)) \geqq c$. Thus testing whether M is weave ordered can be done by testing if $D_C(d_R(c)) < c$ for all columns c. This can be done in linear time once D_C and d_R are known. Now computing D_C from d_C is trivial, and d_C can be computed in the same way as d_R (with rows and columns interchanged). So all that remains is the problem of computing d_R in linear time.

First find for each consecutive pair $r, r+1$ of rows the last column $l_C(r)$ where they differ. To do this, initialize an n-vector z to 0. This takes time $O(n)$. Then for any pair r, s of rows, the maximum column c such that $M(r, c)=0$, $M(s, c)=1$ can be found by: setting $z(c)=1$ for each column c in r's list of 1's; then finding the maximum column c in s's list of 1's for which $z(c)=0$; and then resetting $z(c)$ to 0 for each c in r's list of 1's. This takes time linear in the number of 1's in r and s. Doing this for each pair $r, r+1$, $r=1, \cdots, m-1$ and each pair $r+1, r$ (this second test is unnecessary if M is doubly lexical) takes time linear in L.

Then to compute d_R: Initialize an m vector y to 0. This takes time $O(m)$. For each column c, set $y(r)=1$ for each row r in c's list of 1's, and at the same time find the maximum such r. Look at rows $r-1, r-2, \cdots$ until a row s is found for which $l_C(s)>c$ or $y(s)=0$. In the first case $d_R(c)$ is s, and in the second case $d_R(c)$ is the maximum row $t<s$ for which $M(t, c)=1$. To find t scan once through c's list of 1's. If t does not exist then $d_R(c)$ is 0. Finally reset y to 0. This takes time linear in the number of 1's of c. Summed over all columns the time needed is linear in L. This completes the description of the algorithm to recognize weave orderings.

Both Fagin's characterization of plaid matrices (described above) and the [D'AM] characterization provide polynomial time algorithms to recognize plaid matrices. No implementation details or running times were given. The present recognition algorithm is faster than obvious implementations of either of these.

9. Lexical orderings of chordal graphs. A *perfect elimination ordering* of a graph is an ordering of the vertices which makes the neighbourhood matrix free of Γ's with upper left 1 on the main diagonal of the matrix. Equivalently, and more conventionally, a perfect elimination ordering of a graph G is an ordering $v_1, \cdots, v_n$ of the vertices of G such that if v_i is adjacent to $v_j, j>i$ and $v_k, k>i$, then v_j is adjacent (or equal) to v_k. Perfect elimination orderings are relevant for Gaussian elimination on sparse symmetric matrices [Ro].

THEOREM 9.1 ([FG]). *A graph is chordal iff it has a perfect elimination ordering.*

This classic result gives the equivalence of a co**NP** description (having no chordless cycle) and an **NP** description (having a perfect elimination ordering) of chordal graphs. It was originally proved using the equivalent:

COROLLARY 9.2 ([Di], [LB]). *Every chordal graph has a simplicial vertex—which is a vertex whose neighbours are completely connected.*

Theorem 9.1 can be proved using the following analogues of Theorems 5.1 and 5.2 for symmetric matrices together with the observation that a graph with a chordless cycle of 4 or more vertices has no perfect elimination ordering.

THEOREM 9.3. *Any lexical ordering of any chordal graph is a perfect elimination ordering.*

This is proved by:

THEOREM 9.4. *Let M be a symmetric* 0-1 *matrix with rows and columns indexed by S and with S ordered to make M doubly lexical. Any 3×3 symmetric submatrix of M formed by $s_1<s_2<s_3\in S$ with $M(s_1, s_3)=1$, $M(s_2, s_3)=0$ is, for some $k\geqq 4$, embedded in a $k\times k$ symmetric submatrix of M formed by $s_1<s_2<\cdots<s_k\in S$ with $M(s_i, s_{i+2})=1$, $i=1, \cdots, k-2$, $M(s_{k-1}, s_k)=1$, and $M(s_i, s_j)=0$ for other (i, j), $i<j$ except possibly* (1, 2). *In particular if M is a neighbourhood matrix of a graph then a Γ with upper left* 1 *on the main diagonal is embedded in a neighbourhood matrix of a chordless cycle of* 4 *or more vertices. See Fig.* 9.1.

Proof. An inductive construction beginning with $i=3$: $M(s_{i-1}, s_i)$ is 0. Let s_{i+1} be the rightmost column of M where rows s_{i-2} and s_{i-1} differ. Since $M(s_{i-2}, s_i)=1$,

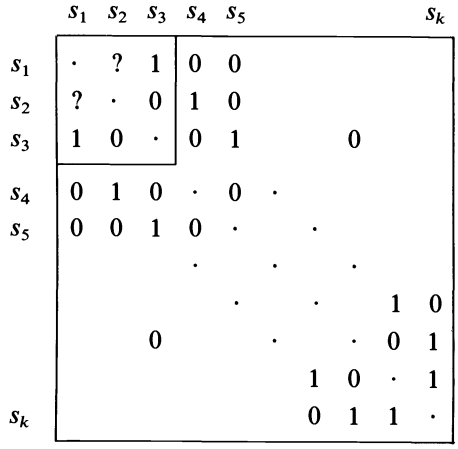


FIG. 9.1

$M(s_{i-1}, s_i) = 0$ and M is lexical s_{i+1} exists, $s_{i+1} > s_i$, and $M(s_{i-1}, s_{i+1}) = 0$, $M(s_{i-1}, s_{i+1}) = 1$. For $j = i, \cdots, 4$ consecutively, since s_j was by induction the rightmost column where rows s_{j-3} and s_{j-2} differed, $M(s_{j-2}, s_{i+1}) = 0$ implies $M(s_{j-3}, s_{i+1}) = 0$. If $M(s_i, s_{i+1}) = 1$ the theorem is proved with $k = i+1$, and otherwise the construction applies once more. □

Recall from § 7 that a matrix is a subtree matrix iff it satisfies the Helly property for columns and its intersection graph is chordal. Since the intersection graph of the clique matrix of a graph is the graph itself, and since clique matrices satisfy the Helly property for columns, this implies:

LEMMA 9.5 ([Bu]). *A graph is chordal iff its clique matrix is a subtree matrix.*

The integrality properties of subtree matrices M in the linear program $\max\{wx: Mx \leqq 1, x \geqq 0\}$ can then be used to derive optimization properties of chordal graphs—e.g., that the maximum size of an independent set is equal to the minimum number of cliques needed to cover the vertices of the graph. Such optimization properties were a main motivation for the study of chordal graphs, and of the broader class of perfect graphs. See [B] or [Du].

The doubly lexical ordering algorithm together with a linear time algorithm to recognize if a vertex ordering of a graph is perfect elimination can be used to recognize chordal graphs of n vertices and m edges in almost linear time $O(L \log^2 L)$ where $L = n + m$.

The papers [RTL] and [TY] contain linear time algorithms, which will be described below, to order the vertices of a graph in what will be a perfect elimination ordering if the graph is chordal. Linear time algorithms to test if an ordering is perfect elimination are given. Neither algorithm produces lexical orderings (even of chordal graphs), nor strong elimination orderings of totally balanced graphs. The advantage of the present method is that a lexical ordering of a graph is a perfect elimination ordering if the graph is chordal, *and* is a strong elimination ordering if the graph is totally balanced.

An approach similar to that used to recognize Γ-free and supported Γ orderings can be used to test if a given vertex ordering of a graph (given by adjacency lists) is perfect elimination. The analogue of Lemmas 5.8 and 7.7 is:

LEMMA 9.6. *Let G be a graph with an ordering on the vertex set V. Let M be the correspondingly ordered neighbourhood matrix. If M has a Γ formed by rows s, t and columns s, u then it has such a Γ with the property that $M(v, s) = 0$ for all v, $s < v < t$.*

It is possible to test for Γ's of the form described in the lemma in linear time since they have distinct upper right 1's. The implementation details are similar to those used in previous sections.

If a Γ with upper left 1 on the main diagonal is found in a symmetric doubly lexical neighbourhood matrix it can be expanded to a chordless cycle of 4 or more vertices in the graph in time $O(L)$ as in the proof of Theorem 9.4. Implementation details are as described in § 5. This problem of finding in linear time a chordless cycle of 4 or more vertices in a nonchordal graph was also solved in [TY2].

The following graph ordering algorithm of Rose, Tarjan and Lueker [RTL] is implementable in linear time, and produces perfect elimination orderings of chordal graphs:

Lexicographic breadth first search (LexBFS). Order vertices by assigning numbers from n to 1. Assign the number k to a vertex v (as yet unnumbered) which has lexically largest vector $(a_i: i = k+1, \cdots, n)$, $a_i = 1$ if v is adjacent to the vertex numbered i, $a_i = 0$ otherwise.

Every lexical ordering of a graph is a LexBFS ordering, but not conversely. (See Fig. 9.2.)

	1	2	3	4	5
1	1	0	1	1	0
2	0	1	1	0	1
3	1	1	1	1	1
4	1	0	1	1	1
5	0	1	1	1	1

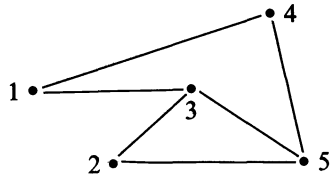


FIG. 9.2. *A* LexBFS *ordering of a graph (hence a P-ordering) which is not a lexical ordering.*

The following graph ordering algorithm of Tarjan and Yannakakis [TY] (or see [G, p. 87]) is also implementable in linear time, and produces perfect elimination orderings of chordal graphs:

Maximum cardinality search (MCS). Order the vertices by assigning numbers from n to 1. As the next vertex to number pick a vertex adjacent to the most numbered vertices.

A lexical ordering of a graph is not always an MCS ordering, nor vice versa.

In [TY] *P-orderings* were introduced as a unification of LexBFS orderings and MCS orderings. It was shown that any *P-ordering* of a chordal graph is a perfect elimination ordering. A *P-ordering* of a graph $G = (V, E)$ is an ordering $v_1, \cdots, v_n$ of V such that if $i < j < k$ and v_k is adjacent to v_i and not to v_j then there is a vertex v_l, $l > j$, adjacent to v_j and not to v_i. No algorithm was given for finding P-orderings.

Recall the very similar characterization of lexical orderings of graphs at the end of § 2. Clearly any lexical ordering is a P-ordering. The converse does not hold (see Fig. 9.2).

Acknowledgments. Jack Edmonds was a main advisor to me in this work; I thank him for substantial ideas and assistance. I thank R. E. Tarjan for help on the doubly lexical ordering algorithm.

REFERENCES

[An] R. P. ANSTEE, *Properties of* $(0,1)$*-matrices with forbidden configurations*, J. Combin. Theory, A 31 (1981), pp. 256-269.

[AF] R. P. ANSTEE AND M. FARBER, *Characterizations of totally balanced matrices*, J. Algorithms, 5 (1984), pp. 215-230.

[BEW] I. BARANY, J. EDMONDS AND L. WOLSEY, *Packing and covering a tree by subtrees*, Combinatorica, 6 (1986), pp. 221-233.

[BFMY] C. BEERI, R. FAGIN, D. MAIER AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput. Mach., 30 (1983), pp. 479-513.

[B] C. BERGE, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[BK] A. E. BROUWER AND A. KOLEN, *A super-balanced hypergraph has a nest point*, Report ZW 146/80, Mathematisch Centrum, Amsterdam, 1980.

[Bu] P. BUNEMAN, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205-212.

[CN] G. J. CHANG AND G. L. NEMHAUSER, *The k-domination and k-stability problems on sun-free chordal graphs*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 332-345.

[D'AM] A. D'ATRI AND M. MOSCARINI, *On the recognition and design of acylic databases*, Proc. 3rd ACM SIGACT-SIGMOD Conf. Principles of Database Systems, Waterloo, Canada 1984, pp. 1-8.

[Di] G. A. DIRAC, *On rigid circuit graphs*, Abh. Math. Sem. Univ. Hamburg, 25 (1961), pp. 71-76.

[Du] P. DUCHET, *Classical perfect graphs: an introduction with emphasis on triangulated and interval graphs*, Ann. Discrete Math., 21 (1984), pp. 67-96.

[EKLLS] J. EDMONDS, A. KOLEN, J.-K. LENSTRA, A. LUBIW AND A. SCHRIJVER, Discussion in '82 of [Lu1] and a preliminary version of [HSK], in which the concept of doubly lexical orderings arose.

[Fg] R. FAGIN, *Degrees of acyclicity for hypergraphs and relational database schemes*, J. Assoc. Comput. Mach., 30 (1983), pp. 514-550.

[Fa1] M. FARBER, *Domination, independent domination, and duality in strongly chordal graphs*, Discrete Appl. Math., 7 (1984), pp. 115-130.

[Fa2] ———, *Characterizations of strongly chordal graphs*, Discrete Math., 43 (1983), pp. 173-189.

[Fr1] A. FRANK, *On a class of balanced hypergraphs*, Discrete Math., 20 (1977), pp. 11-20.

[Fr2] ———, *Some polynomial algorithms for certain graphs and hypergraphs*, Proc. 5th British Combinatorial Conf., Congressus Numeratium No. XV, Utilitas Math., Winnipeg, 1976, 226-261.

[FG] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835-855.

[FHO] D. R. FULKERSON, A. J. HOFFMAN AND R. OPPENHEIM, *On balanced matrices*, Math. Programming Stud., 1 (1974), pp. 120-132.

[Gs] R. GILES, *A balanced hypergraph defined by subtrees of a tree*, ARS Combinatoria, 6 (1978), pp. 179-183.

[G] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[HSK] A. J. HOFFMAN, M. SAKAROVICH AND A. KOLEN, *Totally balanced and greedy matrices*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 721-730.

[Le] J. LEHEL, *A characterization of totally balanced hypergraphs*, Discrete Math., 57 (1985), pp. 59-65.

[LB] C. G. LEKKERKERKER AND J. CH. BOLAND, *Representations of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45-64.

[Lu1] A. LUBIW, *Γ-free matrices*, Master's thesis, Dept. Combinatorics and Optimization, University of Waterloo, 1982.

[Lu2] ———, *Orderings and some combinatorial optimization problems with geometric applications*, Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1985.

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

[PT] R. PAIGE AND R. E. TARJAN, *Three partition refinement algorithms*, this Journal, to appear.

[Ro] D. J. ROSE, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597-609.

[RTL] D. J. ROSE, R. E. TARJAN AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, this Journal, 5 (1976), pp. 266-283.

[S] P. J. SLATER, *R-domination in graphs*, J. Assoc. Comput. Mach., 23 (1976), pp. 446-450.

[Tm] A. TAMIR, *A class of balanced matrices arising from location problems*, SIAM J. Algebraic Discrete Methods, 4 (1983), pp. 363-370.

[TY] R. E. TARJAN AND M. YANNAKAKIS, *Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, this Journal, 13 (1984), pp. 566-579. (See next reference as well.)

[TY2] ———, *Addendum: simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, this Journal, 14 (1985), pp. 254-255.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987
© 1987 Society for Industrial and Applied Mathematics
007

THE PARALLEL COMPLEXITY OF ABELIAN PERMUTATION GROUP PROBLEMS*

PIERRE MCKENZIE† AND STEPHEN A. COOK‡

Abstract. We classify Abelian permutation group problems with respect to their parallel complexity. For such groups specified by generating permutations we show that testing membership, computing order and testing isomorphism are NC^1-equivalent to (and therefore have essentially the same parallel complexity as) determining solvability of a system of linear equations modulo a product of small prime powers; we show that intersecting two such groups is NC^1-equivalent to computing setwise stabilizers; we show that each of these problems is NC^1-reducible to the problem of computing a generator-relator presentation. Then we prove that the aforementioned problems belong to NC^3, thus identifying several natural set recognition problems in NC which may lie outside NC^2. Finally we prove that NC^4 contains the problem of computing the cyclic decomposition of an Abelian permutation group. Background results include an NC^1 solution to the problem of computing the product of n integers modulo a $\lceil \log n \rceil$-bit integer, and an NC^1 reduction from the problem of computing a path between two nodes in a graph to that of determining accessibility of one node from another.

Key words. permutation group, Abelian group, parallel complexity, *NC*

AMS(MOS) subject classification. 68Q

1. Introduction. A fundamental problem in the theory of parallel computation is the determination of which problems with polynomial time sequential algorithms are subject to substantial speedup by using parallel computers. To a large extent, this means trying to classify problems in P either into NC (computable in parallel time polynomial in $\log n$ with a polynomial number of processors [Pi79]) or show they are log space complete for P (in which case they probably are not in NC because of the unlikely consequence $P = NC$). For example context-free language membership [Ru80], minimum spanning forest [SaJa81], and determinant of an integer matrix [Cs76] are in NC. Among the problems known to be complete for P are the circuit value problem [La75], linear programming [DoLiRe79] and maximum flow in a weighted directed graph [GoShSt81]. See [Co85] and [HoRu84] for more examples of both types of problems.

We are concerned here with permutation group algorithms. The sequential complexity of such algorithms is being studied intensively (e.g. [FuHoLu80], [Ho82], [Je82]), but little attention had been paid to their parallel complexity prior to the precursors of this paper ([McCo83], [Mc84], see also [Re83]). A problem fundamental to computational group theory and its applications (see, for example, [Ho82]) is permutation group membership (GM), in which one is required to decide membership of a test permutation in a permutation group specified by a list of permutations generating the group. (Each permutation is presented as a product of disjoint cycles.)

* Received by the editors April 15, 1985; accepted for publication (in revised form) October 22, 1986. This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Killam Foundation of Canada. Most of the results in this paper appear in the Ph.D. thesis of the first author which was done while the author was at the University of Toronto, Toronto, Ontario, Canada. Some results also appear in the Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science.

† Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Québec, Canada H3C 3J7.

‡ Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

GM was shown to be in P in [FuHoLu80] and we believe it to be an appropriate starting point in the study of parallel algorithms in group theory.

This paper deals exclusively with Abelian permutation groups (see [LuMc85] and [Lu86] for results on more general groups). For such Abelian groups specified by generating permutations we consider the following problems: group membership (AGM), isomorphism testing (AISO), computing order (AORDER), computing generating permutations for the intersection of two groups (AINTER), computing generating permutations for a setwise stabilizer subgroup (ASET), computing a generator-relator presentation (AGP), and computing generating permutations for the factors in a cyclic decomposition of the group (FACTOR).

Let LCON denote the problem of determining solvability of a system of linear equations, over the integers, modulo a product of small prime powers. We define LCONNULL as the problem of computing a spanning set for the $\mathbf{Z}$-module of all solutions to an instance of LCON in which the equations are homogeneous. Most of our NC upper bounds depend on the ability to solve (possibly undetermined) systems of linear equations modulo a small prime p in NC^2 (by [Mu86], [BoGaHo82] and [BoCoPi83]).

Our main results are

(i) AGM, AISO, AORDER and LCON are NC^1-equivalent (i.e. equivalent under log depth reducibility).

(ii) AINTER and ASET are NC^1-equivalent.

(iii) AGP and LCONNULL are NC^1-equivalent.

(iv) Problems in (i), (ii) and (iii) are hard for nondeterministic log space under NC^1 reducibility (and thus under "log space Cook" reducibility); hence a breakthrough would be required to solve any one of them better than in NC^2.

(v) Problems in (i), (ii) and (iii) belong to NC^3; in particular, problems in (i) constitute examples of natural set recognition problems in NC which seem to fall outside NC^2.

(vi) FACTOR$\in NC^4$.

We can strengthen (v) in several cases. In particular, AGM restricted to *cyclic* groups belongs to NC^1. When the Abelian groups are *transitive*, AGM and AORDER belong to NC^1 while AINTER and ASET belong to NC^2. If the number of permutations generating the input groups is viewed as constant (or if the number of variables is viewed as constant in the case of LCON), then AGM, AISO, AORDER and LCON belong to NC^1. Finally, AGM restricted to *elementary Abelian* groups is in NC^2 (yet still NC^1-hard for nondeterministic log space).

Section 2 discusses background and notation; it also lists the definitions of most problems discussed in this paper. In § 3 we obtain preliminary results, including the following:

(vii) The product of any number of integers, modulo an integer presented in unary notation, can be computed in NC^1.

(viii) Computing a shortest path between two nodes in a digraph reduces in NC^1 to the problem of determining whether any path exists from a given node to another.

(ix) Generating permutations for each maximal prime power order subgroup (i.e. Sylow subgroup) of an Abelian group can be computed in NC^1.

In § 4 we deal with the case of transitive Abelian groups. In § 5 we reduce linear congruence problems to Abelian permutation group problems and we show hardness of all these for nondeterministic log space. In § 6 we compare the parallel complexities of various Abelian group problems, including AGM, AISO, AORDER, AINTER, ASET and AGP. Then in § 7 we return to linear algebra problems, this time to exhibit

reductions (converse to those found in § 5) from Abelian group problems to linear congruence problems. Finally, we describe our *NC* upper bounds in § 8, and we discuss further work and open questions in § 9.

2. Background and notation. Throughout these pages logarithms are in base 2. The set of all strings over the alphabet $\{0, 1\}$ is denoted $\{0, 1\}^*$. $\mathbf{Z}$ and $\mathbf{Z}_m$ represent the integers and the integers modulo m, respectively. For a prime p and an integer m, the "order of m at p," denoted $\text{ord}_p(m)$, is the exponent of the largest power of p dividing m; the "rest of m at p," denoted $\text{rest}_p(m)$, stands for $m/p^{\text{ord}_p(m)}$.

2.1. Group theory. In this subsection we introduce notation and we state a number of facts which are standard group theory. Our notation and terminology is mostly that of [Wi64].

Let Ω be an n-element set and let G be a subgroup of $\text{Sym}(\Omega)$, the group of permutations of Ω. We write $H \leqq G$ when H is a subgroup of G, and we consider G to be a subgroup of itself. Until otherwise specified, let $g \in G$, $\alpha \in \Omega$, and $\Gamma \subseteq \Omega$. The image of point α under permutation g is denoted α^g. Given h another element of G, we assume the product of g and h defined by the formula $\alpha^{gh} = (\alpha^g)^h$; hence we read products from left to right. In general we write

$$\Gamma^g = \{\beta^g \mid \beta \in \Gamma\}.$$

The *pointwise stabilizer* G_Γ is the largest subgroup of G which fixes Γ pointwise:

$$G_\Gamma = \{g \in G \mid \beta^g = \beta \text{ for each } \beta \in \Gamma\}$$

(we simply write α^G for Γ^G and G_α for G_Γ when $\Gamma = \{\alpha\}$). The *setwise stabilizer* $G_{[\Gamma]}$ is the largest subgroup of G which fixes Γ setwise:

$$G_{[\Gamma]} = \{g \in G \mid \Gamma^g = \Gamma\}.$$

The set Γ is a *fixed block* of G if $\Gamma^G = \Gamma$. If Γ is a fixed block of G then g^Γ denotes g restricted to Γ and

$$G^\Gamma = \{h \in \text{Sym}(\Gamma) \mid h = g^\Gamma \text{ for some } g \in G\}$$

is the *constituent* of G on Γ. The *G-orbit* of α is α^G. G is *transitive* on Ω if Ω is a G-orbit. Clearly any G-orbit Γ is a fixed block of G and the resulting constituent G^Γ is transitive on Γ.

Angle brackets are used to denote *the group generated by*, that is,

$$\langle g_1, g_2, g_3 \rangle$$

is the group generated by g_1, g_2 and g_3, assuming that g_1, g_2 and g_3 belong to some known group (when the g_is are permutations the "known group" will always be the appropriate symmetric group). The *degree* of G is the number of points actually moved by G, and therefore not fixed by all elements of G. The *order* of G, denoted $|G|$, is the number of elements in G; $|g|$ denotes $|\langle g \rangle|$.

Any permutation can be written as a product of disjoint cycles. For example, the permutation of $\{1, 2, 3, 4, 5\}$ whose "pointwise" representation is

$$4, 5, 1, 3, 2$$

has

$$(143)(25)$$

as its lexicographically first disjoint cycle representation.

Fact 2.1. The order of a permutation is the least common multiple of the orders of its disjoint cycles.

A group is *cyclic* if $G=\langle g\rangle$ for some g; we let C_m denote the unique (up to isomorphism) cyclic group of order $m>0$. A group G is *Abelian* if $gh=hg$ for all $g, h\in G$. G is a *p-group* , or a *p-subgroup* of some larger group, if p is prime and $|G|=p^k$ for some integer k. (We say that a group element g is a *p-element* if $\langle g\rangle$ is a p-group.) A maximal p-subgroup (not necessarily proper) of an arbitrary group G is called a *Sylow p-subgroup* of G. We write that group G is a *P-group* if it is a p-group for some prime p. (We use the term P-group to denote a generic group of prime power order and we use the term p-group for a group of order p to some power for a prime p previously agreed upon.)

Fact 2.2 (Lagrange). Given a finite group G, if $H\leqq G$ then $|H|$ divides $|G|$.

Fact 2.3 (see, for example, [Ar44, p. 49]). In a finite Abelian group $G=\langle g_1,\cdots,g_r\rangle$ the order of any element divides 1cm $(|g_1|,\cdots,|g_r|)$ and there is an element of order equal to 1cm $(|g_1|,\cdots,|g_r|)$.

Fact 2.4. For each prime p, an Abelian group G has a unique Sylow p-subgroup, which we denote $S_p(G)$; $S_p(G)$ is exactly the set of all p-elements of G, and G is the direct product of its Sylow subgroups.

Fact 2.5 (see [Ha59, Thm. 3.3.2]). An Abelian p-group G is isomorphic to

$$C_p\times\cdots\times C_p\times C_{p^2}\times\cdots\times C_{p^2}\times\cdots\times\cdots\times C_{p^k}\times\cdots\times C_{p^k}$$

for some positive integers k and $\delta_1,\cdots,\delta_k$, where $\delta_1,\cdots,\delta_k$ are the respective numbers of occurrences of $C_p,\cdots,C_{p^k}$. We refer to the ordered k-tuple $(\delta_1,\cdots,\delta_k)$ as the *signature* of the Abelian p-group G.

Example. Let G be the Abelian group

$$\langle(12)(345678)(ABCDE)(GH),(1G2H)(ABCDE)\rangle.$$

Then G has 3 orbits:

$$\Gamma_1=\{1,2,G,H\},\quad \Gamma_2=\{3,4,5,6,7,8\},\quad \Gamma_3=\{A,B,C,D,E\}$$

and the transitive constituents are

$$G^{\Gamma_1}=\langle(12)(GH),(1G2H)\rangle=\langle(1G2H)\rangle,\quad G^{\Gamma_2}=\langle(345678)\rangle,\quad G^{\Gamma_3}=\langle(ABCDE)\rangle.$$

The Sylow subgroups of G are

$$S_2(G)=\langle(12)(36)(47)(58)(GH),(1G2H)\rangle,\quad S_3(G)=\langle(375)(486)\rangle,$$
$$S_5(G)=\langle(ABCDE)\rangle.$$

Their orders are 8, 3 and 5, respectively, whence $|G|=120$. $S_3(G)$ and $S_5(G)$ are already cyclic, and a possible decomposition of $S_2(G)$ is

$$S_2(G)=\langle(36)(47)(58)\rangle\times\langle(1G2H)\rangle\cong C_2\times C_4.$$

2.2. List of problem definitions. For ease of reference, we list here definitions of the main problems dealt with in this paper.

For each problem discussed we assume a reasonably binary encoding of the problem instances. Unless otherwise specified, the letter n stands for the length of this encoding for the particular problem instance under consideration. In the context of a particular problem instance, we say that an integer m is *tiny* if the absolute value $|m|\leqq n$. (An integer specified in unary notation is the archetypical example of a tiny integer.) We say that an integer m has *tiny factors* if $p^{\mathrm{ord}_p(m)}$ is tiny for every prime p. On input, we assume that integers are specified in binary notation, except when an integer is tiny, in which case unary notation is used. On output, integers are represented in binary; by "computing the prime factorization of an integer m" (problem AORDER)

we mean "computing the binary representation of $p^{\mathrm{ord}_p(m)}$ for each prime p dividing m." We assume that graphs are presented as adjacency lists.

Often a problem has many possible solutions and some of them have lengths larger than any polynomial in n (for example when computing a generating set for a group). It is understood that we only consider the polynomial length solutions as valid solutions.

In some cases we require computing formal expressions in the generators $g_1, \cdots, g_r, g_1^{-1}, \cdots, g_r^{-1}$ of a group (for example when requiring an expression for the test permutation h in terms of the generators in problem AGMX; note that the shortest length product of g_i's expressing h may be exponential even in the case of cyclic groups). In this paper it will suffice to write such a (nontrivial) formal expression as a sequence of pairs of integers $(i_1, j_1), \cdots, (i_k, j_k)$ written in binary, $k \geqq 1$, $1 \leqq i_l \leqq r$. This sequence will represent the formal expression

$$g_{i_1}^{j_1} g_{i_2}^{j_2} \cdots g_{i_k}^{j_k},$$

in which we have abbreviated k consecutive occurrences of a symbol g_i^{ε} by g_i^k if $\varepsilon = 1$ and by g_i^{-k} if $\varepsilon = -1$. (In general it would be appropriate to write such formal expressions as "straight line programs" because this may well be the only way to express an arbitrary permutation h, in terms of an arbitrary generating set for a group containing h, using only a polynomial length string of symbols; since we deal with Abelian groups in this paper the aforementioned representation has polynomial length; see [FuHoLu80] for a proof that in an arbitrary permutation group there always exists a polynomial length straight line program expressing an element h.)

In each problem instance we assume that all input permutations are members of a symmetric group Sym (Ω) for some finite set Ω. Unless otherwise indicated, permutations are presented as products of disjoint cycles. We say that a permutation g is "presented pointwise" if it is presented as a list of pairs (α, α^g) for $\alpha \in \Omega$. For the actual representation of a permutation over the alphabet $\{0, 1\}$ we encode each point in Ω as a unique binary integer between c and $|\Omega| + c$, where c is a small constant which accounts for the binary codes representing the punctuation symbols "(",")", and ",". (There remains to deal with the issue of recognizing the end of each symbol; this is handled by splitting binary strings into "bytes" having a fixed number of bits, say 8, and reserving a specific byte content to represent a separator symbol.)

LIST OF PROBLEMS

AGM (Abelian Group Membership)
APGM (Abelian P-Group Membership)
CYCLICGM (Cyclic Group Membership)
STAGM (Subgroup of Transitive Abelian Group Membership)
TAGM (Transitive Abelian Group Membership)

Given: a test permutation h, together with permutations $g_1, \cdots, g_r$ generating a group G which is
- AGM: Abelian.
- APGM: an Abelian P-group.
- CYCLICGM: cyclic.
- STAGM: contained in an Abelian group transitive on the set of points permuted by G.
- TAGM: Abelian and transitive on the set of points permuted.

Problem: determine whether $h \in G$.

AGMX (Abelian Group Membership and Expression)

CYCLICGMX (Cyclic Group Membership and Expression)
STAGMX (Subgroup of Transitive Abelian Group Membership and Expression)
Given: a test permutation h, together with permutations $g_1, \cdots, g_r$ generating a group G which is
AGMX: Abelian.
CYCLICGMX: cyclic.
STAGMX: contained in an Abelian group transitive on the set of points permuted by G.
Problem: determine whether $h \in G$ and compute a formal expression for h in terms of $g_1, \cdots, g_r, g_1^{-1}, \cdots, g_r^{-1}$ (if any).

AGP (Abelian Group Presentation)
Given: permutations $g_1, \cdots, g_r$ generating an Abelian group G.
Problem: compute vectors $X_1, \cdots, X_m \in \mathbf{Z}^r$ which generate the kernel of the epimorphism $\psi: \mathbf{Z}^r \to G$ (see § 5).

AINTER (Abelian groups Intersection)
STAINTER (Subgroups of Transitive Abelian group Intersection)
Given: permutations $g_1, \cdots, g_r$ generating a group G and permutations $h_1, \cdots, h_s$ generating a group H with the property that $\langle G, H \rangle$ is
AINTER: Abelian.
STAINTER: contained in an Abelian group transitive on the set of points permuted by G.
Problem: compute permutations generating $G \cap H$.

AISO (Abelian group Isomorphism)
APISO (Abelian P-group Isomorphism)
Given: permutations $g_1, \cdots, g_r$ generating a group G and permutations $h_1, \cdots, h_s$ generating a group H such that the groups are
AISO: Abelian.
APISO: Abelian P-groups.
Problem: determine whether G and H are isomorphic as abstract groups.

AORDER (Abelian group Order)
APORDER (Abelian P-group Order)
CYCLICORDER (Cyclic group Order)
STAORDER (Subgroup of Transitive Abelian group Order)
TAORDER (Transitive Abelian group Order)
Given: permutations $g_1, \cdots, g_r$ generating a group G which is
AORDER: Abelian.
APORDER: an Abelian P-group.
CYCLICORDER: cyclic.
STAORDER: contained in an Abelian group transitive on the set of points permuted by G.
TAORDER: Abelian and transitive on the set of points permuted.
Problem: compute the prime factorization of $|G|$.

ASET (Abelian group Setwise Stabilizer)
STASET (Subgroup of Transitive Abelian group Setwise Stabilizer)

Given: $\Delta \subseteq \Omega$, together with permutations $g_1, \cdots, g_r$ generating a group $G \leqq \mathrm{Sym}(\Omega)$ which is
ASET: Abelian.
STASET: contained in an Abelian group transitive on the set of points permuted by G.
Problem: compute permutations generating $G_{[\Delta]}$.

COMPONENT (Connected Component in Graph)
Given: undirected graph (V, E).
Problem: compute the partition of V into connected components.

DGAP [UGAP] {DGAP1} (Directed [Undirected] Graph Accessibility {of outdegree 1})
Given: directed [undirected] graph (V, E) {of outdegree 1} and $u, v \in V$.
Problem: determine whether there is a path from u to v in the graph.

DPATH [UPATH] {DPATH1} (Path in a Directed [Undirected] Graph {of outdegree 1})
Given: directed [undirected] graph (V, E) {of outdegree 1} and $u, v \in V$.
Problem: compute a sequence of edges forming a shortest path (if any) from u to v.

FACTOR (Factor an Abelian group)
PFACTOR (Factor an Abelian P-group)
Given: permutations $g_1, \cdots, g_r$ generating a group G which is
FACTOR: Abelian.
PFACTOR: an Abelian P-group.
Problem: compute nontrivial permutations $h_1, \cdots, h_s \in G$ such that $|h_{i+1}|$ divides $|h_i|$, $1 \leqq i < s$, and $G = \langle h_1 \rangle \times \cdots \times \langle h_s \rangle$.

LCON (Linear Congruences)
LCONX (Linear Congruences and Expression)
Given: $A \in \mathbf{Z}^{k \times m}$, $B \in \mathbf{Z}^k$, and an integer q, with tiny factors, presented only as a list of its tiny factors $p^{\mathrm{ord}_p(q)}$.
Problem:
LCON: determine whether there exists $X \in \mathbf{Z}^m$ satisfying $AX \equiv B \bmod q$ (this notation means that corresponding vector entries are each congruent mod q).
LCONX: determine whether there exists $X \in \mathbf{Z}^m$ satisfying $AX \equiv B \bmod q$ and compute such an X (if any).

LCONNULL (Linear Congruences "Nullspace")
Given: $A \in \mathbf{Z}^{k \times m}$ and an integer q, with tiny factors, presented only as a list of its tiny factors $p^{\mathrm{ord}_p(q)}$.
Problem: Compute a spanning set for the $\mathbf{Z}$-module $\{X \in \mathbf{Z}^m \mid AX \equiv 0 \bmod q\}$.

LDGAP (Length of Directed Graph Accessibility)
Given: directed graph (V, E), $u, v \in V$ and a tiny integer k.
Problem: determine whether there is a path of length k from u to v in the graph.

ORBIT (group Orbit)
Given: permutations $g_1, \cdots, g_r$ generating a group G.
Problem: compute the partition of Ω into G-orbits.

ZERODET (Singularity of Integer Matrix)
Given: $A \in \mathbf{Z}^{m \times m}$.
Problem: determine whether the determinant of A is zero.

2.3. Complexity theory. We assume basic familarity with what might be termed "standard" complexity theory, such as can be found in [HoU179] and in [GaJo79]. In particular, the reader is referred there for more details on the complexity classes P, DSPACE ($\log n$) and NSPACE ($\log n$). An $O(\log n)$ space-bounded deterministic Turing machine will be referred to as a log space transducer.

FP and FL are used to denote the obvious classes of (multiple-valued) functions $\{0, 1\}^* \to \{0, 1\}^*$ corresponding to the classes P and DSPACE ($\log n$) respectively.

The reader will have noticed that many problems listed in § 2.2 involve multiple-valued functions in that we require computing any one of a number of satisfactory solutions (if one exists) to a given problem instance. In the terminology of [GaJo79] these are *search problems.* Garey and Johnson [GaJo79] define the formal counterpart of a search problem by introducing the notion of "solving" a relation. We will make similar conventions. Intuitively "computing a function f" in the case where f is multiple-valued will simply mean that for each x in the domain of f the computation model under consideration outputs one of the possible values of $f(x)$.

In this paper we will only consider (partial and) multiple-valued functions f for which the following holds: whenever $f(x)$ is defined on some input string x of length m, then f is also defined on all inputs of length m.

2.3.1. Parallel computation model. For a detailed definition of the circuit model we refer the reader to [Co81]. We use the same definition of uniformity, namely we require that a log-space transducer be able to compute a description of the circuit family. Note that fan-in in our circuits is bounded (at most 2).

Definition (NC^k and NC). NC^k is the set of all (possibly partial and possibly multiple-valued) functions $f: \{0, 1\}^* \to \{0, 1\}^*$ computable by a uniform Boolean circuit family $\langle \alpha_n \rangle$ in which α_n has size polynomial in n and depth $O((\log n)^k)$. $NC = \bigcup_k NC^k$.

Intuitively NC is the set of functions computable *superfast* on a parallel computer of *feasible* size. The correspondence between uniform circuit size and Turing machine time (see [FiPi74], [Sc76], [Bo77]) yields the important inclusion

$$NC \subseteq FP,$$

only suspected to be proper.

2.3.2. NC^1 reducibility. NC^1 will be the lowest level complexity class of interest to us. We note that the following problems belong to NC^1: sorting a list of binary integers [MuPr75], integer multiplication, iterated integer addition (i.e. addition of m m-bit integers) (see [Sa76]). Referring to our permutation encodings (2.2), the reader should verify that given the encoding of the disjoint cycle presentation of a permutation it is possible to answer, in NC^1, basic questions like determining whether two points belong to the same cycle and computing the length of each cycle.

NC^1 reducibility will provide us with a means of comparing the parallel complexities of different problems. In this section we define this reducibility, we exhibit some of its properties and we define the closure of a set of functions with respect to NC^1 reducibility. Throughout this section a "function" refers to a (possibly partial and possibly multiple-valued) function.

A reduction from a function f to a function g will be defined as a circuit family for f using special "oracle" gates computing g. We can speak of a reduction from a

function f to a set of functions $\{g_1, \cdots, g_m\}$ if the circuit family for f uses oracle gates for $g_1, \cdots, g_m$. The following is in essence the notion of NC^1 reducibility found in [Co85]:

DEFINITION (NC^1 reducibility). A function f is NC^1-reducible to a set of functions S, denoted $f \leqq S$ (or $f \leqq g$ when $S = \{g\}$), if by making use of special gates each computing a function in S, f can be solved by a uniform family of circuits in which the nth circuit has depth $O(\log n)$ and has size $n^{O(1)}$ (where an oracle gate g is considered to have size equal to one and depth equal to $\log(1 + (\#$ of inputs to $g))$).

PROPOSITION 2.6. *Let f be a function and S a set of functions. Then $f \leqq S$ and $S \subseteq NC^k \Rightarrow f \in NC^k$.*

Proof. We simply replace oracle gates in the circuit reducing f to S by the appropriate NC^k circuits. See [Co85] or [Mc84] for the argument that the depth of the resulting circuit is $O(\log^k n)$. □

Given sets of functions S and T, we write $S \leqq T$ when $f \leqq T$ for all $f \in S$.

PROPOSITION 2.7. *Let S, T and W be sets of functions. Then $S \leqq T$ and $T \leqq W \Rightarrow S \leqq W$.*

Proof. The proof is similar to that of Proposition 2.6. □

DEFINITION (closure of a set of functions under NC^1 reducibility). Let S be a set of functions. The closure of S under NC^1 reducibility, denoted $\langle S \rangle$, is defined as $\{f \mid f \leqq S\}$.

Observe that, by Proposition 2.6, NC^k is *closed* under NC^1 reducibility, that is, $NC^k = \langle NC^k \rangle$. By contrast, NSPACE$(\log n)$ is not closed under NC^1 reducibility since $\langle$NSPACE$(\log n)\rangle$ contains functions other than 0-1 (i.e. decision) functions. For the class FL of functions computable in deterministic log space we have

Fact 2.8 [Co85]. $FL = \langle$DSPACE$(\log n)\rangle$.

3. Preliminary results. In this section we describe NC^1 circuits to compute a number of functions which will be used later. In particular, we discuss the parallel complexity of arithmetic operations on tiny integers, we show how to compute all Sylow subgroups of an Abelian group, and we show that when restricted to cyclic groups the problem GM can be solved in NC^1. We also describe an NC^1 reduction from the problem of computing a path between two nodes in a graph to the problem of determining whether any path between two nodes exists. Finally, we examine the issue of the representation of permutations and we are led to a number of problems found NC^1-complete for FL.

The following proposition illustrates the use of a ubiquitous trick, table lookup.

PROPOSITION 3.1. *Let a be any integer and let m be a tiny (relative to the size n of the problem encoding, see § 2.2) integer. NC^1 contains the problems of computing a* mod *m and $\lfloor a/m \rfloor$.*

Proof. Consider first the mod computation. Let $a = \sum_{i=0}^{k-1} a_i 2^i$. The idea is to compute $b_i = 2^i \bmod m$ for $0 \leqq i < k$, to add the b_i's corresponding to nonzero a_i's exactly, and then to reduce the result mod m. Let us take a closer look at how this is done.

The first step is carried out with the help of the log space transducer constructing the Boolean circuit family (recall the notion of uniformity), in the following sense. Clearly a log space transducer on input n in unary notation can compute $2^i \bmod m$ for any $i \leqq n$ and for any tiny integer m. Now the transducer does not know m (it only sees n in unary), so it cannot tell exactly which values of $2^i \bmod m$ the Boolean circuit will require on its input. However there are only a polynomial (in n) number of relevant pairs (i, m), so the transducer will simply compute a "table" of the values of $2^i \bmod m$ for *all* the relevant pairs. Then the transducer will describe a circuit which "looks up

in the table" to find the values corresponding to the pairs (i, m) which it really needs. This "look up" process is an NC^1 computation.

The second step is an iterated addition (hence NC^1). The third step is also an NC^1 computation once we observe that the absolute value of the sum computed in step two is $O(mk)$: the products jm, $1 \leqq j \leqq n$, can be computed in parallel in order to determine which j equals the sum divided by m; then one subtraction yields the remainder. The result is $a \bmod m$.

As to computing a/m, Alt and Blum [A1B183] provide an elegant NC^1 reduction from computation of the integer quotient to mod computation. □

PROPOSITION 3.2. *The problem of factoring a tiny integer m belongs to* NC^1.

Proof. A log space transducer can precompute the factorizations of all relevant tiny integers. Then use table lookup. □

COROLLARY 3.3. *Computing the prime factorization of the order of a permutation g presented as a product of disjoint cycles belongs to* NC^1.

Proof. By Fact 2.1 $|g|$ is the least common multiple of the orders of the lengths of the cycles in g. Since these lengths are tiny we can factor each of them in parallel, by the previous proposition. Finally, for each relevant prime p in parallel, it suffices to pick the largest power to which p appears in any one of the cycle lengths. Observe that there is only a tiny number of relevant primes p. □

PROPOSITION 3.4. *Let a be any integer and let m be a tiny integer relatively prime to a. Computing an integer b satisfying* $ab \equiv 1 \bmod m$ *belongs to* NC^1.

Proof. For the possible values $1, \cdots, m-1$ of b in parallel we compute ab in NC^1 and then $ab \bmod m$ using Proposition 3.1. We output the value of b which satisfies the proposition. □

PROPOSITION 3.5. *Let* $m, m_1, \cdots, m_r$ *be tiny integers.* NC^1 *contains the problem of computing* $m_1 m_2 \cdots m_r \bmod m$.

Proof. Observe that the following can be precomputed by the log space transducer computing the nth circuit:

(1) A list of all primes $p \leqq n$.

(2) For each integer s, $1 \leqq s \leqq n$, the complete prime factorization of s.

(3) For each prime power p^l, $1 \leqq p^l \leqq n$, $p \neq 2$, an integer τ_{p^l}, $1 \leqq \tau_{p^l} \leqq p^l$, generating the cyclic multiplicative group $\mathbf{Z}_{p^l}^*$ (denoting the group of integers s relatively prime to p^l, $1 \leqq s \leqq p^l$, under multiplication $\bmod p^l$, see [NiZu80]) and for each $s \in \mathbf{Z}_{p^l}^*$, an integer t satisfying $(\tau_{p^l})^t \equiv s \bmod p^l$, $0 \leqq t \leqq p^l$.

(4) For each integer l with $1 < 2^l \leqq n$, and each $s \in \mathbf{Z}_{2^l}^*$, a pair of integers u, v satisfying $(-1)^u 5^v \equiv s \bmod 2^l$ (see [NiZu80]).

(5) For each prime power p^l and each pair s, t of integers, $1 \leqq s, t, p^l \leqq n$, $s^t \bmod p^l$.

Write q for $m_1 m_2 \cdots m_r$. We first show how to compute $q \bmod p^l$ when $p^l \leqq n$. If prime p is odd let $\operatorname{ord}_p(m_i) = j_i$ and

$$(\tau_{p^l})^{k_i} \equiv (\operatorname{rest}_p(m_i)) \bmod p^l, \quad 1 \leqq i \leqq r, \quad 0 \leqq k_i \leqq p^l.$$

The integers j_i and the discrete logarithms k_i are effectively computed by table lookup, given the precomputed values above. Then we compute $j = \sum_i j_i$ and $k = \sum_i k_i$, in $O(\log n)$ depth. Clearly $q \equiv p^j (\tau_{p^l})^k \bmod p^l$. Hence assuming $j < l$ we multiply $(\tau_{p^l})^k \bmod p^{l-j}$ by p^j (both read off from the precomputed values), thus obtaining $q \bmod p^l$ in $O(\log n)$ depth. The case $p = 2$ is handled similarly, except that instead of $\mathbf{Z}_{2^l}^*$ being cyclic, for $l > 2$ we have $\mathbf{Z}_{2^l}^* \cong C_2 \times C_{2^{l-2}}$ with $2^l - 1$ and 5 the respective generators in $\mathbf{Z}_{2^l}^*$.

Now to compute $q \bmod m$, we read off the prime factorization of m from the precomputed values. Next we compute $q \bmod p^l$ for each prime power factor of m, as

described above. Finally, for each s in parallel, $0 \leqq s < m$, we test whether $q \equiv s \bmod p^l$ for all prime factors p^l of m simultaneously. By the Chinese Remainder Theorem the unique value of s satisfying all congruences is congruent to $q \bmod m$. □

LEMMA 3.6. *Let $a_1, \cdots, a_r$ be any integers and let m be a tiny integer. NC^1 contains the problem of computing $a_1 \cdots a_r \bmod m$.*

Proof. Compute $m_i = a_i \bmod m$, for $1 \leqq i \leqq r$ in parallel, using Proposition 3.1. Then apply Proposition 3.5. □

COROLLARY 3.7. *Let g be a permutation presented as a product of disjoint cycles and let $a_1, \cdots, a_k$ be any integers. Computing the permutation g^a, where $a = a_1 a_2 \cdots a_k$, belongs to NC^1.*

Proof. One raises each disjoint cycle c in the permutation g to the given power independently, in parallel. Since the order $|c|$ of a cycle equals the number of points in the cycle, $|c|$ is a tiny integer. Hence $\prod a_i \bmod |c|$ can be computed using Lemma 3.6, and given this number c^a is easily obtained. □

We now wish to show how to locate all Sylow subgroups of an Abelian group in NC^1. Our ability to do this will be crucial to us later. We borrow the following characterization of the Sylow subgroups of an Abelian group from Morenz [Mo84], who used it to develop a polynomial time sequential algorithm for AISO [Mo84].

LEMMA 3.8 [Mo84]. *Let $G = \langle g_1, \cdots, g_r \rangle \leqq \mathrm{Sym}(\Omega)$ be Abelian. For any prime p,*

$$(\dagger) \qquad S_p(G) = \langle g_1^{\alpha_1}, \cdots, g_r^{\alpha_r} \rangle$$

where

$$\alpha_j = \mathrm{rest}_p |g_j|, \qquad 1 \leqq j \leqq r.$$

Proof. Write $K = \langle g_1^{\alpha_1}, \cdots, g_r^{\alpha_r} \rangle$ and let

$$H = \{g \in G \mid \gcd(p, |g|) = 1\}.$$

By Fact 2.3 we know that $K \cap H = \{1\}$ and that H is a group. We claim that $G = K \times H$. To prove this it suffices to show that $g_j \in \langle K, H \rangle$, $1 \leqq j \leqq r$. Now fix j. Writing $\beta = p^{\mathrm{ord}_p |g_j|}$, we observe that there exist $s, t \in \mathbf{Z}$ satisfying

$$1 = s\alpha_j + t\beta.$$

Now p does not divide $|g_j^{\beta t}|$ since it follows from $(g_j^{\beta t})^{\alpha_j} = 1$ that $|g_j^{\beta t}|$ divides α_j. Hence $g_j^{\beta t} \in H$. But then $g_j = (g_j^{\alpha_j})^s g_j^{\beta t} \in \langle K, H \rangle$, and our claim is proved.

We have seen that K only contains p-elements, and that all p-elements in G belong to K by virtue of $G \cong K \times H$. We conclude by Fact 2.4. □

We mention that (†) also yields the unique Sylow p-subgroup in the more general case of a nilpotent group G (see [Mc84]). (A finite nilpotent group G is characterized by the uniqueness of a Sylow p-subgroup, for each prime p.)

COROLLARY 3.9. *NC^1 contains the problem of computing generating permutations for every nontrivial Sylow p-subgroup of an Abelian group $G = \langle g_1, \cdots, g_r \rangle \leqq \mathrm{Sym}(\Omega)$.*

Proof. By Lemma 3.8, only the primes dividing the order of some generator of G will yield nontrivial Sylow subgroups. Using Corollary 3.3, we compute all these primes. Then for each such prime p in parallel we compute $S_p(G)$ as given by (†) above, appealing to Corollary 3.7. □

COROLLARY 3.10. CYCLICGM $\in NC^1$.

Proof. With G the cyclic group specified by generating permutations and h the test permutation, let $H = \langle G \cup \{h\} \rangle$ be Abelian (this condition is easily checked in NC^1 by verifying that h commutes with each generator). Then $h \in G$ if and only if $G = H$ if and only if the Sylow p-subgroup of G equals the Sylow p-subgroup of H for every

prime p. It follows from Lemma 3.8 that $h \in G$ if and only if $h^{\text{rest}_p(|h|)} \in S_p(G)$ for every prime p.

Hence it suffices, for each prime p dividing $|h|$ in parallel, to compute $S_p(G)$ using the previous corollary and to perform the appropriate membership test. We need four observations in order to conclude the proof. First, the appropriate powers of h can be computed by Corollary 3.7. Second, a subgroup of a cyclic group is cyclic; therefore $S_p(G)$ is a cyclic p-group. Third, a cyclic p-group is generated by any one of its generators of largest order (Fact 2.3). Fourth, checking whether a permutation can be expressed as a power of a p-element is an NC^1 computation since a p-element has tiny order by Fact 2.1. □

The algorithm implicit in the above argument can be strengthened to find a representation of h in terms of the generators of G. This is done by expressing, for each p, $h^{\text{rest}_p(|h|)}$ in terms of a generator for $S_p(G)$ and then, in NC^2, expressing h as $\prod_p (h^{\text{rest}_p(|h|)})^{i_p}$ for i_p satisfying $i_p \text{rest}_p(|h|) \equiv 1 \bmod p^{\text{ord}_p(|h|)}$ (Corollary 3.7). We thus have the following.

PROPOSITION 3.11. CYCLICGMX $\in NC^2$. □

In the remainder of this section we exhibit NC^1 reducibilities among graph connectivity problems. (No new upper bounds are developed.) Recall problems DGAP and LDGAP, which consist respectively of determining accessibility and accessibility within a prescribed number of steps from a node u to a node v in a directed graph.

Fact 3.12. DGAP is complete for NSPACE($\log n$) under NC^1 reducibility [Jo75].

The next lemma provides a reduction from computing a path to determining accessibility.

LEMMA 3.13. DPATH $\leqq$ LDGAP $\leqq$ DGAP.

Proof. The last reduction follows from Fact 3.12 since obviously LDGAP $\in$ NSPACE ($\log n$). Turning to DPATH $\leqq$ LDGAP, let the graph have m nodes and assume that we are looking for a path between nodes 1 and m. We first observe that in $O(\log n)$ depth we can determine, for each node i in parallel, the length $l(i)$ of a shortest path from node i to node m: for fixed i we use parallel oracle gates for LDGAP with $k = 0, 1, \cdots, m$, and retain the smallest value of k for which LDGAP returns "yes" (or ∞ if no path is found). Now assume that $l(1) \neq \infty$ and denote by P the lexicographically least shortest path from 1 to m.

Define a new graph G^*, of outdegree 1, as follows. Delete from G all the edges except, for each i, $1 \leqq i \leqq m$, an edge (i, j) for the least j satisfying $l(j) = l(i) - 1$ (if such a j exists). We claim that the resulting graph G^* still contains P.

To see this, assume otherwise and let (i, j) be the first edge along the path P to have been deleted from G. Since (i, j) belongs to a shortest path from 1 to m, $l(j) = l(i) - 1$. Hence (i, j) was deleted because an edge (i, k) with $k < j$ and $l(k) = l(i) - 1$ was found. In other words (i, k) is the first edge of a shortest path from i to m. But then there is a shortest path from 1 to m, via (i, k), which is lexicographically less than P. This contradicts the choice of P, and our claim is proved.

Since G^* has outdegree 1, we can compute the (unique) path from 1 to m to G^* as follows. For each i in parallel, we compute the length of the shortest path from 1 to i, in $O(\log n)$ depth, using LDGAP oracle nodes as before. Clearly there is a unique node at distance k from 1, $0 \leqq k \leqq l(1)$. Thus the nodes at distances $0, 1, \cdots, l(1)$ make up P, in that order. This completes the proof. □

If a graph is undirected, computing paths reduces to the directed case:

COROLLARY 3.14. UPATH $\leqq$ DPATH $\leqq$ DGAP.

Recall COMPONENT, which consists of computing the connected components of an undirected graph.

PROPOSITION 3.15. COMPONENT $\leqq$ UGAP.

Proof. Using parallel oracle gates for the Undirected Graph Accessibility Problem we compute, for each node u, the connected component containing u. At this point we have all the components available, but each one may be repeated many times. Imposing an arbitrary order on the set of nodes, we can remove duplicate sets by computing in NC^1, for each node u, the smallest node v connected to u; then the connected component containing u is output iff $u = v$. The result is a partition of the set of nodes into connected components. □

The following corollary to this proposition is folklore. Let $G = \langle g_1, \cdots, g_r \rangle \leqq$ Sym (Ω). We associate with G and its generating set an edge-labelled undirected graph $\chi(\{g_1, \cdots, g_r\})$ defined as follows. (This graph will also be referred to in § 4.) The vertex set of $\chi(\{g_1, \cdots, g_r\})$ is Ω. The edge (α, β) belongs to the graph $\chi(\{g_1, \cdots, g_r\})$ if and only if $\alpha^{g_k} = \beta$ for some generator g_k; in that case (α, β) is labelled k (for definiteness pick the least possible k). Recall that problem ORBIT consists of computing the orbit partition of the set Ω of points acted upon by a permutation group G.

COROLLARY 3.16. ORBIT $\leqq$ UGAP.

Proof. An induction shows that two points in Ω belong to the same G-orbit if and only if they belong to the same connected component in $\chi(\{g_1, \cdots, g_r\})$. Now $\chi(\{g_1, \cdots, g_r\})$ is easily constructed and we conclude the proof by Proposition 3.15. □

We have discussed the Graph Accessibility and the Path Finding problems for directed and for undirected graphs. In the special case of directed graphs of outdegree 1, the following apply.

Fact 3.17. DGAP1 is complete for *FL* under NC^1 reducibility [Jo75].

COROLLARY 3.18. DPATH1 $\leqq$ DGAP1.

Fact 3.19 [Bo77]. NSPACE $(\log n) \subseteq NC^2$.

We note that Fact 3.19 and Proposition 2.6 imply that $\langle \text{NSPACE } (\log n) \rangle \subseteq NC^2$. We record the (well-known) upper bounds resulting from the previous facts and propositions as shown below.

PROPOSITION 3.20. $\{$DPATH1, DGAP1$\} \subseteq \langle \text{DSPACE } (\log n) \rangle \subseteq NC^2$. $\{$ORBIT, COMPONENT, UPATH, UGAP, DPATH, DGAP$\} \subseteq \langle \text{NSPACE } (\log n) \rangle \subseteq NC^2$.

We conclude this section by pointing out that two simple operations are complete for *FL*.

PROPOSITION 3.21. *Converting a permutation from pointwise representation to disjoint cycle representation and computing the disjoint cycle representation of the product of two permutations are each complete for FL under NC^1 reducibility.*

The proof appears in [CoMc86]. The fact that these problems are complete for *FL* shows that they are not in NC^1, unless $NC^1 = FL$.

4. Transitive Abelian groups. Parallel algorithms developed in § 7 break up an Abelian permutation group problem into subproblems involving transitive Abelian groups. This section discusses such subproblems. The following two propositions are implicit in [Wi64].

PROPOSITION 4.1. *Let $G \leqq$ Sym (Ω) be Abelian and transitive on Ω. Pick arbitrary $g_1, g_2 \in G$ and let $\alpha \in \Omega$. Then $g_1 = g_2$ iff $\alpha^{g_1} = \alpha^{g_2}$.*

Proof. Pick any $\beta \in \Omega$. Since G is transitive there exists $g \in G$ mapping α to β, written $\alpha^g = \beta$. Then G_α and G_β are conjugate by g, that is,

$$g^{-1} G_\alpha g = G_\beta.$$

But by commutativity of G, in fact $G_\alpha = G_\beta$. Since β was arbitrary, the elements in G_α must fix all letters in Ω. In other words $G_\alpha = \{1\}$. The result follows since $\alpha^{g_1} = \alpha^{g_2}$ implies $g_1 g_2^{-1} \in G_\alpha$. □

PROPOSITION 4.2. *Let* $G \leqq \mathrm{Sym}(\Omega)$ *be Abelian and transitive on* Ω. *Then* $|G| = |\Omega|$.

Proof. From the proof of Proposition 4.1 we have $|G_\alpha| = 1$ for any $\alpha \in \Omega$. Now $|G| = |\Omega|$ follows from the equality

$$|G| = |G_\alpha||\alpha^G|,$$

which holds in arbitrary groups since

$$hG_\alpha = gG_\alpha \Leftrightarrow \alpha^h = \alpha^g$$

and any two cosets hG_α and gG_α are either equal or disjoint. □

Proposition 4.2 yields a trivial algorithm to compute the order of a transitive Abelian group.

COROLLARY 4.3. TAORDER $\in NC^1$.

COROLLARY 4.4. *Let* $G = \langle g_1, \cdots, g_r \rangle \leqq \mathrm{Sym}(\Omega)$ *be Abelian and transitive on* Ω *and let* $h \in \mathrm{Sym}(\Omega)$. *Then* $h \in G$ *iff* $hg_i = g_i h$ *for* $1 \leqq i \leqq r$.

Proof. Clearly $h \in G$ if and only if $|G| = |H|$, where $H = \langle g_1, \cdots, g_r, h \rangle$. Since H is transitive, by Proposition 4.2 this condition holds iff H is Abelian. □

Exploiting Corollary 4.4 to test membership in a transitive Abelian group,

COROLLARY 4.5. TAGM $\in NC^1$.

Problem STAGMX consists of determining a formal expression (if any) for a test permutation in terms of the generators of a subgroup of a transitive Abelian group.

LEMMA 4.6. STAGMX $\leqq$ {UPATH, DGAP1} $\leqq$ DGAP.

Proof. The second reduction follows immediately from Corollary 3.14, so let it be required to determine membership of a permutation h in a group $G = \langle g_1, \cdots, g_r \rangle \leqq H \leqq \mathrm{Sym}(\Omega)$ with H Abelian and transitive on Ω. Consider the graph $\chi(\{g_1, \cdots, g_r\})$ defined before Corollary 3.16. Pick any $\alpha \in \Omega$, and let $\beta = \alpha^h$. Using an oracle gate for UPATH, we determine a path P from α to β in $\chi(\{g_1, \cdots, g_r\})$ (if no path exists then $h \notin G$). We claim that $h \in G$ if and only if h equals the product of the generators corresponding to the labels along P (in any order since G is Abelian). This follows from Proposition 4.1 since the product clearly sends α to β, and the conclusion of Propostion 4.1 holds in particular for any two elements of $G \subseteq H$.

This concludes the proof since if $h \in G$ then the set of generating permutations corresponding to the labels along P constitutes the desired expression for h in terms of the generators of G. Since computing the product of the permutations along P (required to test membership) trivially belongs to FL, this reduces to DGAP1 by Fact 3.17. □

Similarly, for other problems involving subgroups of transitive Abelian groups, we have the following.

PROPOSITION 4.7. {STAGM, STAORDER, STASET, STAINTER} $\leqq$ {UPATH, DGAP1} $\leqq$ DGAP.

Proof. Because transitive Abelian permutation groups have tiny order (Proposition 4.2), the technique used in the proof of Lemma 4.6 can be used to list all elements of any subgroup G (permuting a set of points Ω) of an Abelian group H transitive on Ω. This requires the UPATH and DGAP1 oracles. Using the list of elements of G, STAGM, STAORDER, STASET and STAINTER are easily solved. □

It follows from Proposition 3.20 that problems in Lemma 4.6 and in Proposition 4.7 belong to NC^2.

5. Hardness of the group membership problem. Recall that LCON, LCONX and LCONNULL consist of determining solvability, of computing a solution, and of computing a spanning set for the **Z**-module of all solutions to a (homogeneous in the case of LCONNULL) system of linear equations modulo an appropriately restricted

integer. Their "Abelian group counterparts" are AGM, AGMX and AGP, which consist of determining membership, of expressing the test permutation in terms of the generating permutations, and of computing a presentation of an Abelian permutation group. (We discuss the concept of the presentation of an Abelian group below.)

In this section we show that LCON, LCONX and LCONNULL reduce in NC^1 to AGM, AGMX and AGP respectively. Then we prove that LCON $\leqq$ LCONNULL and that LCONX "almost" reduces to LCONNULL. Finally we show that LCON, hence LCONX, LCONNULL, AGM, AGMX and AGP, are NC^1-hard for NSPACE ($\log n$). This implies that a breakthrough would be required to solve any one of these problems better than in NC^2.

First we need to define the notion of a presentation of a finitely generated Abelian group. The free Abelian group with r generators is the set $\mathbf{Z}^r$ under vector addition. If G is any Abelian group with generators $\{g_1, \cdots, g_r\}$, then the homomorphism $\psi: \mathbf{Z}^r \to G$ given by $\psi(X) = g_1^{x_1} g_2^{x_2} \cdots g_r^{x_r}$ maps onto G (where $X = (x_1, \cdots, x_r)$). A *relator* of G (with respect to generators $g_1, \cdots, g_r$) is any vector $X \in \operatorname{Ker} \psi$. Thus X is a relator of G iff $g_1^{x_1} \cdots g_r^{x_r} = 1$. A *presentation* of G (with respect to generators $g_1, \cdots, g_r$) is any finite set $\{X_1, \cdots, X_m\}$ of relators such that $\operatorname{Ker} \psi = \langle X_1, \cdots, X_m \rangle$. Note that a presentation of G determines G up to isomorphism, since G is isomorphic to $\mathbf{Z}^r / \operatorname{Ker} \psi$.

Returning to the example given in § 2, we find that if $g_1 = (12)(345678)(ABCDE)(GH)$ and $g_2 = (1G2H)(ABCDE)$, then a presentation of G is $\{(30, 0), (6, 4)\}$. (This can be verified by using the Todd-Coxeter algorithm [ToCo37].)

LEMMA 5.1. LCONNULL $\leqq$ AGP.

Proof. Let $A \in \mathbf{Z}^{k \times m}$ and let $p_1^{s_1} \cdots p_l^{s_l}$ be the prime factorization of q, a positive integer with tiny prime factors. We wish to compute a spanning set for

$$\text{NULL} = \{X \in \mathbf{Z}^m \mid AX \equiv 0 \bmod q\}.$$

Consider a permutation π with disjoint cycle representation $c_1 c_2 \cdots c_l$, where c_i is a cycle of length $p_i^{s_i}$, $1 \leqq i \leqq l$. The order of π is q by Fact 2.1, while its degree is $\sum_i p_i^{s_i} \leqq n$. Now define k copies of π, denoted $\pi_1, \cdots, \pi_k$, each acting on a separate set of points. The group G effecting the reduction to AGP will be a subgroup of the Abelian group $\langle \pi_1, \cdots, \pi_k \rangle$. Hence its degree will be at most $kn < n^2$.

Now observe that $X \in \text{NULL}$ iff $\sum_{j=1}^m x_j A_j \equiv 0 \bmod q$, where x_j is the jth entry in X and A_j is the jth column of A. This motivates defining G as follows. For each column A_j of $A = (a_{ij})$, $1 \leqq j \leqq m$, include the permutation

$$g_j = \pi_1^{a_{1j}} \pi_2^{a_{2j}} \cdots \pi_k^{a_{kj}},$$

obtainable in NC^1 by Corollary 3.7, as a generator of G. Then

$$g_1^{x_1} \cdots g_m^{x_m} = 1$$

iff $X \in \text{NULL}$, and hence NULL is the set of relators of G. Therefore a presentation of G is precisely a spanning set for NULL. □

PROPOSITION 5.2. LCON $\leqq$ AGM; LCONX $\leqq$ AGMX.

Proof. In both reductions we generate the same group G as that used in reducing LCONNULL to AGP above. In the notation of that lemma, we let the test permutation be

$$h = \pi_1^{b_1} \cdots \pi_k^{b_k}$$

where b_i is the ith entry of the vector B in the nonhomogeneous LCON and LCONX instances. It should be clear that

$$h = g_1^{x_1} \cdots g_m^{x_m}$$

if and only if $AX \equiv B \bmod q$ for $X = (x_i) \in \mathbf{Z}^m$. Since a formal expression for h in terms of $g_1, \cdots, g_m, g_1^{-1}, \cdots, g_m^{-1}$ easily yields such an X, AGM and AGMX oracle gates, respectively, solve LCON and LCONX in NC^1, providing the desired reductions. □

Next we examine the relative parallel complexities of LCON, of LCONX and of LCONNULL. The parallel complexities of similar linear algebra problems, under the assumption of a field as the underlying domain, have been compared in [BoGaHo82] and in [Ga84a] (using a different parallel computation model). The following lemma draws some ideas from [Ga84a].

LEMMA 5.3. LCON $\leqq$ LCONNULL *and* LCONX $\leqq$ {ITPROD, LCONNULL}, *where* ITPROD *is the iterated product problem, consisting of computing the exact product of a number of integers specified in binary.*

Proof. Let $A \in \mathbf{Z}^{k \times m}$, let $B \in \mathbf{Z}^k$, and let $q \in \mathbf{Z}$ have tiny factors. With $X_i \in \mathbf{Z}^m$ and $y_i \in \mathbf{Z}$, $1 \leqq i \leqq t$, let

$$\begin{bmatrix} X_1 \\ y_1 \end{bmatrix}, \quad \begin{bmatrix} X_2 \\ y_2 \end{bmatrix}, \cdots, \begin{bmatrix} X_t \\ y_t \end{bmatrix}$$

form a generating set for the $\mathbf{Z}$-module of all $\begin{bmatrix} X \\ y \end{bmatrix} \in \mathbf{Z}^{m+1}$ satisfying

$$AX + yB \equiv 0 \bmod q.$$

Then the system $AX \equiv B \bmod q$ has a solution if and only if there exist $u_1, \cdots, u_t \in \mathbf{Z}$ satisfying

$$(*) \qquad u_1 y_1 + \cdots + u_t y_t \equiv 1 \bmod q$$

and moreover, if such u_i's exist then

$$-u_1 X_1 - u_2 X_2 \cdots - u_t X_t$$

is a solution to $AX \equiv B \bmod q$.

If (*) is solvable then for each prime p dividing q there exists i, $1 \leqq i \leqq t$, such that p does not divide y_i. (The converse is also true as will become clear.) For each prime p dividing q then choose such a y_i and call it γ_p (or declare that the linear congruences have no solution). If for prime p dividing q we let $\alpha_p \in \mathbf{Z}$ satisfy

$$\alpha_p \gamma_p \operatorname{rest}_p(q) \equiv 1 \bmod p^{\operatorname{ord}_p(q)}$$

then

$$\sum_{\text{prime } p \text{ divides } q} \alpha_p \gamma_p \operatorname{rest}_p(q) \equiv 1 \bmod q$$

by the Chinese Remainder Theorem. Hence

$$u_i = \sum_{\gamma_p \text{ is } y_i} \alpha_p \operatorname{rest}_p(q), \qquad 1 \leqq i \leqq t$$

satisfies (*).

We now describe how the above yields the desired reductions. Since q has tiny factors, testing for each prime p dividing q whether there exists i, $1 \leqq i \leqq t$, such that p does not divide y_i, is an NC^1 computation. Hence

$$\text{LCON} \leqq \text{LCONNULL}.$$

To solve LCONX we must actually compute the u_is. Now computing α_p is an NC^1 computation (Lemma 3.6 and Proposition 3.4). Hence all steps are straightforward except the computation of an integer x satisfying $x \equiv \operatorname{rest}_p(q) \bmod q$, for fixed prime p dividing q. We use the ITPROD oracle to compute $\operatorname{rest}_p(q)$ exactly. This concludes the proof. □

We conclude § 5 by establishing that the Directed Graph Accessibility Problem DGAP reduces to LCON. Combinined with Proposition 5.2 and by virtue of Fact 3.12, this will prove that AGM is hard for $\langle \text{NSPACE}(\log n)\rangle$ under NC^1 reducibility.

Recall problem ZERODET which consists of determining whether an $n \times n$ matrix A of n-bit integers is singular. We have the following.

Fact 5.4. [Co85] DGAP $\leqq$ ZERODET.

LEMMA 5.5. ZERODET $\leqq$ LCON.

Proof. To determine whether an $n \times n$ matrix of n-bit integers A has zero determinant, we test whether $\det(A) \equiv 0 \bmod p$ for sufficiently many primes p in parallel. Note that $\det(A) \equiv 0 \bmod p$ iff for some i, the ith column of A is a linear combination of the remaining columns over $\mathbf{Z}_p$. This condition can be checked by one call to the LCON oracle for each i, in parallel. □

THEOREM 5.6. LCON, LCONX, LCONNULL, AGM, AGMX *and* AGP *are hard for* $\langle \text{NSPACE}(\log n)\rangle$ *under* NC^1 *reducibility* (*and thus under* "log *space Cook reducibility*").

Proof. DGAP is complete for $\langle \text{NSPACE}(\log n)\rangle$ (Fact 3.12). Now

$$\text{DGAP} \leqq \text{ZERODET} \leqq \text{LCON} \leqq \text{LCONNULL}$$

by Fact 5.4 and Lemmas 5.5 and 5.3. Now clearly LCON $\leqq$ LCONX, and LCON, LCONX and LCONNULL reduce to AGM, AGMX and AGP, respectively, by Lemma 5.1 and Proposition 5.2. The space reducibility claim follows because log space transducers can simulate NC^1 circuits [Bo77]. □

Let us write EAGM for the Group Membership problem restricted to the case of elementary Abelian permutation groups (an elementary Abelian group is an Abelian P-group all of whose nontrivial elements have prime order). We have in fact shown that DGAP $\leqq$ EAGM. Hence EAGM is itself NC^1-hard for $\langle \text{NSPACE}(\log n)\rangle$.

For future reference we record a consequence of Corollary 3.16, of Lemma 4.6 and of Propostion 4.7 combined with the results of this section.

PROPOSITION 5.7. {ORBIT, STAGMX, STASET} $\leqq$ DGAP $\leqq$ LCON $\leqq$ AGM.

6. Reductions among the group-theoretic problems. In this section we examine Abelian permutation group problems which reduce to AGP (consisting of computing a generator-relator presentation of an Abelian permutation group). We consider the class $\langle \text{AGM}\rangle$ in § 6.1, the class $\langle \text{AINTER}\rangle$ in § 6.2, and we show that $\langle \text{AGP}\rangle$ contains both classes in § 6.3. We have not yet resolved whether any of the inclusions

$$\langle \text{AGM}\rangle \subseteq \langle \text{AINTER}\rangle \subseteq \langle \text{AGP}\rangle$$

could be replaced by an equality.

6.1. Problems NC^1-equivalent to Abelian group membership. This subsection deals with determining membership, testing isomorphism, and computing order for Abelian and for Abelian P-groups. Specifically, we are concerned with problems AGM, APGM, AISO, APISO, AORDER and APORDER. We develop NC^1 reductions which prove that these six problems are NC^1-equivalent.

We begin by observing that testing whether a group is an Abelian or an Abelian P-group is a simple matter. Indeed, to check whether $G = \langle g_1, \cdots, g_r\rangle$ is Abelian it clearly suffices to determine whether $g_j^{-1} g_i g_j = g_i$ for all i and j. Once G is known to be Abelian, it suffices to check that each generator is a p-element (for the same p) to conclude that G is an Abelian p-group. This allows the following proposition, which will be applied both in this section and in the next one.

PROPOSITION 6.1.

$$\text{AGM} \leqq \text{AORDER } \textit{and } \text{APGM} \leqq \text{APORDER},$$
$$\text{AGM} \leqq \text{AISO } \textit{and } \text{APGM} \leqq \text{APISO},$$
$$\text{AGM} \leqq \text{AINTER } \textit{and } \text{APGM} \leqq \text{APINTER}.$$

Proof. Clearly

$$\begin{aligned} h \in \langle g_1, \cdots, g_r \rangle &\Leftrightarrow |\langle g_1, \cdots, g_r \rangle| = |\langle g_1, \cdots, g_r, h \rangle| \\ &\Leftrightarrow \langle g_1, \cdots, g_r \rangle \cong \langle g_1, \cdots, g_r, h \rangle \\ &\Leftrightarrow h \in \langle g_1, \cdots, g_r \rangle \cap \langle h \rangle. \end{aligned}$$

Hence both computing the orders of two Abelian groups (resp. Abelian p-groups) and testing isomorphism of two Abelian groups (resp. Abelian p-groups) can answer the membership question for h, provided it can be checked that adding h to the generators of G does not destroy the property that the group generated is Abelian (resp. an Abelian p-group) (if h destroys this property then $h \notin G$).

As for the reductions to computing intersections, assume that $\langle g_1, \cdots, g_r, h \rangle$ is Abelian (resp. an Abelian p-group). Observe that the output from an AINTER (resp. APINTER) oracle on the input groups $\langle g_1, \cdots, g_r \rangle$ and $\langle h \rangle$ will be a list of permutations $\langle h_1, \cdots, h_s \rangle$ generating a subgroup of $\langle h \rangle$. This subgroup is therefore cyclic, being a subgroup of a cyclic group. Hence AGM $\leqq$ {AINTER, CYCLICGM} (resp. APGM $\leqq$ {APINTER, CYCLICGM}) and the results follow from Corollary 3.10. □

PROPOSITION 6.2. AISO $\leqq$ APISO; AORDER $\leqq$ APORDER.

Proof. Consider the first reduction. Since Abelian groups are direct products of their Sylow subgroups (Fact 2.4), two Abelian groups $G = \langle g_1, \cdots, g_r \rangle$ and $H = \langle h_1, \cdots, h_s \rangle$ are isomorphic if and only if $S_p(G) \cong S_p(H)$ for all relevant primes p. By Corollary 3.9 we can compute generators for all groups $S_p(G)$ and $S_p(H)$ in NC^1. Using oracle gates for APISO, it suffices to check whether $S_p(G) \cong S_p(H)$ for each relevant prime p in parallel, and to fan in the outcomes.

The second reduction is equally obvious since the order of a direct product is the product of the orders of the factors in the direct product. Here no fanning in is even required below the APORDER oracle gates since p is the only prime dividing the order of the Sylow p-subgroup of G. □

In the same spirit we have Proposition 6.3.

PROPOSITION 6.3. AGM $\leqq$ APGM.

Proof. Similar to the proof of Corollary 3.10, except that here the relevant membership tests are performed by APGM oracles. □

In [Mo84], Morenz shows that AISO $\in P$. He proceeds by locating each Sylow subgroup $S_p(G)$ of G using Lemma 3.8, and then by computing the cyclic factors in the cyclic decomposition of $S_p(G)$. His method does not parallelize directly. To solve APISO in parallel we will require Proposition 6.4.

PROPOSITION 6.4. *Let* $G = \langle g_1, \cdots, g_r \rangle$ *and* $H = \langle h_1, \cdots, h_s \rangle$ *be Abelian p-groups, and let* $p^k = \max\{|g_1|, \cdots, |g_r|, |h_1|, \cdots, |h_s|\}$. *Then* $G \cong H$ *if and only if*

$$|\langle g_1^{p^i}, \cdots, g_r^{p^i} \rangle| = |\langle h_1^{p^i}, \cdots, h_s^{p^i} \rangle|, \qquad 0 \leqq i < k.$$

Proof. Define $G_i = \{g^{p^i} \mid g \in G\}$ and $H_i = \{h^{p^i} \mid h \in H\}$, $0 \leqq i < k$. Observe that $G_i = \langle g_1^{p^i}, \cdots, g_r^{p^i} \rangle$. It suffices to show that the signature of G equals the signature of H (Fact 2.5) if and only if $|G_i| = |H_i|$ for all i. Now if $G \cong H$ then clearly $|G_i| = |H_i|$ for each i, since $|G_i|$ and $|H_i|$ are isomorphism invariants. For the converse, assume throughout the rest of this proof that $|G_i| = |H_i|$ for all i.

From the definition of k and Fact 2.5, we know that the signature of G and the signature of H each contain at most k nonzero entries. When we write $(\delta_1, \cdots, \delta_k)$ and $(\gamma_1, \cdots, \gamma_k)$, respectively, for these two signatures, it follows from the definition of G_i and Fact 2.5 that the signature of G_i, $0 \leqq i < k$, is simply given by

$$(\delta_{i+1}, \cdots, \delta_k).$$

Hence

$$|G_i| = p^{\delta_{i+1}+2\delta_{i+2}+\cdots+(k-i)\delta_k}$$

and similarly,

$$|H_i| = p^{\gamma_{i+1}+2\gamma_{i+2}+\cdots+(k-i)\gamma_k}$$

for $0 \leqq i < k$. A straightforward induction on j therefore shows that if $|G_{k-i}| = |H_{k-i}|$ for $i = 1, \cdots, j$ then $(\delta_{k-j+1}, \cdots, \delta_k) = (\gamma_{k-j+1}, \cdots, \gamma_k)$. We conclude, setting $j = k$, that the signatures of G and of H are the same. □

LEMMA 6.5. APISO $\leqq$ AORDER.

Proof. We apply Proposition 6.4. (Note that k as defined in that proposition is tiny.) Generators for $G_i = \langle g_1^{p^i}, \cdots, g_r^{p^i} \rangle$ and for $H_i = \langle h_1^{p^i}, \cdots, h_s^{p^i} \rangle$ are obtained for $1 \leqq i < k$ in parallel, by Corollary 3.7. Then, again in parallel, oracle gates for AORDER are used to compute $|G_i|$ and $|H_i|$ for each i. Finally the tests for equality of $|G_i|$ and $|H_i|$ are made, and the outcomes fanned in. □

PROPOSITION 6.6. *Let $G = \langle g_1, \cdots, g_r \rangle$ be a finite Abelian group. Then*

$$|G| = t_1 t_2 \cdots t_r$$

where t_j is the least positive integer such that

$$g_j^{t_j} \in \langle g_{j+1}, \cdots, g_r \rangle, \qquad 1 \leqq j \leqq r$$

(such a $t_j \leqq |g_j|$ must exist for each j).

Proof. We show by induction on r that there is a unique way to write $g \in G$ as $g_1^{s_1} g_2^{s_2} \cdots g_r^{s_r}$ with $0 \leqq s_j < t_j$, $1 \leqq j \leqq r$. The base case $r = 1$ is clear since then G is cyclic. So assume the statement true for $H = \langle g_2, \cdots, g_r \rangle$. To see that an arbitrary $g \in G$ is expressible in the desired form, consider any expression for g in terms of the generators where the exponent α of g_1 is nonnegative. Writing $\alpha = ut_1 + v$ for $0 \leqq v < t_1$ and using the expression for $g_1^{t_1}$ in terms of $g_2, \cdots, g_r$ we find that $g = g_1^v h$ for $h \in H$. Hence g is expressible by the induction hypothesis.

To see that any $g \in G$ is expressible uniquely, assume to the contrary that $g = g_1^{\alpha} h_1 = g_1^{\beta} h_2$ with $h_1, h_2 \in H$ and $0 \leqq \alpha, \beta < t_1$. If we can show $\alpha = \beta$ we are done, since then $h_1 = h_2$ is uniquely expressible by the induction hypothesis. But $g_1^{|\alpha-\beta|} \in H$ so $|\alpha - \beta|$ cannot be positive by choice of t_1. Hence $\alpha - \beta = 0$. □

LEMMA 6.7. APORDER $\leqq$ APGM.

Proof. Let G be a P-group. We apply Proposition 6.6. We first compute p such that G is a p-group (Corollary 3.3). Then we compute t_j, $1 \leqq j \leqq r$, for each j in parallel, as follows. For fixed j, all possible values of t_j will be tried, that is, oracle gates for APGM will be used to test membership of $g_j^{t_j}$ in the p-group $\langle g_{j+1}, \cdots, g_r \rangle$ for $t_j = 1, p, p^2, \cdots, |g_j|$ in parallel ($|g_j|$ is tiny by Fact 2.1). Using an additional $O(\log n)$ depth, the smallest value of t_j for which membership succeeds is retained. Finally, $l = \sum_j \operatorname{ord}_p(t_j)$ is computed, and $|G| = p^l$. □

THEOREM 6.8. AISO, APISO, AORDER, APORDER, AGM *and* APGM *are each complete for* $\langle \text{AGM} \rangle$ *under* NC^1 *reducibility.*

Proof. Obviously APGM $\leqq$ AGM, and combining Proposition 6.2, Lemmas 6.5 and 6.7 and Proposition 6.1 we have

$$\text{AISO} \leqq \text{APISO} \leqq \text{AORDER} \leqq \text{APORDER} \leqq \text{APGM} \leqq \text{AGM} \leqq \text{AISO}. \qquad \square$$

6.2. The complexity class $\langle$AINTER$\rangle$. Consider the class of problems NC^1-reducible to AINTER, which consists of computing generating permutations for the intersection of two subgroups of an Abelian group. By Proposition 6.1 we know that $\langle \text{AGM} \rangle \subseteq \langle \text{AINTER} \rangle$. However we do not know whether the inclusion should really be an equality. In this section we show that ASET, which consists of computing a generating set for a setwise stabilizer subgroup of an Abelian permutation group, is NC^1-equivalent to AINTER.

PROPOSITION 6.9. *Let $G \leqq \text{Sym}(\Omega)$, let $\Delta \subseteq \Omega$ and let $\{\Gamma_1, \cdots, \Gamma_s\}$ be the G-orbit partition of Ω. Viewing each G^{Γ_i} as a subgroup of* Sym (Ω), *define*

$$H = \langle G^{\Gamma_1}_{[\Gamma_1 \cap \Delta]} \cup G^{\Gamma_2}_{[\Gamma_2 \cap \Delta]} \cup \cdots \cup G^{\Gamma_s}_{[\Gamma_s \cap \Delta]} \rangle.$$

Then $G_{[\Delta]} = G \cap H$.

Proof. If $g = g^{\Gamma_1} g^{\Gamma_2} \cdots g^{\Gamma_s}$ stabilizes Δ setwise then, for each i, $g^{\Gamma_i} \in G^{\Gamma_i}_{[\Gamma_i \cap \Delta]}$ since $\Gamma_i \cap \Gamma_j = \varnothing$, $i \neq j$. The reverse inclusion is clear. $\square$

Recall that STASET is the setwise stabilizer problem restricted to subgroups of transitive Abelian groups.

PROPOSITION 6.10. ASET $\leqq \{\text{ORBIT, STASET, AINTER}\} \leqq$ AINTER.

Proof. The last reduction holds because

$$\{\text{ORBIT, STASET}\} \leqq \text{AGM} \leqq \text{AINTER}$$

by Proposition 5.7 and Proposition 6.1. Turning to the first reduction, observe that H as defined in Proposition 6.9 is Abelian if G is Abelian. Hence parallel ORBIT and STASET oracle gates can compute generating permutations for each $G^{\Gamma_i}_{[\Gamma_i \cap \Delta]}$ as subgroups of Sym (Ω). Finally, an oracle gate for AINTER computes generating permutations for $G_{[\Delta]} = G \cap H$. $\square$

The converse reduction follows by observing that, in the case of arbitrary permutation groups, the corresponding polynomial time reduction (credited to Luks and described in Chapter 6 of [Ho82]) is in fact an NC^1 reduction.

PROPOSITION 6.11. AINTER $\leqq$ ASET.

Proof. Given $G = \langle g_1, \cdots, g_r \rangle$ and $H = \langle h_1, \cdots, h_s \rangle$ subgroups of an Abelian group ($\leqq$Sym (Ω)), we define a new group $K \leqq \text{Sym}(\Omega \times \Omega)$ as follows. To each pair (g, h) where $g \in G$ and $h \in H$ corresponds $k \in K$ whose action on $\Omega \times \Omega$ is defined as $(\alpha, \beta)^k = (\alpha^g, \beta^h)$, $(\alpha, \beta) \in \Omega \times \Omega$. One can see that $K \cong G \times H$ is Abelian and that $K_{[\Delta]} \cong G \cap H$ for $\Delta = \{(\alpha, \alpha) \mid \alpha \in \Omega\} \subseteq \Omega \times \Omega$.

Now since $K \cong G \times H$ we can generate K using $r + s$ permutations, r corresponding to $g_1, \cdots, g_r$ and s to $h_1, \cdots, h_s$. From the disjoint cycle presentation say of g_1, it is possible to write down in NC^1 the disjoint cycle presentation of the corresponding permutation on $\Omega \times \Omega$. Then an ASET oracle gate is used to compute generators for $K_{[\Delta]}$ and it suffices to project along the "first component" of the generators of $K_{[\Delta]}$ to obtain generators for $G \cap H$. $\square$

Proposition 6.12 follows from the previous propositions.

PROPOSITION 6.12. AINTER *and* ASET *are NC^1-equivalent.*

6.3. The complexity class $\langle$AGP$\rangle$. In this section we show that AINTER reduces to AGP, the problem of computing a presentation of an Abelian group. This shows that $\langle \text{AINTER} \rangle \subseteq \langle \text{AGP} \rangle$, and again here we do not know whether equality holds.

PROPOSITION 6.13. AINTER $\leqq \{\text{AGP, DGAP1}\} \leqq$ AGP.

Proof. The second reduction holds because

$$\text{DGAP} \leqq \text{LCON} \leqq \text{LCONNULL} \leqq \text{AGP}$$

by Proposition 5.7 and Lemmas 5.3 and 5.1. Consider the first reduction. Let $g_1, \cdots, g_r$ and $h_1, \cdots, h_s$ generate the groups G and H to be intersected, and let

$$M = \{X \in \mathbf{Z}^{r+s} \mid g_1^{x_1} \cdots g_r^{x_r} h_1^{x_{r+1}} \cdots h_s^{x_{r+s}} = 1\}.$$

Observe that $g_1^{x_1} \cdots g_r^{x_r} \in G \cap H$ if and only if there exists $X \in M$ with $x_1, \cdots, x_r$ as its first r entries. Hence

$$\phi : M \to G \cap H, \qquad X \mapsto g_1^{x_1} \cdots g_r^{x_r}$$

is a group epimorphism. Now if $\{X_1, \cdots, X_m\} \subseteq \mathbf{Z}^{r+s}$ is a presentation of the Abelian permutation group $\langle g_1, \cdots, g_r, h_1, \cdots, h_s\rangle$ then by definition of presentation $M = \langle X_1, \cdots, X_m\rangle$. Therefore

$$G \cap H = \langle \phi(X_1), \cdots, \phi(X_m)\rangle.$$

To conclude the proof we note that $\{X_1, \cdots, X_m\}$ can be obtained by an AGP oracle gate. To compute $\phi(X_j)$ note that raising g_i to the power x_i is an NC^1 computation by Corollary 3.7. The need for a DGAP1 oracle arises in computing the product $g_1^{x_1} \cdots g_r^{x_r}$ (Proposition 3.20 and Fact 3.17). □

7. Membership reduces to solving linear congruences. In this section we reduce AGM, AGMX and AGP to LCON, LCONX and LCONNULL respectively. The converse reductions were established in § 5. Hence it will follow from the results of this section that LCON, LCONX and LCONNULL are NC^1-equivalent to their "Abelian group counterparts" AGM, AGMX and AGP respectively. Figure 7.1 at the end of this section summarizes the relationships between parallel complexities of several problems considered in this paper.

Let $g_1, \cdots, g_r$ generate the Abelian group G and let h be a test permutation. We begin with the aim of reducing problem AGMX, which requires expressing h in terms of the g_i's if possible, to LCONX, which consists of solving a nonhomogeneous system of linear congruences. Let

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_r \end{bmatrix} \in \mathbf{Z}^r$$

and define

$$\psi : \mathbf{Z}^r \to G, \qquad Y \mapsto \prod_j g_j^{y_j}.$$

PROPOSITION 7.1. *$h \in G$ if and only if there exists $Y \in \mathbf{Z}^r$ satisfying $\psi(Y) = h$.*

Now let $\Gamma_1, \cdots, \Gamma_s$ be the orbits of G. Write G_i for G^{Γ_i} and, assuming that h^{Γ_i} is defined, h_i for h^{Γ_i}, $1 \leqq i \leqq s$. We define onto each G_i a homomorphism similar to ψ:

$$\psi_i : \mathbf{Z}^r \to G_i, \qquad Y \mapsto \prod_j (g_j^{\Gamma_i})^{y_j}, \qquad 1 \leqq i \leqq s.$$

Write $K_i = \text{Ker}(\psi_i)$, $1 \leqq i \leqq s$.

PROPOSITION 7.2. *For each $Y \in \mathbf{Z}^r$, $\psi(Y) = h \Leftrightarrow \psi_i(Y) = h_i$ for $1 \leqq i \leqq s$.*

Proof. h is fully determined by $h_1, \cdots, h_s$. □

PROPOSITION 7.3. *Let $B_i \in \mathbf{Z}^r$ be such that $\psi(B_i) = h_i$, $1 \leqq i \leqq s$. Then for each $Y \in \mathbf{Z}^r$, $h = \psi(Y) \Leftrightarrow Y - B_i \in K_i$ for $1 \leqq i \leqq s$.*

Proof.

$$h = \psi(Y) \Leftrightarrow \psi_i(Y) = h_i \quad \text{for } 1 \leqq i \leqq s$$
$$\Leftrightarrow \psi_i(Y) = \psi_i(B_i) \quad \text{for } 1 \leqq i \leqq s$$
$$\Leftrightarrow \psi_i(Y - B_i) = 1 \quad \text{for } 1 \leqq i \leqq s. \qquad \square$$

We now take a closer look at each kernel K_i. For $1 \leqq i \leqq s$ and $1 \leqq j \leqq r$ let K_i^j be the subgroup of K_i consisting of all vectors in K_i whose first $j-1$ components are 0. Define a matrix $A_i = (a_{jk}^{(i)}) \in \mathbf{Z}^{r \times r}$ as follows. For each j, $1 \leqq j \leqq r$ let

$$a_{jj}^{(i)} = \min \{t \in \mathbf{Z} \mid t > 0 \text{ and some } Y_i^{(j)} \text{ in } K_i^j \text{ has } t \text{ as its } j\text{th component}\}.$$

Note that $a_{jj}^{(i)} \leqq |g_j^{\Gamma_i}|$ because the vector with jth entry equal to $|g_j^{\Gamma_i}|$ and zeros elsewhere belongs to K_i. Let A_i be the lower triangular matrix whose columns are $[Y_i^{(1)}, Y_i^{(2)}, \cdots, Y_i^{(r)}]$.

PROPOSITION 7.4. *For* $1 \leqq i \leqq s$, $K_i = \{A_i X \mid X \in \mathbf{Z}^r\}$.

Proof. Observe that the lemma can be restated to say that the columns of A_i generate K_i.

We prove by induction on l that the last l columns of A_i generate K_i^{r-l+1}. When $l = 1$, this is the obvious statement that $Y_i^{(r)}$ generates K_i^r. For $l > 1$, let Y be any vector in K_i^j, where $j = r - l + 1$. Clearly $a_{jj}^{(i)}$ divides the jth component of Y exactly (otherwise a vector with jth entry equal to the remainder of this division would belong to K_i^j, contradicting the choice of $a_{jj}^{(i)}$), and hence $Y - uY_i^{(j)} \in K_i^{j+1}$ for some $u \in \mathbf{Z}$. By the induction hypothesis $Y - uY_i^{(j)}$ is a linear combination of the last $l-1$ columns of A_i, and hence Y is a linear combination of the last l columns of A_i. $\square$

PROPOSITION 7.5. *Let* B_i *and* A_i, $1 \leqq i \leqq s$, *be as in Propositions* 7.3 *and* 7.4. *For each* $Y \in \mathbf{Z}^r$, $h = \psi(Y)$ *if and only if there exist* $X_1, \cdots, X_s \in \mathbf{Z}^r$ *such that*

$$(*) \qquad Y = B_i + A_i X_i, \qquad 1 \leqq i \leqq s.$$

Proof. The proof follows immediately from Propositions 7.3 and 7.4. $\square$

PROPOSITION 7.6. *Let* $q = \text{lcm}(|g_1|, \cdots, |g_r|)$. *First, if* $h \in G$ *then* (*) *is solvable* mod q. *Second, if* $Y, X_1, \cdots, X_s \in \mathbf{Z}^r$ *satisfy* (*) mod q, *then* $\psi(Y) = h$.

Proof. Both statements follow from Propositions 7.1 and 7.5. "First" is obvious. To see "second" note that $K_i \supseteq \{qX \mid X \in \mathbf{Z}^r\}$. $\square$

It is not hard to rewrite (*) mod q as a single system of rs congruences in $rs + r$ unknowns

$$(**) \qquad AX \equiv B \bmod q$$

where $A \in \mathbf{Z}^{rs \times (rs+r)}$, $B \in \mathbf{Z}^{rs}$, and $X \in \mathbf{Z}^{rs+r}$ is made up of the unknown vectors $X_1, \cdots, X_s$, Y. Then from each solution to (**) it is easy to extract Y.

Recall that STAGMX is the problem which consists of computing an expression (if any) for a test permutation in terms of the generating permutations of a subgroup of a transitive Abelian group.

LEMMA 7.7. AGMX $\leqq$ {ORBIT, STAGMX, LCONX} $\leqq$ LCONX.

Proof. The second reduction follows because

$$\{\text{ORBIT, STAGMX}\} \leqq \text{DGAP} \leqq \text{LCON}$$

by Proposition 5.7. Consider the first reduction.

We apply Proposition 7.6. Observe that the factorization of $q = \text{lcm}(|g_1|, \cdots, |g_r|)$ into its tiny prime power factors is easy to compute (Corollary 3.3). Now determining solvability of (**) and solving it for $X \in \mathbf{Z}^{rs+r}$ is an instance of LCONX. Given such

an X, the formal expression corresponding to $\psi(Y)$, for $Y \in \mathbf{Z}^r$ the vector made up of the last r entries of X, yields the desired formal expression for h in terms of the generators of G. It remains to show how to compute $B_i \in \mathbf{Z}^r$ (Proposition 7.3) and $A_i \in \mathbf{Z}^{r\times r}$ (Proposition 7.4), $1 \leqq i \leqq s$.

The first step is to obtain the G-orbit partition $\Gamma_1, \cdots, \Gamma_s$ of Ω using an oracle gate for ORBIT. From then on we work on each i, $1 \leqq i \leqq s$, in parallel. We check that h^{Γ_i} is defined by verifying for each cycle $(\alpha_1 \cdots \alpha_j)$ of h that either $\{\alpha_1, \cdots, \alpha_j\} \subseteq \Gamma_i$ or else that $\{\alpha_1, \cdots, \alpha_j\} \cap \Gamma_i$ is empty. (If h^{Γ_i} is not defined then $h \notin G$.) Then we compute $h_i = h^{\Gamma_i}$ and generators $g_1^{\Gamma_i}, \cdots, g_r^{\Gamma_i}$ for the transitive group $G_i = G^{\Gamma_i}$. Now an oracle gate for STAGMX provides a formal expression for h_i in terms of $g_1^{\Gamma_i}, \cdots, g_r^{\Gamma_i}$ (if $h_i \notin G_i$ then $h \notin G$), hence also the vector B_i satisfying $\psi(B_i) = h_i$.

Finally, to compute A_i, notice that an equivalent definition of $a_{jj}^{(i)}$ (see paragraph preceding Proposition 7.4) is the least positive $t \in \mathbf{Z}$ such that $(g_j^{\Gamma_i}) \in \langle g_{j+1}^{\Gamma_i}, \cdots, g_r^{\Gamma_i} \rangle$. Thus it suffices to try in parallel possible values of t in the range $1 \leqq t \leqq |g_j^{\Gamma_i}| \leqq n$ (Proposition 4.2) and to use STAGMX oracle gates for the membership test, since the groups involved are subgroups of the transitive Abelian group G_i. The complete jth column $Y_i^{(j)}$ of A_i is easily obtained from the formal expression (returned by STAGMX) for $(g_j^{\Gamma_i})^t$ in terms of $g_{j+1}^{\Gamma_i}, \cdots, g_r^{\Gamma_i}$.

One can verify that the total depth of the reduction is $O(\log n)$. □

The proof of Lemma 7.7 also shows the following.

LEMMA 7.8. AGM $\leqq$ LCON.

LEMMA 7.9. AGP $\leqq$ LCONNULL.

Proof. Using the first line of the proof of Lemma 7.7, and Lemma 5.3, it suffices to show AGP $\leqq$ {ORBIT, STAGMX, LCONNULL}. Proceeding as in the proof of Lemma 7.7 using $h = 1$ as the test permutation, the vector B in (**) is 0, so an oracle gate for LCONNULL using (**) as input will provide vectors $Y_1, \cdots, Y_t \in \mathbf{Z}^r$ generating the $\mathbf{Z}$-module

$$M = \{Y \in \mathbf{Z}^r \mid \psi(Y) = 1\}.$$

Then $\{Y_1, \cdots, Y_t\}$ is a presentation for G. □

Proposition 5.2 and Lemmas 5.1, 7.8, 7.7 and 7.9 yield Theorem 7.10.

THEOREM 7.10. LCON, LCONX *and* LCONNULL *are* NC^1*-equivalent to* AGM, AGMX *and* AGP, *respectively.*

Our results concerning the relative parallel complexities of Abelian permutation group problems are summarized in Fig. 7.1.

8. Upper bounds. In this section we prove that LCONNULL $\in NC^3$. Referring to Fig. 7.1 and to Lemma 5.3, we deduce that all problems appearing in the figure belong to NC^3. We conclude by showing how to obtain the standard cyclic decomposition of an Abelian permutation group in NC^4.

The key to the upper bounds in this section is the (known) NC^2 algorithm for the problem of solving a (possibly undetermined) system of linear equations mod p for p a tiny prime. Indeed techniques developed in [BoGaHo82] for *algebraic* circuits (that is, for circuits whose gates perform field operations at unitary cost) yield an NC^1 reduction from the problem of solving such equations mod p (or from the problem of computing a basis for the nullspace of a homogeneous system) to the problems of computing integer matrix determinants and of computing ranks of matrices over $\mathbf{Z}_p$. Now [BoCoPi83] solves the determinant problem in NC^2, and Mulmuley recently obtained a similar upper bound for the computation of ranks over tiny fields [Mu86].

Let $A \in \mathbf{Z}^{k\times m}$ and let $q \in \mathbf{Z}$ have tiny factors. We consider first the case of LCONNULL in which the modulus q is a (tiny) prime power p^l. The following lemma

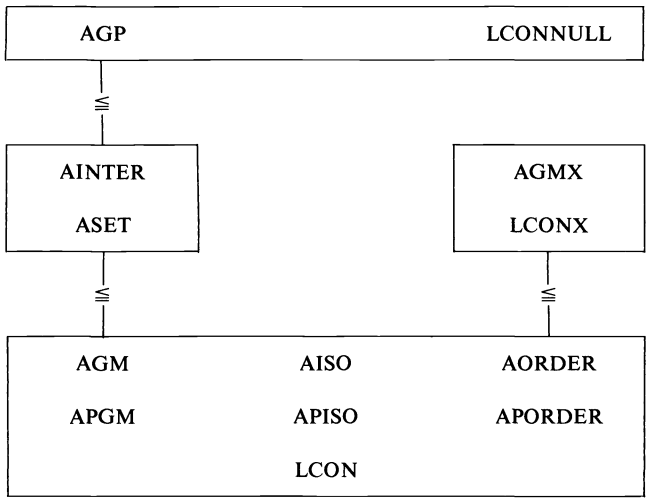


FIG. 7.1. *This diagram summarizes our results concerning relationships between the parallel complexites of Abelian permutation group problems. Problems within one box are NC^1-equivalent and problems appearing in a box below the symbol "≦" are NC^1-reducible to the problems appearing in the box immediately above "≦". (For a reduction involving LCONX and LCONNULL see Lemma 5.3.)*

introduces a lifting technique, for solving a system of homogeneous linear congruences mod p^l, which does not require an initial guess whose lifting is "guaranteed" at each lifting step (see for example [Ga84b]). More precisely, the goal of a lifting step is to compute *all* solutions mod p^{i+1}, given *all* solutions mod p^i. (The technique can be adapted to solve a nonhomogeneous system [McCo83]; then a lifting step may "fail," in which case *no* solution mod p^i lifts to a solution mod p^{i+1} and we conclude that the original system had no solution.)

LEMMA 8.1. *Suppose we are given $X_1, \cdots, X_r \in \mathbf{Z}^m$, $1 \leqq i < l$, spanning a submodule of $\mathbf{Z}^m$ whose image in $\mathbf{Z}_{p^l}^m$ equals*

$$\{X \in \mathbf{Z}_{p^l}^m \mid AX \equiv 0 \bmod p^i\}. \tag{1}$$

Then an NC^2 computation yields $Y_1, \cdots, Y_s \in \mathbf{Z}^m$, with $s \leqq 2r + m$, spanning a submodule of $\mathbf{Z}^m$ whose image in $\mathbf{Z}_{p^{i+1}}^m$ equals

$$\{Y \in \mathbf{Z}_{p^{i+1}}^m \mid AY \equiv 0 \bmod p^{i+1}\}. \tag{2}$$

Proof.

$$AY \equiv 0 \bmod p^{i+1} \Rightarrow AY \equiv 0 \bmod p^i$$

$$\Rightarrow$$

$$Y \equiv \sum_{j=1}^{r} u_j X_j \bmod p^i \quad \text{for some } u_1, \cdots, u_r \in \mathbf{Z}$$

$$\Rightarrow$$

$$Y = \sum_{j=1}^{r} u_j X_j + p^i \hat{Y} \tag{3}$$

for some $u_1, \cdots, u_r \in \mathbf{Z}$ and some $\hat{Y} \in \mathbf{Z}^m$. Writing $A^{(i)}$ for A reduced modulo p^i, let

$$A = A^{(i)} + p^i \hat{A}$$

for $\hat{A} \in \mathbf{Z}^{k \times m}$. Then

$$A\left(\sum_{j=1}^{r} u_j X_j + p^i \hat{Y}\right) \equiv 0 \bmod p^{i+1}$$

$$\Leftrightarrow$$

$$\sum_{j=1}^{r} (A^{(i)} X_j + p^i \hat{A} X_j) u_j + p^i A^{(i)} \hat{Y} \equiv 0 \bmod p^{i+1}$$

$$\Leftrightarrow$$

$$\sum_{j=1}^{r} (A^{(i)} X_j / p^i + \hat{A} X_j) u_j + A^{(i)} \hat{Y} \equiv 0 \bmod p \tag{4}$$

using that the image in $\mathbf{Z}_{p^i}^m$ of each X_j belongs to (1) to justify p^i as denominator. Now (4) is a system of k linear equations over the field $\mathbf{Z}_p$ in the $r+m$ unknowns $u_1, \cdots, u_r$ and the entries of $\hat{Y}$. In NC^2 we can therefore compute vectors $V_1, \cdots, V_t \in \mathbf{Z}^{r+m}$, $t \leqq r+m$, whose images modulo p form a basis for the nullspace of system (4) (by [BoGaHo82], [BoCoPi83] and [Mu86]; see comment prior to this lemma). Taking stock, we have therefore shown that $AY \equiv 0 \bmod p^{i+1}$ iff the integer vector Y is expressible as in (3) where

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_r \\ \hat{Y} \end{bmatrix} = v_1 V_1 + \cdots v_t V_t + p \begin{bmatrix} v_{t+1} \\ v_{t+2} \\ \vdots \\ v_s \\ \hat{V} \end{bmatrix} \tag{5}$$

for some $v_1, \cdots, v_s \in \mathbf{Z}$, $\hat{V} \in \mathbf{Z}^m$, and for $s = t + r \leqq 2r + m$. To compute a set of integer vectors spanning a submodule of $\mathbf{Z}^m$ whose image mod p^{i+1} equals the set of all solutions $Y \in \mathbf{Z}_{p^{i+1}}^m$ to $AY \equiv 0 \bmod p^{i+1}$, it therefore suffices to substitute (5) into (3), to reduce mod p^{i+1} and to collect as desired vectors the factors of each one of the "variables" $v_1, \cdots, v_s$ and the entries of $\hat{V}$. More precisely let V_{qj} denote the jth entry in V_q, $1 \leqq q \leqq t$, $1 \leqq j \leqq r$ and let $\hat{V}_q \in \mathbf{Z}^m$ denote the vector whose components are the last m entries of V_q, $1 \leqq q \leqq t$. Then (5) into (3) yields

$$Y \equiv \sum_{j=1}^{r} \left(\sum_{q=1}^{t} v_q V_{qj} + p v_{t+j}\right) X_j + p^i \left(\sum_{q=1}^{t} v_q \hat{V}_q + p\hat{V}\right) \bmod p^{i+1}$$

$$\equiv \sum_{j=1}^{s} v_j Y_j \bmod p^{i+1}$$

for $Y_1, \cdots, Y_s$ defined as

$$Y_q = \begin{cases} \sum_{j=1}^{r} V_{qj} X_j + p^i \hat{V}_q & \text{if } 1 \leqq q \leqq t, \\ p X_{q-t} & \text{if } t < q \leqq s. \end{cases}$$

As for the complexity of the above steps, computing $\hat{A}$ (and performing all divisions by p^i and by p) is in NC^1 by Proposition 3.1 since p^l is tiny. The Y_j's and the products $A_i X_j$ and $\hat{A} X_j$ are obtained in NC^1 also by parallel multiplications followed by iterated additions. □

COROLLARY 8.2. *Let $A \in \mathbf{Z}^{k\times m}$ and let p^l be a tiny prime power. NC^3 contains the problem of computing $X_1, \cdots, X_r \in \mathbf{Z}^m$, $r < 2n^2$ (with n the length of the problem encoding), spanning a submodule of $\mathbf{Z}^m$ whose image in $\mathbf{Z}_{p^l}^m$ equals*

$$\{X \in \mathbf{Z}_{p^l}^m | AX \equiv 0 \bmod p^l\}.$$

Proof. Using [BoGaHo82], [BoCoPi83] and [Mu86] (see comment prior to Lemma 8.1) we compute in NC^2 a basis $\{X_1, \cdots, X_r\}$, $r \leqq m$, for the nullspace of A over $\mathbf{Z}_p$. When viewed as integer vectors, $X_1, \cdots, X_r$ satisfy the hypothesis of Lemma 8.1 with $i = 1$. Then we apply the lemma $l-1$ times (updating r and the X_i's after each application). The value of r before the ith lifting step satisfies the recurrence

$$f(1) \leqq m, \qquad f(i) \leqq 2f(i-1) + m.$$

By induction one sees that $f(i) \leqq (2^{i+1} - 1)m$. Now $p^l \leqq n$ implies that $l \leqq \log n$. Hence the last iteration above computes $r \leqq 2^{l+1}m < 2n^2$ vectors satisfying the statement of the corollary, and the complete procedure is an NC^3 computation. □

THEOREM 8.3. LCONNULL $\in NC^3$.

Proof. Let $A \in \mathbf{Z}^{k\times m}$ with $q = p_1^{l_1} \cdots p_s^{l_s}$ an integer with tiny prime factors. Using parallel oracle gates for the problem of Corollary 8.2 we compute for each i, $1 \leqq i \leqq s$, a set of vectors

$$M_i = \{X_1^{(i)}, \cdots, X_{r_i}^{(i)}\}, \qquad r_i < 2n^2,$$

spanning a submodule of $\mathbf{Z}^m$ whose image in $\mathbf{Z}_{p_i^{l_i}}^m$ equals the set of all solutions to $AX \equiv 0 \bmod p_i^{l_i}$. Let $\alpha_i = \text{rest}_{p_i}(q)$ and let β_i be such that $\alpha_i\beta_i \equiv 1 \bmod p_i^{l_i}$, $1 \leqq i \leqq s$. Write $I_j \in \mathbf{Z}^m$ for the vector having q as its jth entry and 0 everywhere else. We claim that

$$M = \{\alpha_i X_{j_i}^{(i)} | 1 \leqq i \leqq s,\ 1 \leqq j_i \leqq r_i\} \cup \{I_1, \cdots, I_m\}$$

generates the $\mathbf{Z}$-module of all solutions $X \in \mathbf{Z}^m$ to $AX \equiv 0 \bmod q$. Any $X \in M$ obviously satisfies $AX \equiv 0 \bmod q$. Conversely, suppose $AX \equiv 0 \bmod q$. Then $AX \equiv 0 \bmod p_i^{l_i}$, so $X \equiv Y^{(i)} \bmod p_i^{l_i}$, where $Y^{(i)}$ is in $\langle M_i \rangle$. Hence $X \equiv Y \bmod q$, where $Y = \sum_i \alpha_i\beta_i Y^{(i)}$ is in $\langle M \rangle$. Therefore $X - Y$ is in $\langle M \rangle$ and so is X.

Observe that the number of vectors in M is $< m + 2sn^2 = O(n^3)$. Now we can obtain each α_i in NC^2, $1 \leqq i \leqq s$, as the iterated product of its prime power factors (q is presented in factored form on input). Then NC^1 suffices to compute the β_i's (Proposition 3.4) and all vectors. □

THEOREM 8.4. *The following problems belong to NC^3*: AGM, AGMX, AISO, AORDER, ASET, AINTER, AGP, LCON, LCONX *and* LCONNULL.

Proof. Use Theorem 6.8, Proposition 6.12, Proposition 6.13, Theorem 7.10, Lemma 5.3, Theorem 8.3 and Proposition 2.6. Note that to conclude $\{\text{LCONX, AGMX}\} \subseteq NC^3$ we need to observe that the problem of computing the product of any number of integers in binary belongs to $NC^2 \subseteq NC^3$. □

Recall the Elementary Abelian Group Membership problem (see the discussion following Theorem 5.6). We observe that EAGM $\in NC^2$. To see this, note that instances of EAGM reduce to instances of LCONNULL (via LCON; see Lemmas 7.8 and 5.3) in which the modulus q is in fact a single prime p. Now LCONNULL restricted to the case of a prime modulus q is solvable in NC^2, since no lifting is required (see Corollary 8.2).

Let us write AISO_c, AORDER_c and AGM_c respectively for AISO, AORDER and AGM when the number of permutations generating the input groups is viewed as

constant. Also, let us denote LCON with bounded number of variables as LCON_c. We mention (see [McCo85] for details) the following.

PROPOSITION 8.5. $\{\mathrm{AGM}_c, \mathrm{AORDER}_c, \mathrm{AISO}_c, \mathrm{LCON}_c\} \subseteq NC^1$.

We now turn to problem FACTOR, which consists of expressing an Abelian group as a product of cyclic factors. We first reduce the problem to the case in which the group is an Abelian P-group.

PROPOSITION 8.6. $\mathrm{FACTOR} \leqq \{\mathrm{PFACTOR}, \mathrm{DGAP1}\} \leqq \mathrm{PFACTOR}$.

Proof. The last reduction holds because

$$\mathrm{DGAP1} \leqq \mathrm{AGM} \leqq \mathrm{APISO}$$

by Theorems 5.6 and 6.8, and it is not hard to see that $\mathrm{APISO} \leqq \mathrm{PFACTOR}$.

As to the first reduction, the reader can verify that the following circuit solves FACTOR. First we compute all the Sylow subgoups of G in NC^1, appealing to Corollary 3.9. Then, in parallel for each prime p, we use a PFACTOR oracle gate to compute the cyclic decomposition of each Sylow p-subgroup. Finally, for $1 \leqq i \leqq s$ in parallel (where s is the number of cyclic factors in the Sylow subgroup having the largest number of factors), we use DGAP1 oracles in order to output as h_i the product of all permutations generating the ith cyclic group in the decomposition of some Sylow p-subgroup. □

In [Mo84] Morenz develops a sequential PFACTOR algorithm. Our parallel solution to PFACTOR uses some of the ideas found there.

THEOREM 8.7. $\mathrm{FACTOR} \in NC^4$.

Proof. Using the previous proposition it suffices to show that $\mathrm{PFACTOR} \in NC^4$. The inputs to PFACTOR are $g_1, \cdots, g_r$ generating a group G. Let G be a p-group, and let $(\delta_1, \cdots, \delta_k)$ be its signature (see Fact 2.5). The computation proceeds in k stages. (Observe that k is the integer such that p^k is the largest of the orders of the generators of G.) Define H_{k+1} as the trivial group (that is, the group with a single element) and assume inductively that we have computed the cyclic decomposition

$$\langle h_1 \rangle \times \cdots \times \langle h_t \rangle$$

of a group $H_{i+1} \leqq G$ having signature

$$(0, \cdots, 0, \delta_{i+1}, \cdots, \delta_k).$$

We will describe a stage computing the cyclic decomposition of a subgroup H_i, $H_{i+1} \leqq H_i \leqq G$, having signature

$$(0, \cdots, 0, \delta_i, \delta_{i+1}, \cdots, \delta_k).$$

Observe that

(i) $G/H_{i+1} = \langle g_1 H_{i+1}, \cdots, g_r H_{i+1} \rangle$;

(ii) The signature of G/H_{i+1} is $(\delta_1, \cdots, \delta_i)$.

We claim that it suffices, in parallel for each j satisfying

$$(\dagger) \qquad g_j^{p^{i-1}} \notin \langle \{g_1^{p^{i-1}}, \cdots, g_{j-1}^{p^{i-1}}\} \cup H_{i+1} \rangle, \qquad 1 \leqq j \leqq r,$$

to compute integers $a_1, \cdots, a_t$ satisfying

$$(\dagger\dagger) \qquad g_j^{p^i} = \prod_{m=1}^{t} (h_m^{p^i})^{a_m}$$

and to include

$$(\dagger\dagger\dagger) \qquad g_j \prod_{m=1}^{t} h_m^{-a_m}$$

as a new cyclic generator of H_i. Internally the new generators are to be labelled $h_{t+1}, h_{t+2}, \cdots$ (in the order in which they would be discovered as j ran from 1 to r) and added to those of H_{i+1} to form the cyclic decomposition of H_i. This completes the stage.

Fix a j for which (†) holds and observe the following. The integers $a_1, \cdots, a_t$ referred to in (††) exist, since g_j is expressible as a product of an element of H_{i+1} and an element of G of order p^i or less. Then say the permutation in (†††) is internally labelled h_{t+l}, $t+l \leqq r$. The order of h_{t+l} is at most p^i by (††), at least p^i by (†), and h_{t+l} together with $h_1, \cdots, h_t$ generate g_j. Now an induction shows that $g_m^{p^{i-1}} \in H_i$, $1 \leqq m \leqq j$, and that $\langle h_{t+l} \rangle$ has only the identity element in common with

$$\langle h_1, \cdots, h_t, h_{t+1}, \cdots, h_{t+l-1} \rangle.$$

We note that the divisibility requirements on the orders of the cyclic generators of H_i are satisfied provided these were satisfied by the cyclic generators of H_{i+1}. This proves our claim.

To conclude the proof we note that the bottleneck at each stage consists of solving instances of AGM (†) and of AGMX (††). Hence each stage is an NC^3 computation by Theorem 8.4, and our NC^4 upper bound results from having $k = O(\log n)$ stages. □

9. Conclusion. In this paper we have examined a number of Abelian permutation group problems from the viewpoint of their parallel complexity. Our goal was always twofold: first, to exhibit NC^1 reductions among problems and second, whenever possible, to obtain upper bounds on the parallel complexities of the problems under consideration. The importance of NC^1 reductions lies in the fact that they allow comparing the (unknown) parallel complexities of problems and to establish a classification of these into classes of problems essentially equivalent with respect to their parallel complexities.

Refer to Fig. 7.1 for a summary of our results concerning the relative parallel complexities of the main problems discussed in this paper. Whether NC^1 reductions can be exhibited which would allow merging some of the NC^1-equivalence classes appearing in that figure is an interesting open question. In particular, does any of AGP ≦ AINTER ≦ AGM or of AGP ≦ AGMX ≦ AGM hold? Are ⟨AINTER⟩ and ⟨AGMX⟩ comparable via NC^1 reducibility? Does LCONX reduce to LCONNULL without the need for an iterated integer product oracle gate (Lemma 5.3)?

A natural extension to our work is the investigation of larger classes of groups. In [Mc84] P-groups and nilpotent groups are examined. Results obtained there are an NC^4 nilpotency test and several NC^1 reductions involving nilpotent group problems. In [LuMc85] a number of problems involving nilpotent groups and more generally solvable groups (e.g. membership, order, normal closures, composition series, centers, point stabilizers in the nilpotent case) are solved in NC. Nonetheless, several questions involving problems apparently harder than membership (e.g. isomorphism, intersection and setwise stabilizer) remain unanswered even in the case of P-groups. For even larger classes of groups the relevant questions remain out of reach.

Turning to the question which essentially motivated the work described here, we would like to know whether the general permutation group membership problem GM is complete for the class P, with respect to log space or to some form of NC^k reducibilty. (Luks in [Lu86] remarks on the bearing of the classification of finite simple groups on GM, suggesting evidence that the problem is in fact not complete for P; see [Lu86] also for the most general class of permutation groups for which an NC membership test is yet available.) In attempting to show this, the algebraic structure of a group makes it difficult to determine the right properties to exploit in reducing a complete problem (see [HoRu84] for an extensive list) to GM. Since it is unlikely that $P \subseteq NC$,

the results presented here provide evidence that a reduction showing GM complete for P would involve non-Abelian groups (indeed nonsolvable groups by [LuMc85]).

We conclude by considering the problem AGP consisting of computing a generator-relator presentation of an Abelian permutation group. We investigated this problem only because the linear algebra problem LCONNULL, which holds the key to the NC^3 upper bounds to all our Abelian group problems, did not appear to generalize to the case of non-Abelian groups. AGP provided the group-theoretic counterpart to LCONNULL, and since non-Abelian groups have a generator-relator presentation, AGP generalizes (say to GP) to more general classes of groups (see [Mc84] for a precise definition of GP). We anticipate that this generalization may bear fruit because we can now relate our permutation group problems to problems which combinatorists have been investigating for many years. Indeed, the generalization of LCONNULL which we were seeking, in the case of nilpotent groups, has already been studied: in [Th83] languages associated with nilpotent groups of bounded nilpotency class are characterized in term of subword counts. This newly discovered relationship opens an interesting avenue for future research.

Acknowledgments. We are indebted to Rob Morenz for Lemma 3.8, to Mike Luby for a crucial idea in the proof of Proposition 3.21, to Joachim von zur Gathen for valuable comments on an earlier version of Lemma 8.1, and to Eugene Luks and Charles Rackoff for other helpful comments.

REFERENCES

[A1B183] H. ALT AND N. BLUM, *On the Boolean circuit depth of division related functions*, Dept. of Computer Science, Pennsylvania State Univ., 1983.

[Ar44] E. ARTIN, *Galois Theory*, Notre Dame Mathematical Lectures # 2, Univ. of Notre Dame Press, second ed., 1944 (sixth printing, 1971).

[Bo77] A. BORODIN, *On relating time and space to size and depth*, this Journal, 6 (1977), pp. 733–744.

[BoCoPi83] A. BORODIN, S. A. COOK AND N. PIPPENGER, *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Inform. and Control, 58 (1983), pp. 113–136.

[BoGaHo82] A. BORODIN, J. VON ZUR GATHEN AND J. HOPCROFT, *Fast parallel matrix and* GCD *computations*, Inform. and Control, 52 (1982), pp. 241–256.

[Co81] S. A. COOK, *Towards a complexity theory of synchronous parallel computation*, Enseign. Math. (2), 27 (1981), pp. 99–124.

[Co85] ———, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.

[CoMc86] S. A. COOK AND P. MCKENZIE, *Problems complete for deterministic logarithmic space*, Publication 560, D.I.R.O., Univ. of Montreal, Canada, J. Algorithms, to appear.

[Cs76] L. CSANKY, *Fast parallel matrix inversion algorithms*, this Journal, 5 (1976), pp. 618–623.

[DoLiRe79] D. DOBKIN, R. J. LIPTON AND S. REISS, *Linear Programming is log-space hard for P*, Inform. Process. Lett., 8 (1979), pp. 96–97.

[FiPi74] M. FISHER AND N. PIPPENGER, *M.J. Fisher Lecture Notes on Network Complexity*, Univ. of Frankfurt, West Germany, 1974.

[FuHoLu80] M. FURST, J. HOPCROFT AND E. LUKS, *Polynomial time algorithms for permutation groups*, Proc. 21st IEEE Foundations of Computer Science, 1980, pp. 36–41.

[Ga84a] J. VON ZUR GATHEN, *Lecture Notes for* CSC2408S (*Parallel Arithmetic Complexity*), Dept. of Computer Science, Univ. of Toronto, Ontario, Canada, 1984.

[Ga84b] ———, *Hensel and Newton methods in valuation rings*, Math. Comp., 42 (1984), pp. 637–661.

[GaJo79] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[GoShSt81] L. M. GOLDSCHLAGER, R. A. SHAW AND J. STAPLES, *The maximum flow problem is log space complete for P*, Theoret. Comput. Sci., 21 (1982), pp. 105–111.

[Ha59] M. HALL, *Theory of Goups*, Macmillan, New York, 1959.

[Ho82] C. M. HOFFMAN, *Group-Theoretic Algorithms and Graph Isomorphism*, Lecture Notes in Computer Science, 136, Springer-Verlag, Berlin-New York, 1982.

[HoRu84] H. J. HOOVER AND W. L. RUZZO, *A compendium of problems complete for P*, Preliminary draft, March 1984.

[HoU179] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[Je82] M. JERRUM, *A compact representation for permutation groups*, Proc. 23rd IEEE Foundations of Computer Science, 1982, pp. 126-133.

[Jo75] N. D. JONES, *Space-bounded reducibility among combinatorial problems*, J. Comput. System Sci., 11 (1975), pp. 68-85.

[La75] R. E. LADNER, *The circuit value problem is log space complete for P*, SIGACT News 7, 1975, pp. 18-20.

[Lu86] E. M. LUKS, *Parallel algorithms for permutation groups and graph isomorphism*, Proc. 27th IEEE Foundations of Computer Science, 1986, pp. 292-302.

[LuMc85] E. M. LUKS AND P. MCKENZIE, *Fast parallel computation with permutation groups*, Proc. 26th IEEE Foundations of Computer Science, 1985, pp. 505-514.

[Mc84] MCKENZIE, *Parallel complexity and permutation groups*, Ph.D. thesis, Technical Report # 173/84, Dept. of Computer Science, Univ. of Toronto, Toronto, Ontario, Canada, 1984.

[McCo83] P. MCKENZIE AND S. A. COOK, *The parallel complexity of the Abelian permutation group membership problem*, Proc. 24th IEEE Foundations of Computer Science, 1983, pp. 154-161.

[McCo85] ———, *The parallel complexity of Abelian permutation group problems*, Technical Report # 181/85, Dept. of Computer Science, Univ. of Toronto, Toronto, Ontario, Canada, April 1985.

[Mo84] R. G. MORENZ, *Isomorphism of Abelian permutation groups and related problems*, M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto, Toronto, Ontario, Canada, 1984.

[Mu86] K. MULMULEY, *A fast parallel algorithm to compute the rank of a matrix over an arbitrary field*, Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 338-339.

[MuPr75] D. E. MULLER AND F. P. PREPARATA, *Bounds to complexities of networks for sorting and for searching*, J. Assoc. Comput. Mach. (1975), pp. 195-201.

[NiZu80] I. NIVEN AND H. S. ZUCKERMAN, *An Introduction to the Theory of Numbers*, 4th ed., John Wiley, New York, 1980.

[Pi79] N. PIPPENGER, *On simultaneous resource bounds*, Proc. 20th IEEE Foundations of Computer Science, 1979, pp. 307-311.

[Re83] J. REIF, *Parallel algorithms for graph isomorphism*, Report TR-14-83, Aiken Computation Laboratory, Harvard Univ., Cambridge, MA, May 1983.

[Ru80] W. L. RUZZO, *Tree-size bounded alternation*, J. Comput. System Sci., 21 (1980), pp. 218-235.

[Sa76] J. E. SAVAGE, *The Complexity of Computing*, John Wiley, New York, 1976.

[SaJa81] C. SAVAGE AND J. JA'JA', *Fast, efficient parallel algorithms for some graph problems*, this Journal, 10 (1981), pp. 682-691.

[Sc76] C. P. SCHNORR, *The network complexity and the Turing machine complexity of finite functions*, Acta Inform., 7 (1976) pp. 95-107.

[Th83] D. THÉRIEN, *Subword counting and nilpotent groups*, in Combinatorics on Words: Progress and Perspectives, L. J. Cummings, ed., 1983, pp. 297-305.

[ToCo37] J. A. TODD AND H. S. M. COXETER, *A practical method for enumerating cosets of a finite abstract group*, Proc. Edinburgh Math. Soc . (2), 5, 1937, pp. 26-34.

[Wi64] H. WIELANDT, *Finite Permutation Groups*, Academic Press, New York, 1964.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987

© 1987 Society for Industrial and Applied Mathematics
008

NONLINEAR ALGEBRA AND OPTIMIZATION ON RINGS ARE "HARD"*

H. B. HUNT III† AND R. E. STEARNS‡

Abstract. Several general **NP**- or **coNP**-hardness results are presented for nonlinear algebraic and optimization problems on rings. These results include the following:

(1) The problem of determining if a system of nonlinear equations on a ring has a solution is **NP**-hard for virtually all of the rings studied in mathematics or computer science.

(2) Let S be any nondegenerate ring with a multiplicative identity, and let $\leqq$ be any linear order on S. Then, the problems of maximizing or of minimizing a linear function on S subject to quadratic constraints are both **NP**- and **coNP**-hard.

(3) Let S be any ordered ring, and let $\leqq$ be the associated order. Then, the problems of maximizing or of minimizing a quadratic multiple variable polynomial on S subject to constraints of the form $a \leqq x \leqq b$ are both **NP**- and **coNP**-hard.

Several additional related nonlinear problems for such rings are shown to be **NP**-, **coNP**-, or **#P**-hard. In particular, the following variant of Hilbert's Tenth Problem is shown to be **NP**-complete:

(4) The problem of determining, for a quadratic multiple variable polynomial f with integer coefficients over the reals, if there is an assignment **v** of values from $\{0, 1\}$ to the variables of f such that f takes on the value 0 under **v**.

Key words. nonlinear algebra, combinatorics, rings, computational complexity **NP**-hard, **coNP**-hard **#P**-hard

AMS(MOS) subject classifications. 68Q25, 13A99, 16A99

1. Introduction. By a **ring S** $= (S, +, -, \cdot, 0)$, we mean an Abelian group $(S, +, -, 0)$ augmented with a (not necessarily commutative) binary operation $\cdot$ on S such that the operation $\cdot$ is associative and distributes on both sides over the operation $+$. Unlike [L], [MB], we do not assume that every ring has a multiplicative identity. We do assume, however, that $\|S\| \geqq 2$, i.e., that **S** is nondegenerate. In this paper we study the computational complexities of several nonlinear algebraic and optimization problems on rings. Several general **NP**- or **coNP**-hardness results are obtained. These general results include the following:

(1) The problem of determining if a system of quadratic equations on a ring has a solution is **NP**-hard for each ring S such that either

(1a) S has a nonzero element x for which $x = x^2$, or

(1b) S has no nonzero zero divisors.

(2) Let S be any ring with a multiplicative identity, and let $\leqq$ be any linear order on S. Then, the problems of maximizing or of minimizing a linear function on S subject to quadratic constraints are both **NP**- and **coNP**-hard.

(3) Let S be any ordered ring [MB], and let $\leqq$ be the associated order. Then, the problems of maximizing or of minimizing a quadratic multiple variable polynomial on S subject to constraints of the form $a \leqq x \leqq b$ are both **NP**- and **coNP**-hard. In particular, result 1 holds for the ring **Z** of integers, the ring **Q** of rationals, the ring **R** of reals, the ring **C** of complex numbers, the ring **Qu** of quaterions, each ring with a multiplicative identity, each field, each division ring, each integral domain, each Boolean ring, and each ring $\mathbf{Z}_k$ of integers modulo k [Ab], [L], [MB], [van]. To the best of our knowledge, this list includes most of the rings studied in mathematics or computer science.

* Received by the editors May 27, 1986; accepted for publication (in revised form) December 16, 1986.

† Computer Science Department, State University of New York, Albany, New York 12222. The research of this author was supported in part by National Science Foundation grant DCR84-03014.

‡ Computer Science Department, State University of New York, Albany, New York 12222. The research of this author was supported in part by National Science Foundation grant DCR83-03932.

Several additional nonlinear problems for such rings are also shown to be **NP**-, **coNP**-, or #**P**-hard. One particular result of special interest is the following:

(4) The problem of determining if a quadratic multiple variable polynomial with integer coefficients over the reals has {0, 1}-valued roots is **NP**-complete.

The problems studied for rings include the following:

(A) The *Simultaneous Quadratic Equation Satisfiability Problem* denoted by *SQES Problem*, i.e., the problem of determining if a finite system of quadratic equations on **S** has a solution;

(B) Letting **T** be a subset of *S*, the *Simultaneous Quadratic Equation* **T**-*Satisfiability Problem* denoted by *SQE* **T**-*Satisfiability Problem*, i.e., the problem of determining if a finite system of quadratic equations on **S** has a solution such that each variable takes on a value from **T**;

(C) The *Implicational Identity Problem*: denoted by II *Problem*, i.e., the problem of determining, for formulas f and g on S, if f and g are equivalent on S subject to the algebraically expressed constraints

$$f_1 = g_1, \cdots \quad \text{and} \quad f_k = g_k (k \geqq 1)$$

on the values of their variables;

(D) Letting S be any ring with a multiplicative identity, and $\leqq$ be any linear order on S, the problems of computing the maximum or the minimum values of a linear function on S subject to quadratic constraints;

and letting S be any ordered ring, **T** be a subset of S, f be a multiple variable polynomial on S, and $\leqq$ be the associated order on S;

(E) Computing the maximum or the minimum values taken on by f for assignments **v** of values from **T** to the variables of f, and

(F) Determining if there is an assignment **v** of values from **T** to the variables of f such that f takes the value 0 under **v**.

An outline of the results obtained for those problems, together with where these results appear in this paper, is shown in Table 1. Additional results on the complexities of several related enumeration problems appear in § 6.

TABLE 1
Overview of results obtained.

Problem	Complexity bound	Obtained in
(1) SQES-problem	**NP**-hard	§ 3, Theorems 3.1 and 3.2
(2) SQE **T**-satisfiability problem	**NP**-hard	§ 4, Theorems 4.1 and 4.2
(3) II-problem	**coNP**-hard	§ 4, Theorem 4.3
(4) Problem D	Both **NP**- and **coNP**-hard	§ 5, Theorem 5.1
(5) Problems E and F	Both **NP**- and **coNP**-hard	§ 5, Theorem 5.2 and its corollaries

Previous work on the computational complexities of nonlinear problems for rings includes the following:

(1) [Ma], [D] There is no algorithm to determine if a quartic multiple variable polynomial with integer coefficients over **R** has integer-valued roots.

(2) [J] There is no algorithm for integer programming with quadratic constraints.

(3) [D], [J] There is no algorithm to determine if a system of quadratic equations with integer coefficients over **R** has an integer-valued root. Hence, the SQES-Problem for **Z** is undecidable.

(4) [FY], [GJ] The *SQES-Problem* for the finite field Z_2 is **NP**-complete. The *SQES-Problem* for the algebraic closure of $\mathbf{Z}_2$ is **NP**-hard.

(5) [Sa] The problems of maximizing or of minimizing a linear function on **Q** subject to quadratic constraints are both **NP**- and **coNP**-hard.

(6) [Sa] The problems of maximizing or of minimizing a quadratic multiple variable polynomial on **Q** subject to linear inequality constraints are both **NP**- and **coNP**-hard.

(7) [AM] The problem of determining if a sixth-degree multiple variable polynomial with integer coefficients over **R** has {0, 1}-valued roots is **NP**-complete.

(8) [Sz] There is a random polynomially time-bounded algorithm to determine if two formulas on **Z** are not equivalent.

(9) (1) [IM], [BHR] Let **F** be any finite field. Then, the problem of determining if two formulas on **F** are equivalent is **coNP**-complete. (2) [BHR] Let **F** be any ring Z_k ($k \geqq 2$). Then, the problem of determining if two formulas on **F** are equivalent is **coNP**-complete.

The results presented here significantly extend, simplify, unify and delimit the above known results, especially results (4)–(8), as follows.

(i) In Theorems 3.1 and 3.2, we show that **NP**-hard lower bounds on the complexities of *SQES-Problems* hold for the vast majority of the rings used in mathematics and computer science, not just for certain fields of characteristic two (result (4)). In particular, the SQES-Problem is **NP**-hard for each ring with a multiplicative identity.

(ii) Theorem 4.3 shows that the II Problem is **coNP**-hard for any ring satisfying conditions (1a) or (1b) above. Thus unless **RP** = **NP**, random polynomially time-bounded algorithms do not exist for these II problems. Hence, no analogue of Schwartz's **RP** algorithm for Formula Equivalence Problems (result (8)) is likely to exist for the II Problem for any of these rings (including most rings studied in mathematics and computer science). In Proposition 4.4, we observe that the II Problem for the ring **Z** (and certain other rings) is undecidable. Thus the **RP** algorithm does not extend to the II Problem for **Z** even if **RP** = **NP**.

(iii) Theorem 5.1 shows that Sahni's lower bounds (result (5)) for **Q** on the complexities of maximizing or of minimizing linear functions subject to quadratic constraints hold for any ring with a multiplicative identity.

(iv) Theorem 5.2 and its corollaries show that the problem of determining whether a multiple variable polynomial f with integer coefficients over the ring **R** has {0, 1}-valued roots is **NP**-complete, even when f is of second degree. Our proof of this result depends neither on the fact that 0 and 1 are integers nor on the fact that the set {0, 1} is finite. All that is required on a set **T** of possible values of a second degree multiple variable polynomial f for problem F to be **NP**-hard is that

$$|T| \geqq 2$$

and

$$\text{both } a = \text{glb}(\mathbf{T}) \text{ and } b = \text{lub}(\mathbf{T}) \text{ are elements of } \mathbf{T}.$$

Thus, Theorem 5.2 and its corollaries show that Sahni's result (6) above, and Adleman and Manders' result (7) above, hold for essentially the same reason.

The remainder of this section consists of definitions and notation from complexity theory [GJ] and from algebra [Ab], [MB] needed to read this paper. We assume that the reader is familiar with the concepts of *polynomial* (*time*) *reducibility* and *polynomial*

(*time*) *Turing reducibility* and with the complexity classes **P**, **NP**, and **coNP**; otherwise see [GJ].

DEFINITION 1.1. Let L be a language over a finite alphabet $\sum$. We say that L is **NP**-*hard* if each element of **NP** is polynomially reducible to it. If in addition, L is recognizable by a nondeterministic polynomially time-bounded Turing machine, we say that L is **NP**-*complete.* We say that L is **coNP**-*hard* if each element of **coNP** is polynomially reducible to it. If in addition, $\sum^* - L$ is recognizable by a nondeterministic polynomially time-bounded Turing machine, we say that L is **coNP**-*complete.*

When discussing the complexity of enumeration problems in § 6, we use the following definitions and notation from [GJ], [Va].

DEFINITION 1.2. A *search problem* Π consists of a set D_Π of objects called *instances* and, for each instance $I \in D_\Pi$, a set $S_\Pi[I]$ of objects called *solutions* for I. An algorithm is said to *solve* a search problem Π if, given $I \in D_\Pi$ as input, the algorithm outputs "no," if $S_\Pi[I] = \varnothing$ and outputs $s \in S_\Pi[I]$, otherwise.

The *enumeration problem* associated with a search problem Π is the problem of determining, given $I \in D_\Pi$, the cardinality of $S_\Pi[I]$.

DEFINITION 1.3. The class **#P** consists of all enumeration problems associated with search problems Π such that

there is a nondeterministic algorithm for Π such that, for all $I \in D_\Pi$, the number of distinct guesses of values s in $S_\Pi[I]$ that leads to acceptance of I by the algorithm equals $|S_\Pi[I]|$, and

the length of the longest accepting computation of the algorithm on $I \in D_\Pi$ is bounded by a polynomial in the length of I.

An enumeration problem is said to be **#P**-hard if each element of **#P** is polynomially Turing reducible to it. If in addition, the enumeration problem is an element of **#P**, the problem is said to be **#P**-*complete.*

DEFINITION 1.4. Let Π and Π' be search problems. A *parsimonious polynomial* (*time*) *reduction* [Si] of Π and Π' is a function $f: D_\Pi \to D_{\Pi'}$, computable deterministically in polynomial time such that, for all $I \in D_\Pi$,

$$|S_\Pi[I]| = |S_{\Pi'}[f(I)]|.$$

It is generally assumed that each **NP**-, **coNP**-, or **#P**-hard problem is computationally intractable. It is also generally assumed that **NP** $\neq$ **coNP** and, thus, that no **coNP**-hard set is in **NP**.

DEFINITION 1.5. By an *algebraic structure* S, we mean a nonempty set S called the *domain* of S, together with a collection of operations on S of various arities. A structure S is said to be *nondegenerate* if $|S| \geqq 2$.

The algebraic structures considered in this paper consist of various kinds of rings, distributive lattices, and Boolean algebras. Henceforth, we assume that all rings, lattices, and Boolean algebras are nondegenerate.

DEFINITION 1.6. Let $S = (S, +, -, \cdot, 0)$ be a ring. S is said to be a *Boolean ring* if, for all $x \in S$, $x = x^2$. S is said to be a *unitary ring* if there exists $1 \in S$ such that, for all $x \in S$, $1 \cdot x = x \cdot 1 = x$. (The element 1 is called the *multiplicative identity* of S.) S is said to be an *ordered ring* when there exists a nonempty set $P \subset S$, called the set of *positive elements* of S, such that

(i) $a \in P$ and $b \in P$ implies $a + b \in P$ and $a \cdot b \in P$;

(ii) for all $a \in S$, exactly one of the following three alternatives holds:

$$a \in P, \quad a = 0, \quad -a \in P.$$

Let S be an ordered ring with set of positive elements P. Let $\leqq$ be the partial order on S defined by

$$a \leqq b \quad \text{if and only if } a = b \text{ or } (b - a) \in P.$$

We call $\leqq$ the *associated linear order* of S.

For some authors (e.g. [L], [MB]), a ring by definition has a multiplicative identity. Under this definition of ring, each ring satisfies condition (1a) above because $1 = 1^2$. We also note that, for each ordered ring S, the associated linear order of S is, in fact, a linear order [MB].

DEFINITION 1.7. Let $S = (S, +, -, \cdot, 0)$ be a ring. A *nonzero zero divisor* of S is a nonzero element x of S, for which there exists a nonzero element y of S, such that either $x \cdot y = 0$ or $y \cdot x = 0$.

DEFINITION 1.8. Let $S = (S, +, -, \cdot, 0)$ be a ring. Let f be a multiple variable polynomial on S; and let C be a finite system of equations on S. Let $\mathbf{T} \in S$.

(1) An assignment $\mathbf{v}$ of values from $\mathbf{T}$ to the variables of f such that f takes on the value 0 under $\mathbf{v}$ is said to be a $\mathbf{T}$-*valued root* of f.

(2) An assignment $\mathbf{v}$ of values from $\mathbf{T}$ to the variables of C such that the equations of C are simultaneously satisfied under $\mathbf{v}$ is said to be a $\mathbf{T}$-*valued solution* of C.

DEFINITION 1.9. A *lattice* $L = (S, \vee, \wedge)$ is an algebraic structure where the operations $\vee$ and $\wedge$ are binary operations on S that are commutative, associative, and idempotent, and such that for all $x, y \in S$,

$$x \vee (x \wedge y) = x \wedge (x \vee y) = x.$$

If, in addition, for all $x, y, z \in S$

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \quad \text{and} \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z),$$

the lattice L is said to be *distributive*.

We denote the two-element distributive lattice by **BIN** and the two-element Boolean algebra by **BOOLE**.

It is well known that lattices are closely related to *partially-ordered sets* as follows. If L is a lattice with domain S, then $(S, \leqq)$ is a partially-ordered set, where the binary relation $\leqq$ on S is defined by

$$a \leqq b \quad \text{if and only if } a \vee b = b$$

or equivalently, is defined by

$$a \leqq b \quad \text{if and only if } a \wedge b = a.$$

We also assume that the reader is familiar with *Boolean formulas*; otherwise see [GJ].

DEFINITION 1.10. A Boolean formula is said to be *monotone* if it has no occurrences of the operator **not**. A Boolean formula F of the form

$$F_1 \textbf{ and } \cdots \textbf{ and } F_m (m \geqq 1)$$

where each F_i is of the form F_{i1} **or** $\cdots$ **or** F_{ik} and each F_{ij} is a variable, is said to be an **and, or**-*formula*. **Or, and**-*formulas* are defined dually.

A *CNF formula* is a Boolean formula F of the form

$$F_1 \textbf{ and } \cdots \textbf{ and } F_m (m \geqq 1)$$

where each F_i is of the form F_{i1} **or** $\cdots$ **or** F_{ik} and each F_{ij} is a variable or a negated

variable. A CNF formula with no occurrences of **not** is said to be a *monotone CNF formula.* A CNF formula, where each F_i is a disjunct of at most two literals, is said to be a 2 *CNF formula.* A CNF formula, where each F_i is a disjunct of at most three literals, is said to be a 3 *CNF formula.*

DEFINITION 1.11. Let S be an algebraic structure with domain **S**. Let $\mathbf{T} \subset \mathbf{S}$. Let F be a formula on S. Let $\mathbf{v}$ be an assignment of values from **T** to the variables of F. We denote the value taken on by F under $\mathbf{v}$ by $\mathbf{v}[F]$.

2. Complexity theoretic and algebraic preliminaries. Several propositions needed in this paper are presented. Propositions 2.1–2.3 deal with the complexities of decision problems for Boolean formulas, monotone Boolean formulas, and systems of simultaneous equations on the structures **BIN** and **BOOLE**. Propositions 2.4 and 2.5 present elementary properties of distributive lattices and of powers $x^n (n \geqq 1)$ of elements of a ring. Proposition 2.6 shows how distributive lattices can be embedded into any ring S such that

$$\exists x \neq 0 \text{ in } S \text{ for which } x = x^2.$$

Proposition 2.7 shows that the only Boolean ring with no nonzero divisors is the finite field $\mathbf{Z}_2$.

PROPOSITION 2.1. (1) [Co] *The sets of satisfiable* 3 *CNF Boolean formulas and of satisfiable Boolean formulas are* **NP**-*complete.*

(2) [Va] *The problem of computing, given a monotone* 2 *CNF Boolean formula F, the number of assignments* $\mathbf{v}$ *of values from* $\{0, 1\}$ *to the variables of F such that* $\mathbf{v}[F] = 1$ *is* $\#\mathbf{P}$-*complete.*

(3) [Sc] *The set of monotone* 3 *CNF Boolean formulas f such that there is an assignment* $\mathbf{v}$ *of values from* $\{0, 1\}$ *to the variables of f such that, for all clauses* $\mathbf{c}$ *of f, exactly one variable appearing in* $\mathbf{c}$ *takes on the value* 1 *under* $\mathbf{v}$ *is* **NP**-*complete.*

(4) [BHR] *The set of monotone Boolean formulas F and G such that* $F \leqq G$ *is* **coNP**-*complete.*

(5) [HS1] *The set of monotone Boolean formulas F and G such that*

(i) *F is an* **and, or**-*formula,*

(ii) *G is an* **or, and**-*formula, and*

(iii) $F \leqq G$

is **coNP**-*complete.*

PROPOSITION 2.2. *The set of uniquely satisfiable Boolean formulas is* **coNP**-*hard.*

Proof. Let $F(x_1, \cdots, x_n)$ be a Boolean formula. Then, the Boolean formula $(F(x_1, \cdots, x_n)$ **and not** $(x_{n+1}))$ **or** $(x_1$ **and** $\cdots$ **and** $x_{n+1})$ is satisfiable, and is uniquely satisfiable if and only if F is not satisfiable. □

The details of the proof of the next proposition are referred to repeatedly throughout this paper.

PROPOSITION 2.3. (1) *The set* $\{(F, G) | F$ *and G are monotone formulas; and* $\neg[F \leqq G]\}$ *is parsimoniously polynomially reducible to the set* $\{(E_1, \cdots, E_p) | p \geqq 1$; *and* $E_1, \cdots,$ *and* E_p *are simultaneously satisfiable equations on* **BIN**$\}$.

(2) *The set* $\{F | F$ *is a satisfiable Boolean formula*$\}$ *is parsimoniously polynomially reducible to the set* $\{(E_1, \cdots, E_p) | p \geqq 1$; *and* $E_1, \cdots,$ *and* E_p *are simultaneously satisfiable equations on* **BOOLE**$\}$.

Proof. (1) Let F and G be monotone Boolean formulas. Let $x_1, \cdots, x_n$ be the variables appearing in F and G. For subformulas f of F and G, let $\mathbf{E}[f]$ be the system of equations defined recursively as follows:

If f is a variable x, then $\mathbf{E}[f]$ is the equation $v_x = x$;

If f is (f_1 or f_2), then $\mathbf{E}[f]$ is the system of equations

$$\mathbf{E}[f_1], \mathbf{E}[f_2], v_f = (v_{f_1} \textbf{ or } v_{f_2});$$

If f is (f_1 **and** f_2), then $\mathbf{E}[f]$ is the system of equations

$$\mathbf{E}[f_1], \mathbf{E}[f_2], v_f = (v_{f_1} \textbf{ and } v_{f_2}).$$

Let $v_F^1, \cdots, v_F^m$ and $v_G^1, \cdots, v_G^p$ be the variables appearing on the left-hand sides of the equations of $\mathbf{E}[F]$ and of $\mathbf{E}[G]$, respectively. We assume that each of the variables $x_1, \cdots, x_n$, $v_F^1, \cdots, v_F^m$, $v_G^1, \cdots$, and v_G^p is distinct, that v_F^m is the variable v_F and that v_G^p is the variable v_G.

Then, $\neg[F \leqq G]$ if and only if the system of equations

$$\mathbf{E}[F], v_F^m = 1, \qquad \mathbf{E}[G], v_G^p = 0 \tag{2.3.1}$$

has a solution on **BIN**. Moreover, the number of distinct assignments $\mathbf{v}$ of values from $\{0, 1\}$ to the variables $x_1, \cdots, x_n$ such that $\mathbf{v}[F] = 1$ and $\mathbf{v}[G] = 0$ equals the number of distinct assignments of values from $\{0, 1\}$ to the variables $x_1, \cdots, x_n$, $v_F^1, \cdots, v_F^m$, $v_G^1, \cdots$, and v_G^p that satisfy the system of (2.3.1). Thus, the reduction is parsimonious. This can be seen by noting that

(i) $\neg[F \leqq G]$ if and only if there is an assignment $\mathbf{v}$ of values from $\{0, 1\}$ to the variables $x_1, \cdots, x_n$ such that $\mathbf{v}[F] = 1$ and $\mathbf{v}[G] = 0$;

(ii) The possible values of the variables $v_F^1, \cdots, v_F^m, v_G^1, \cdots$, and v_G^p are uniquely determined by the values of the variables $x_1, \cdots$, and x_n; and

(iii) Each collection of values $t_1, \cdots, t_n \in \{0, 1\}$ for $x_1, \cdots, x_n$, respectively, that determines a solution of the system of (2.3.1) is an assignment $\mathbf{v}$ of values from $\{0, 1\}$ to the variables of F and G such that $\neg[\mathbf{v}[F] \leqq \mathbf{v}[G]]$.

(2) The proof is similar and is omitted. □

PROPOSITION 2.4. *Let* $\mathbf{T} = (S, \vee, \wedge)$ *be a nondegenerate distributive lattice. Let* F *and* G *be formulas involving only variables, parentheses, and the operators* $\vee$ *and* $\wedge$. *Let* $a, b \in S$ *with* $a < b$. *Let* **a** *and* **b** *be distinct symbols denoting* a *and* b, *respectively. Then, the following are equivalent*:

(1) $\neg[F \leqq G]$ *on* **T**;

(2) $\neg[F \leqq G]$ *on* **BIN**;

(3) *There exists an assignment of values* $\mathbf{v}$ *from* $\{a, b\}$ *to the variables of* F *and* G *such that* $\mathbf{v}[F] = b$ *and* $\mathbf{v}[G] = a$;

(4) *If we let* $\mathbf{E}[F]$ *and* $\mathbf{E}[G]$ *be the systems of equations defined from* F *and* G, *respectively, as in the proof of Proposition* 2.3, *the system of equations*

$$E[F], v_F^m = \mathbf{b}, \qquad E[G], v_G^p = \mathbf{a} \tag{2.4.1}$$

has an $\{a, b\}$*-valued solution on* **T**;

(5) *If we let* $\mathbf{E}[F \wedge G]$ *equal* $E_{F\wedge G}^1, \cdots, E_{F\wedge G}^q$ *and* $\mathbf{E}[F]$ *equal* $E_F^1, \cdots, E_F^m$ *be the systems of equations defined from* $F \wedge G$ *and* F, *respectively, as in the proof of Proposition* 2.3,

$$\neg\{[E_{F\wedge G}^1 \textbf{ and } \cdots \textbf{ and } E_{F\wedge G}^q \textbf{ and } E_F^1 \textbf{ and } \cdots \textbf{ and } E_F^m] \Rightarrow (v_{F\wedge G}^q = v_F^m)\} \quad \textit{on } \mathbf{T}.$$

Proof. The equivalence of 1 and 2 follows by noting that $F \leqq G$ on **T** if and only if $F \wedge G = F$ on **T** if and only if $F \wedge G = F$ on **BIN** if and only if $F \leqq G$ on **BIN** [Ab], [Bi]. Let $\mathbf{T}' = (\{a, b\}, \vee', \wedge')$, where $\vee'$ and $\wedge'$ are the restrictions of $\vee$ and $\wedge$, respectively, to $\{a, b\} \times \{a, b\}$. The equivalence of 2 and 3 follows by our noting that the lattices $\mathbf{T}'$ and **BIN** are isomorphic. The equivalence of 3 and 4 follows from the proof of 1 of

Proposition 2.3. The equivalence of 1 and 5 follows by our noting that $F \wedge G = F$ on **T** if and only if

$$\{[E^1_{F\wedge G} \textbf{ and } \cdots \textbf{ and } E^q_{F\wedge G} \textbf{ and } E^1_F \textbf{ and } \cdots \textbf{ and } E^m_F] \Rightarrow (v^q_{F\wedge G} = v^m_F)\} \quad \text{on } \mathbf{T}. \qquad \square$$

PROPOSITION 2.5. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be a ring. If there exists $k > l \geqq 1$ and x in* **S** *such that $x^k = x^l \neq 0$, then there exists $y \neq 0$ in* **S** *such that $y = y^2$.*

Proof. Let l be the least positive integer for which there exists $k > l$ such that $x^k = x^l$. Let k_0 be the least such k. Then, for all $n \geqq 0$, $x^l = x^{l+n\cdot(k_0-l)}$. Let $n = l$. Then, $x^l = (x^l)^{1+k_0-l}$. Let $y = (x^l)^{k_0-l}$, then $y \neq 0$ and $y = y^2$. $\square$

PROPOSITION 2.6. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be a ring. Let D be a nonempty subset of* **S** *such that, for all $x, y \in D$, $x = x^2$ and $x \cdot y = y \cdot x$. Let* **g** *and* **h** *be the functions from* $\mathbf{S}\times\mathbf{S}$ *to* **S** *given by, for all $x, y \in \mathbf{S}$,*

$$\mathbf{g}(x, y) = x + y + -(x \cdot y) \quad \textit{and} \quad \mathbf{h}(x, y) = x \cdot y.$$

Let $\hat{D}$ be the closure of D under **g** *and* **h**. *Then, for all $x, y \in \hat{D}$, $x = x^2$ and $x \cdot y = y \cdot x$. Moreover, let* $\hat{\mathbf{g}}$ *and* $\hat{\mathbf{h}}$ *be the restrictions of* **g** *and* **h**, *respectively, to* $\hat{D}\times\hat{D}$. *Then, the functions* $\hat{\mathbf{g}}$ *and* $\hat{\mathbf{h}}$ *are binary functions on $\hat{D}$; and the structure $T = (\hat{D}, \hat{\mathbf{g}}, \hat{\mathbf{h}})$ is a distributive lattice.*

Proof. Let the sets D_i, $i \in N$, be defined inductively by

(i) $D_0 = D$, and

(ii) $D_i = D_{i-1} \cup \{z \in \mathbf{S} | z = x + y + -(x \cdot y)$ or $z = x \cdot y$, where $x, y \in D_{i-1}\}$ for $i \geqq 1$.

Then, $\hat{D} = \bigcup_{i\geqq 0} D_i$. It can be shown by mathematical induction that, for all $i \geqq 0$ and for all $x, y \in D_i$, $x = x^2$ and $x \cdot y = y \cdot x$. (See Appendix.) Thus since $D_i \subset D_j$ for all $0 \leqq i < j$, for all $x, y \in D$, $x = x^2$ and $x \cdot y = y \cdot x$. Once this is known, it is straightforward to verify that the functions $\hat{\mathbf{g}}$ and $\hat{\mathbf{h}}$ are binary functions on D that are associative, commutative, idempotent, absorptive, and mutually distributive. $\square$

PROPOSITION 2.7. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be a Boolean ring with no nonzero zero divisors. Then S is isomorphic to the field* $\mathbf{Z}_2$.

Proof. Let $T = (\mathbf{S}, \vee, \wedge)$, where for all $a, b \in \mathbf{S}$, $a \vee b = a + b + a \cdot b$ and $a \wedge b = a \cdot b$. It is well known [Ab], [Bi] and can easily be verified that T is a distributive lattice.

The proof is by contradiction and proceeds by cases.

Case 1. There exist $a, b \in \mathbf{S}$ such that neither $a \leqq b$ on T nor $b \leqq a$ on T. Let $c = a \wedge b = a \cdot b$ and $d = a \vee b$. The four elements a, b, c, and d are distinct. Moreover, $c = a \cdot b = a \wedge c = a \cdot c$. Hence, $a \cdot (b + c) = 0$. But $a \neq 0$ since the minimal element of T is 0; and $b + c \neq 0$ since $b + b = c + c = 0$. (In any Boolean ring $x + x = 0$ for all x.) Thus, S has nonzero zero divisors, a contradiction.

Thus, the lattice T is a linear order.

Case 2. There exist a, b, c in **S** with $a < b < c$ on T. Then, $a \wedge b = a \cdot b = a \wedge c = a \cdot c$, and thus $a \cdot (b + c) = 0$. But $a \neq 0$ and $b + c \neq 0$. Thus, again S has nonzero zero divisors, a contradiction.

Case 3. $\mathbf{S} = \{0, b\}$ with $0 < b$ on T. Then $S = (\{0, b\}, +, -, \cdot, 0)$ is isomorphic to $\mathbf{Z}_2$. $\square$

3. SQES-problems are "hard." We show that the SQES-problems are **NP**-hard for very simple systems of nonlinear equations on any ring S satisfying either of the following two conditions:

(3a) There exists a nonzero element x of S such that $x = x^2$;

(3b) The ring S has no nonzero zero divisors.

Rings satisfying conditions (3a) or (3b) include each unitary ring, integer domain, field, division ring, ordered ring, and Boolean ring [Bi], [L], [MB].

THEOREM 3.1. *Let* $S = (\mathbf{S}, +, -, \cdot, 0)$ *be a ring for which there exists a nonzero element* a *of* $\mathbf{S}$ *such that* $a = a^2$. *Let* $\mathbf{a}$ *be a constant symbol denoting the element* a. *Then, the SQES-problem is* **NP***-hard for systems of equations on* S *such that*

(1) *The only constant symbols appearing in the system are* $\mathbf{0}$ *and* $\mathbf{a}$,

(2) *The system has a solution if and only if it has a* $\{0, a\}$*-valued solution.*

Proof. Let $f(x_1, \cdots, x_n)$ and $G(x_1, \cdots, x_n)$ be monotone Boolean formulas. Let $\mathbf{E}[F]$ equal to $E_F^1, \cdots, E_F^m$ and $\mathbf{E}[G]$ equal to $E_G^1, \cdots, E_G^p$ be the systems of equations on **BIN** as in the proof of Proposition 2.3. Let v_F^1, v_F^m and $v_G^1, \cdots, v_G^p$ be the variables appearing on the left-hand sides of the corresponding equations of $\mathbf{E}[F]$ and $\mathbf{E}[G]$, respectively. As in the proof of Proposition 2.3, we assume that each of the variables $x_1, \cdots, x_n, v_F^1, \cdots, v_F^m, v_G^1, \cdots, v_G^p$ is distinct and that v_F^m is the variable v_F and v_G^p is the variable v_G. Let $\mathbf{EQ}[F, G]$ be the following system of equations on S:

(1) For $1 \leqq i \leqq n$, $x_i = x_i^2$;

(2) For $1 \leqq i < j \leqq n$, $x_i \cdot x_j = x_j \cdot x_i$;

(3) $E_F^{1'}, \cdots, E_F^{m'}, E_G^{1'}, \cdots, E_G^{p'}$, where, for $U \in \{F, G\}$:

(i) $E_U^{i'}$ is $v_U^i = x$ if E_U^i is $v_U^i = x$,

(ii) $E_U^{i'}$ is $v_U^i = v + w + -(v \cdot w)$ if E_U^i is $v_U^i = v$ **or** w,

(iii) $E_U^{i'}$ is $v_U^i = v \cdot w$ if E_U^i is $v_U^i = v$ **and** w;

(4) $v_F^m = \mathbf{a}$ and $v_G^p = \mathbf{0}$.

Clearly, the only constant symbols occurring in the system of equations $\mathbf{EQ}[F, G]$ are $\mathbf{0}$ and $\mathbf{a}$. Also clearly, the system of equations $\mathbf{EQ}[F, G]$ is constructible deterministically in polynomial time in $|F| + |G|$. We claim that the following are equivalent.

Claim 3.1.1. (a) $\neg[F \leqq G]$;

(b) The system of equations $\mathbf{EQ}[F, G]$ has a solution;

(c) The system of equations $\mathbf{EQ}[F, G]$ has a $\{0, a\}$-valued solution.

Part (4) of Proposition 2.1 and Claim 3.1.1, together, imply that the SQES-problem is *NP*-hard for systems of equations on S satisfying conditions (1) and (2) of the statement of the theorem. Thus, it only remains to prove Claim 3.1.1.

Proof of Claim 3.1.1. Let $V = \{x_1, \cdots, x_n, v_F^1, \cdots, v_F^m, v_G^1, \cdots, v_G^p\}$. Let $\mathbf{g}$ and $\mathbf{h}$ be the binary functions on $\mathbf{S}$ defined by $\mathbf{g}(x, y) = x + y + -(x \cdot y)$ and $\mathbf{h}(x, y) = x \cdot y$. Let $\mathbf{g}'$ and $\mathbf{h}'$ be the restrictions on $\mathbf{g}$ and $\mathbf{h}$, respectively, to $\{0, a\}$. Let $\mathbf{T}'$ be the structure $(\{0, a\}, \mathbf{g}', \mathbf{h}')$. We note that $\mathbf{T}'$ and **BIN** are isomorphic.

Suppose that $\neg[F \in G]$. By the proof of Proposition 2.3, there is an assignment $\mathbf{v}$ of values from the set $\{0, 1\}$ to the variables in V such that the equations $E_F^1, \cdots, E_F^m$, $V_F^m = 1$, $E_G^1, \cdots, E_G^p$, $v_G^p = 0$ are satisfied by $\mathbf{v}$. Let $\mathbf{w}$ be the assignment of values from the set $\{0, a\}$ to the variables in V such that, for all variables y, $\mathbf{w}(y) = a$, if $\mathbf{v}(y) = 1$, and $\mathbf{w}(y) = 0$, if $\mathbf{v}(y) = 0$. Since $\mathbf{T}'$ and **BIN** are isomorphic, the system $\mathbf{EQ}[F, G]$ is satisfied by $\mathbf{w}$.

Suppose that the system of equations $\mathbf{EQ}[F, G]$ has a solution $\mathbf{v}$. Then, the values taken on by the variables in V under $\mathbf{v}$ are uniquely determined by the values taken on by the variables $x_1, \cdots$, and x_n under $\mathbf{v}$. Since the values $\mathbf{v}(x_1), \cdots, \mathbf{v}(x_n)$ satisfy the following equations:

$$\text{for } 1 \leqq i \leqq n \quad \mathbf{v}(x_i) = \mathbf{v}(x_i)^2,$$

$$\text{for } 1 \leqq j \leqq n \quad \mathbf{v}(x_i) \cdot \mathbf{v}(x_j) = \mathbf{v}(x_j) \cdot \mathbf{v}(x_i),$$

the set $D = \{\mathbf{v}(x_1), \cdots, \mathbf{v}(x_n)\}$ is a subset of S such that, for all $x, y \in D$, $x = x^2$ and $x \cdot y = y \cdot x$. From the definitions of the equations $E_F^1, \cdots, E_F^{m'}, E_G^{1'}, \cdots$, and $E_G^{p'}$ above, the values $\mathbf{v}(v_F^1), \cdots, \mathbf{v}(v_F^m), \mathbf{v}(v_G^1), \cdots, \mathbf{v}(v_G^p)$ are in the closure $\hat{D}$ of the set $\mathbf{D}$ under the functions $\mathbf{g}$ and $\mathbf{h}$ of the statement of Proposition 2.6. Also, $\mathbf{v}(v_F^m) = a$ and $\mathbf{v}(v_G^p) = 0$. Let $\mathbf{g}''$ and $\mathbf{h}''$ be the restrictions of the functions $\mathbf{g}$ and $\mathbf{h}$, respectively, to

$\hat{D} \times \hat{D}$. By Proposition 2.6, the structure $\mathbf{T}'' = (\hat{\mathbf{D}}, \mathbf{g}'', \mathbf{h}'')$ is a nondegenerate distributive lattice. Also, $0 < a$ on $\mathbf{T}''$. Thus by the definition of the system $\mathbf{EQ}[F, G]$, the system $E_F^1, \cdots, E_F^m, v_F^m = \mathbf{a}, E_G^1, \cdots, E_G^p, v_G^p = \mathbf{0}$ has a $\{0, a\}$-valued solution on $\mathbf{T}''$, and $\neg[F \leqq G]$. Hence, the system $\mathbf{EQ}[F, G]$ also has a $\{0, a\}$-valued solution. □

THEOREM 3.2. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be a ring with no nonzero zero divisors. Let* **a** *be a constant symbol denoting a nonzero element a of* **S**. *Then, the SQES-problem is* **NP**-*hard for systems of equations on S such that*

(1) *The only constant symbols appearing in the system are* **0** *and* **a**;

(2) *All solutions of the system, if they exist, are $\{0, a\}$-valued.*

Proof. Let $F = c_1$ **and** $\cdots$ **and** $c_m (m \geqq 1)$ be a monotone 3 CNF Boolean formula. Let $x_1, \cdots$, and x_n be the variables appearing in F. For each clause c_j, let V_j be the set of variables appearing in c_j. Let the system of equations $\mathbf{EQ}[F]$ consist of the equations

(3.2.1) for $1 \leqq i \leqq n \quad x_i \cdot (x_i - \mathbf{a}) = \mathbf{0}$,

(3.2.2) for $1 \leqq j \leqq m \sum_{x \in V_j} x - \mathbf{a} = \mathbf{0}$,

(3.2.3) for $1 \leqq j \leqq m$, if $V_j = \{x, y\}$ then $x \cdot y = \mathbf{0}$ else if $V_j = \{x, y, z\}$ then $x \cdot y = x \cdot z = y \cdot z = \mathbf{0}$.

Let **w** be any assignment of values from **S** to the variables $x_1, \cdots$, and x_n such that **w** is a solution of $\mathbf{EQ}[F]$. By (3.2.1), **w** assigns either 0 or **a** to each of the variables $x_1, \cdots$, and x_n. By (3.2.2), **w** assigns the value **a** to at least one variable x in each set V_j. By (3.2.3), **w** assigns the value **a** to at most one variable x in each of the sets V_j. Let **v** be the assignment of values from $\{0, 1\}$ to the variables $x_1, \cdots$, and x_n given by the following:

$$\text{for } 1 \leqq i \leqq n, \mathbf{v}[x_i] = 1, \text{ if } \mathbf{w}[x_i] = \mathbf{a}, \text{ and } \mathbf{v}[x_i] = 0, \text{ if } \mathbf{w}[x_i] = 0.$$

Then $\mathbf{v}[F] = 1$ and, for each clause c_j of F, exactly one variable per clause takes on the value 1 under **v**. Thus, there is a solution **w** of the system of equations $\mathbf{EQ}[F]$ on **S** only if there is a satisfying assignment **v** to the variables of F such that under **v** exactly one variable per clause takes on the value 1. The converse is immediate. Finally, the system of equations $\mathbf{EQ}[F]$ satisfies conditions (1) and (2) of the statement of Theorem 3.2. Thus by part (3) of Proposition 2.1, the SQES-problem is **NP**-hard for systems of equations on **S** satisfying conditions (1) and (2) of the statement of Theorem 3.2. □

For each such S that satisfies the conditions of Theorems 3.1 and 3.2, the problem of solving a system of very simple quadratic equations on S is **NP**-hard, and thus is likely to be computationally intractible. These rings include each unitary ring, each ring S for which there exists $k \geqq 2$ such that for all x in $S x = x^k$ and thus each Boolean ring, each integer domain, each principal ideal domain, each field, each division ring and each ordered ring. For finite rings, more can be said. For each finite ring S, it can easily be seen that

either $\exists n \geqq 1$ such that for all x in S, $x^n = 0$,

or $\exists k > l \geqq 1$ and there exists x in S such that $x^k = x^l \neq 0$.

By Proposition 2.5, if S satisfies the latter case, then S satisfies the conditions of Theorem 3.1 and, thus, the SQES-problem for S is **NP**-hard.

Finally, let **S** be any ring satisfying conditions (1a) or (1b) above. Then each ring $\mathbf{S}^{n \times n}$ of n by n matrices over **S**, each ring **Poly(S)** of formal polynomials over **S**, and

each ring **Power_series(S)** of formal power series over **S** also has a "hard" SQES-problem.

THEOREM 3.3. *Let* $\mathbf{S}=(S,+,-,\cdot,0)$ *be any ring satisfying conditions* (1a) *or* (1b) *above. Then the SQES-problem is* **NP**-*hard for the following*:

(1) *The ring* $\mathbf{S}^{n\times n}$, *for all* $n\geqq 1$;

(2) *The ring* **Poly(S)**;

(3) **Power_series(S)**.

Proof. If **S** satisfies condition (1a) above, then the ring $\mathbf{S}^{n\times n}(n\geqq 1)$, the ring **Poly(S)**, and the ring **Power_series(S)** also satisfy the condition (1a). Thus, the SQES-problem for each such ring is **NP**-hard by Theorem 3.1. If **S** satisfies condition (1b) above, then the rings **Poly(S)** and **Power_series(S)** also satisfy the condition (1b). Thus, the SQES-problem for each such ring is **NP**-hard by Theorem 3.2.

Let **S** be a ring that satisfies condition (1b) above, i.e., **S** has no nonzero zero divisors. Let **a** be any nonzero element of **S**. For $n\geqq 1$, let A_n be the n by n diagonal matrix (a_{ij}) over **S** given by $a_{11}=\mathbf{a}$ and $a_{ij}=0$ otherwise. Let $\mathbf{EQ}'[F]$ be the system of equations over $\mathbf{S}^{n\times n}$ that results from the system of equations $\mathbf{EQ}[F]$ over **S** of the proof of Theorem 3.2 by

(i) replacing each occurrence of a variable x by $\mathbf{A}_n\cdot x\cdot \mathbf{A}_n$;

(ii) replacing each occurrence of the constants **a** and **0** by $\mathbf{A}_n\cdot\mathbf{A}_n\cdot\mathbf{A}_n$ and by the n by n zero matrix, respectively.

Then, $\mathbf{EQ}[F]$ has a solution over **S** if and only if $\mathbf{EQ}'[F]$ has a solution over $\mathbf{S}^{n\times n}$. □

Theorem 3.3 yields a number of rings, not satisfying either condition (1a) or (1b) above, for which the SQES-problem is also **NP**-hard.

4. Some additional NP-hard nonlinear problems. Several additional nonlinear problems for rings are shown to be **NP**- or **coNP**-hard. In Theorems 4.1–4.3, we extend the proofs of Theorems 3.1 and 3.2 to show that the SQE **T**-Satisfiability and the II-Problems are **NP**- and **coNP**-hard, respectively, for all rings S that satisfy the conditions of Theorems 3.1 and 3.2.

THEOREM 4.1. *Let* $S=(\mathbf{S},+,-,\cdot,0)$ *be a ring. Let* **T** *be any subset of* **S** *containing* 0 *and a nonzero element* a *such that* $a=a^2$. *Then, the SQE* **T**-*Satisfiability-problem for* S *is* **NP**-*hard.*

Proof. The theorem follows directly from the proof of Theorem 3.1. □

THEOREM 4.2. Let $S=(\mathbf{S},+,-,\cdot,0)$ *be a ring with no nonzero zero divisors. Let* **T** *be any subset of* **S** *of cardinality* $\geqq 2$. *Then, the SQE* **T**-*Satisfiability-problem for* S *is* **NP**-*hard.*

Proof. The proof closely follows that of Theorem 3.2. Let $a, b\in\mathbf{T}$ with $a\neq b$. Let **a** and **b** be distinct constant symbols denoting a and b, respectively. Let $F=c_1$ **and** $\cdots$ **and** $c_m(m\geqq 1)$ be a monotone 3 CNF Boolean formula. Let $x_1,\cdots$, and x_n be the variable appearing in F. For each clause c_j, let V_j be the set of variables appearing in c_j. Let the system of equations $\mathbf{EQ}[F]$ as follows:

(4.2.1) for $1\leqq j\leqq n$ $(x_i-\mathbf{a})\cdot(x_i-\mathbf{b})=0$,

(4.2.2) for $1\leqq j\leqq m$, if $V_j=\{x,y\}$ then $(\sum_{x\in V_j}x)-(\mathbf{a}+\mathbf{b})=0$ else if $V_j=\{x,y,z\}$ then $(\sum_{x\in V_j}x)-(\mathbf{a}+\mathbf{a}+\mathbf{b})=0$, and

(4.2.3) for $1\leqq j\leqq m$, if $V_j=\{x,y\}$ then $(x-\mathbf{a})\cdot(y-\mathbf{a})=0$ else if $V_j=\{x,y,z\}$ then $(x-\mathbf{a})\cdot(y-\mathbf{a})=(x-\mathbf{a})\cdot(z-\mathbf{a})=(y-\mathbf{a})\cdot(z-\mathbf{a})=0$.

Clearly, any solution of $\mathbf{EQ}[F]$ is $\{a,b\}$-valued and, thus, is **T**-valued. As in the proof of Theorem 3.2, the reader can verify that the following are equivalent:

(i) $\mathbf{EQ}[F]$ has a **T**-valued solution;

(ii) **EQ**[F] has an {**a**, **b**}-valued solution;

(iii) There exists an assignment **v** of values from {0, 1} to the variables of F such that, for all clauses c_i of F, exactly one variable appearing in c_i takes on the value 1 under **v**.

Thus by part (3) of Proposition 2.1, the SQE **T**-Satisfiability-problem is **NP**-hard for each such S and **T**. □

THEOREM 4.3. *Let* $S=(\mathbf{S}, +, -, \cdot, 0)$ *be a ring that satisfies the conditions of Theorems* 3.1 *and* 3.2. *Then, the* II-*Problem for* S is **coNP**-*hard.*

Proof. Let $EQ_1, \cdots, EQ_m (m \geqq 1)$ be a system of simultaneous quadratic equations on S. Let $x_1, \cdots,$ and x_n be the variables occurring in $EQ_1, \cdots,$ or EQ_m. Let u and v be two distinct variables other than $x_1, \cdots,$ and x_n. Then, (EQ_1 **and** $\cdots$ **and** EQ_m)$\Rightarrow$ ($u = u + v$) on S if and only if the system $EQ_1, \cdots, EQ_m$ of simultaneous quadratic equations on S has no solution. □

The proof of Theorem 4.3 suggests that the **coNP**-hard lower bounds on II-Problems of Theorem 4.3 may depend critically upon instances I with unsatisfiable conditional equations. Such instances might be considered unnatural. However, a more complex proof of Theorem 4.3, similar to the proofs of Theorems 3.1 and 3.2, shows that this is not the case. The II-Problems are **coNP**-hard for such rings for instances I whose conditional equations are provably satisfied by an exponential number of different assignments of values to their variables. This proof also shows that the II-Problem is **coNP**-hard for any ring S that satisfies the conditions of Theorem 3.1 for instances with no occurrences of constants.

Finally, the proof of Theorem 4.3 actually shows that for all rings **S** the SQES-Problem for **S** is polynomially reducible to the II-Problem for **S**. Thus, recalling result (3) above from [D], [J], we can immediately conclude the following.

PROPOSITION 4.4. *The* II *Problem for the ring* **Z** *is undecidable.*

5. Nonlinear optimization problems for unitary and ordered rings. Variants of the proofs of Theorems 3.1 and 3.2 can also be used to show that several nonlinear optimization problems and that several very simple analogues of Hilbert's Tenth Problem [D] are also "hard." In Theorem 5.1, we use a variant of the proof of Theorem 3.1 to show the following.

> Let **S** be any unitary ring, and let $\leqq$ be any linear order on **S**. Then, the problems of computing the maximum and the minimum values of a linear function on **S** (under $\leqq$) subject to quadratic constraints are both **NP**- and **coNP**-hard.

In Theorem 5.2 and its corollaries, we use a variant of the proof of Theorem 3.2 to show the following:

> Let **S** be any ordered ring, and let $\leqq$ be the associated linear order. Let **T** be a subset of **S** for which there exist $\mathbf{a}, \mathbf{b} \in \mathbf{T}$ with $\mathbf{a} < \mathbf{b}$ such that $\mathbf{a} = \text{glb}(\mathbf{T})$ and $\mathbf{b} = \text{lub}(\mathbf{T})$ (under $\leqq$). Then, the problems of computing the maximum and the minimum values of a quadratic multiple variable polynomial on **S** (under $\leqq$), for assignments from **T** to the variables, is both **NP**- and **coNP**-hard. In addition, the problem of determining if a quadratic multiple variable polynomial on **S** has **T**-valued roots is **NP**-hard.

One particular corollary is that

> The problem of determining if a quadratic multiple variable polynomial on **R** with integer coefficients has {0, 1}-valued roots is **NP**-complete.

THEOREM 5.1. *Let* $\mathbf{S}=(S,+,-,\cdot,0,1)$ *be an unitary ring. Let* $\leqq$ *be a total order on* $\mathbf{S}$. *Then, the problems of determining the maximum or the minimum value (under* $\leqq$*) taken on by a linear function on* $\mathbf{S}$, *subject to quadratic constraints, are both* **NP**- *and* **coNP**-*hard.*

Proof. The proof follows that of Theorem 3.1. Let $F(x_1,\cdots,x_n)$ and $G(x_1,\cdots,x_n)$ be monotone Boolean formulas. The system of quadratic constraints associated with F and G, denoted by CONSTRAINTS $[F, G]$, consists of the equations of (1), (2) and (3) of the proof of Theorem 3.1, together with the equation

$$v_{F\wedge G}=v_F^m\cdot v_G^P. \tag{4'}$$

Let the linear function on $\mathbf{S}$ associated with F and G, denoted by LIN-FUNC $[F, G]$, be

$$(v_F-v_{F\wedge G}).$$

Then, the following are equivalent:

(a) $F\leqq G$;

(b) $F\wedge G=F$;

(c) For all assignments $\mathbf{v}$ of values from S to the variables of CONSTRAINTS $[F, G]$ and LIN-FUNC $[F, G]$ satisfying the equations in CONSTRAINTS $[F, G]$, $\mathbf{v}[v_F-v_{F\wedge G}]=0$.

Moreover, if $\neg(F\leqq G)$ then there exists an assignment $\mathbf{v}$ of values from S to the variables satisfying the equations in CONSTRAINTS $[F, G]$ such that $\mathbf{v}[v_F-v_{F\wedge G}]=1$.

Thus, $F\leqq G$ if, for all assignments $\mathbf{v}$ of values from S to the variables of LIN-FUNC $[F, G]$ satisfying the equations in CONSTRAINTS $[F, G]$, $\mathbf{v}$[LIN-FUNC $[F, G]]=0$. Also $\neg(F\leqq G)$ if there exists an assignment $\mathbf{v}$ of values from S to the variables of LIN-FUNC $[F, G]$ satisfying the equations in CONSTRAINTS $[F, G]$ such that $\mathbf{v}$[LIN-FUNC $[F, G]]=1$. Thus, the maximal value of LIN-FUNC $[F, G]$ under $\leqq$ subject to the quadratic constraints CONSTRAINTS $[F, G]$ equals 1 if and only if $F\leqq G$. Similarly the minimal value of $v_{F\wedge G}-v_F$ equals -1 if and only if $F\leqq G$. Thus, the claims of the theorem follows from part (4) of Proposition 2.1. □

We note that the idea of the proof of Theorem 5.1 can actually be used to show that optimization problems are **NP**- or **coNP**-hard for each ordered ring with a nonzero element x such that $x=x^2$. (Use x instead of 1.)

THEOREM 5.2. *Let* $S=(\mathbf{S},+,-,\cdot,0)$ *be an ordered ring. Then the following sets are* **NP**-*hard.*

(1) *Let* $\mathbf{T}$ *be any subset of* $\mathbf{S}$ *of cardinality* $\geqq 2$; *the set*

$$\{f \mid f \text{ is a quartic polynomial on } S \text{ that has } \mathbf{T}\text{-valued roots}\};$$

(2) *Let* $\mathbf{T}$ *be any subset of* $\mathbf{S}$ *of cardinality* $\geqq 2$ *for which there exists* $a\in\mathbf{T}$ *such that*

$$\text{for all } x\in\mathbf{T},\ a\leqq x \text{ or, for all } x\in\mathbf{T},\ x\leqq a;$$

the set

$$\{f \mid f \text{ is a cubic polynomial on } S \text{ that has } \mathbf{T}\text{-valued roots}\};$$

(3) *Let* $\mathbf{T}$ *be any subset of* $\mathbf{S}$ *of cardinality* $\geqq 2$ *for which there exists* $a, b\in\mathbf{T}$ *such that*

$$\text{for all } x\in\mathbf{T},\ a\leqq x \text{ and } x\leqq b;$$

the set

$$\{f \mid f \text{ is a quadratic polynomial on } S \text{ that has } \mathbf{T}\text{-valued roots}\}.$$

Proof. Let $F=c_1$ **and** $\cdots$ **and** $c_m (m\geqq 1)$, $V_1,\cdots,V_m$, $x_1,\cdots$, and x_n be as in the proof of Theorem 3.2.

Proof of (1). Let a and b be distinct elements of $\mathbf{T}_1$; and let $\mathbf{a}$ and $\mathbf{b}$ be distinct constant symbols denoting a and b, respectively. Let $p_F(x_1, \cdots, x_n)$ be the quartic polynomial on S

$$\sum_{i=1}^{n} \{(x_i - \mathbf{a}) \cdot (x_i - \mathbf{b})\}^2 + \sum_{\substack{V_j \\ |V_j|=2}} \left\{ \sum_{x \in V_j} x - (\mathbf{a}+\mathbf{b}) \right\}^2 + \sum_{\substack{V_j \\ |V_j|=3}} \left\{ \sum_{x \in V_j} x - (2\mathbf{a}+\mathbf{b}) \right\}^2.$$

Since S is an ordered ring and $a, b \in S$ with $a \neq b$, it follows that $a+a \neq b+b$. Thus for all $a_1, \cdots, a_n \in \mathbf{S}$, $p_F(a_1, \cdots, a_n) = 0$ if and only if

(1i) For $1 \leqq i \leqq n$, $x_i = a$ or $x_i = b$;

(1ii) For all V_j with $|V_j| = 2$, $\sum_{x \in V_j} x = a + b$;

(1iii) For all V_j with $|V_j| = 3$, $\sum_{x \in V_j} x = 2a + b$.

This follows since $p_F(a_1, \cdots, a_n)$ equals a sum of squares t_i^2 and, thus, $p_F(a_1, \cdots, a_n) = 0$ if and only if each $t_i = 0$. Thus, $p_F(x_1, \cdots, x_n)$ has roots if and only if $p_F(x_1, \cdots, a_n)$ has $\{a, b\}$-valued roots if and only if there exists an assignment $\mathbf{v}$ of values from $\{0, 1\}$ to the variables $x_1, \cdots$, and x_n such that, for $1 \leqq j \leqq m$, exactly one variable appearing in clause c_j of F takes on the value 1 under $\mathbf{v}$. Thus by Proposition 2.1, the problem of (1) of the statement of the theorem is **NP**-hard.

Proof of (2). By assumption there exists $a \in \mathbf{T}_2$ such that, $a \leqq x$ for all $x \in \mathbf{T}_2$ or $x \leqq a$ for all $x \in \mathbf{T}_2$. Thus for all $x \in \mathbf{T}_2$, $x - a \geqq 0$ or for all $x \in \mathbf{T}_2$, $a - x \geqq 0$. Let b be an element of $\mathbf{T}_2$ different from a. Let $\mathbf{a}$ and $\mathbf{b}$ be distinct constant symbols denoting a and b, respectively. Let $q(x_1, \cdots, x_n)$ be the cubic multiple variable polynomial on S given by

$$q(x_1, \cdots, x_n) \text{ is } \sum_{i=1}^{n} (x_i - \mathbf{a}) \cdot (x_i - \mathbf{b})^2 \quad \text{if } a \leqq x \text{ for all } x \in \mathbf{T}_2,$$

$$q(x_1, \cdots, x_n) \text{ is } \sum_{i=1}^{n} (\mathbf{a} - x_i) \cdot (x_i - \mathbf{b})^2 \quad \text{if } x \leqq a \text{ for all } x \in \mathbf{T}_2.$$

Let $p_F(x_1, \cdots, x_n)$ be the cubic polynomial on S given by

$$q(x_1, \cdots, x_n) + \sum_{\substack{V_j \\ |V_j|=2}} \left\{ \sum_{x \in V_j} x - (\mathbf{a}+\mathbf{b}) \right\}^2 + \sum_{\substack{V_j \\ |V_j|=3}} \left\{ \sum_{x \in V_j} x - (2\mathbf{a}+\mathbf{b}) \right\}^2.$$

Since $q(a_1, \cdots, a_n)$ equals a sum of nonnegative elements for all $a_1, \cdots, a_n \in \mathbf{T}_2$, $p_F(a_1, \cdots, a_n) = 0$ if and only if

(2i) For $1 \leqq i \leqq n$, $x_i = a$ or $x_i = b$;

(2ii) For all V_j with $|V_j| = 2$, $\sum_{x \in V_j} x = a + b$;

(2iii) For all V_j with $|V_j| = 3$, $\sum_{x \in V_j} x = 2a + b$.

This follows since, for all $a_1, \cdots, a_n \in \mathbf{T}_2$, $p_F(a_1, \cdots, a_n)$ equals a sum of nonnegative elements t_i of S and, thus, $p_F(a_1, \cdots, a_n) = 0$ if and only if each $t_i = 0$.

The remainder of the proof closely follows that of 1 and is omitted.

Proof of (3). By assumption, there exist distinct $a, b \in \mathbf{T}_3$ such that, for all $x \in \mathbf{T}_3$, $a \leqq x$ and $x \leqq b$. Let $\mathbf{a}$ and $\mathbf{b}$ be distinct constant symbols denoting a and b, respectively. Let $p_F(x_1, \cdots, x_n)$ be the quadratic polynomial on S

$$\sum_{i=1}^{n} (x_i - \mathbf{a}) \cdot (\mathbf{b} - x_i) + \sum_{\substack{V_j \\ |V_j|=2}} \left\{ \sum_{x \in V_j} x - (\mathbf{a}+\mathbf{b}) \right\}^2 + \sum_{\substack{V_j \\ |V_j|=3}} \left\{ \sum_{x \in V_j} x - (2\mathbf{a}+\mathbf{b}) \right\}^2.$$

Then for all $x \in \mathbf{T}_3$, $(x - a) \cdot (b - x) \geqq 0$. Thus for all $a_1, \cdots, a_n \in \mathbf{T}_3$, $p_F(a_1, \cdots, a_n) = 0$ if and only if

(3i) For $1 \leqq i \leqq n$, $x_i = a$ or $x_i = b$;

(3ii) For all V_j with $|V_j| = 2$, $\sum_{x \in V_j} x = a + b$;

(3iii) For all V_j with $|V_j| = 3$, $\sum_{x \in V_j} x = 2a + b$.

Again, this follows since for all $a_1, \cdots, a_n \in \mathbf{T}_3$, $p_F(a_1, \cdots, a_n)$ equals a sum of nonnegative elements t_i of S and, thus $p_F(a_1, \cdots, a_n) = 0$ if and only if each $t_i = 0$.

The remainder of the proof also closely follows that of 1 and is omitted. □

The next two direct and immediate corollaries of the proof of Theorem 5.2 show that many optimization problems for quadratic, cubic, and quartic polynomials on ordered rings are also **NP**-hard.

COROLLARY 5.3. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be an ordered ring. Let $\mathbf{T}_1$ be any subset of $\mathbf{S}$ of cardinality $\geqq 2$. Let $\mathbf{T}_2$ and $\mathbf{T}_3$ be any subsets of $\mathbf{S}$ of cardinalities $\geqq 2$ such that*

(i) *There exists $a \in \mathbf{T}_2$ for which $a \leqq x$ for all $x \in \mathbf{T}_2$ or $x \leqq a$ for all $x \in \mathbf{T}_2$;*

(ii) *There exists $a, b \in \mathbf{T}_3$ for which $a \leqq x$ and $x \leqq b$ for all $x \in \mathbf{T}_3$.*

Then, the following problems are **NP**-*hard*:

(1) *The problems of determining, given a quartic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_1\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_1\} = 0;$$

(2) *The problems of determining, given a cubic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_2\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_2\} = 0;$$

(3) *The problems of determining, given a quadratic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_3\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in T_3\} = 0.$$

Proof. The proof follows immediately from the proof of Theorem 5.2 since in each case the polynomial $p_F(x_1, \cdots, x_n)$ equals a sum of nonnegative elements of S. □

COROLLARY 5.4. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be an ordered ring. Let a and b be distinct elements of* $\mathbf{S}$. *Then, the following problems are* **NP**-*hard*:

(1) *The problems of determining, given a quartic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in S\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a_1, \cdots, a_n \in S\} = 0;$$

(2) *The problems of determining, given a cubic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a \leqq a_1, \cdots, \text{ and } a \leqq a_n\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a_1 \leqq a, \cdots, \text{ and } a_n \leqq a\} = 0;$$

(3) *The problems of determining, given a quadratic polynomial $f(x_1, \cdots, x_n)$ on S if*

$$\max \{f(a_1, \cdots, a_n) | a \leqq a_1 \leqq b, \cdots, \text{ and } a \leqq a_n \leqq b\} = 0, \text{ and if}$$

$$\min \{f(a_1, \cdots, a_n) | a \leqq a_1 \leqq b, \cdots, \text{ and } a \leqq a_n \leqq b\} = 0.$$

Proof. The proof follows immediately from Corollary 5.3, letting $\mathbf{T}_1$ be S, $\mathbf{T}_2$ be $\{x \in S | a \leqq x\}$ or $\{x \in S | x \leqq a\}$, and $\mathbf{T}_3$ be $\{x \in S | a \leqq x \leqq b\}$. □

Finally, results analogous to Theorem 5.2 hold for additional nonordered rings. For example the following analogue of part (3) of Theorem 5.2 holds for the ring of complex numbers **C**, provided we allow complex conjugation $^-$ as a basic operation in formulas and we consider the degree of polynomials of the form $x \cdot \bar{y}$, $\bar{x} \cdot y$, and $\bar{x} \cdot \bar{y}$ to be 2. Here, $\mathrm{Re}(x) = (x+\bar{x})/2$.

PROPOSITION 5.5. *Let* **T** *be any subset of* **C** *for which there exist distinct real numbers a and b such that*

(i) *For all* $x \in \mathbf{T}$, $a \leqq \mathrm{Re}(x) \leqq b$;

(ii) *There exist* $x_1, x_2 \in \mathbf{T}$ *such that* $a = \mathrm{Re}(x_1)$ *and* $b = \mathrm{Re}(x_2)$.

Then, the set

$$\{f \mid f \text{ is a quadratic polynomial on } C \text{ that has } \mathbf{T}\text{-valued roots}\}$$

is **NP**-*hard.*

Proof. The proof follows directly from the proof of part (3) of Theorem 5.2, letting f be the quadratic polynomial

$$\sum_{i=1}^{n} \left(\frac{x_i+\bar{x}_i}{2} - a\right) \cdot \left(b - \frac{x_i+\bar{x}_i}{2}\right) + \sum_{\substack{V_j \\ |V_j|=2}} \left\{\sum_{x \in V_j} \frac{x+\bar{x}}{2} - (a+b)\right\}^2 + \sum_{\substack{V_j \\ |V_j|=3}} \left\{\sum_{x \in V_j} \frac{x+\bar{x}}{2} - (2a+b)\right\}^2. \qquad \square$$

6. Some "hard" enumeration problems. In this section, we show that several enumeration problems related to problems *A*, *B*, and *F* above are #**P**-hard. In Theorem 6.1, we show that the problem of computing the number of $\{0, 1\}$-valued solutions of a system of quadratic equations on a unitary ring is #**P**-hard. In Theorem 6.2, we show that the problem of computing the number of solutions of a system of quadratic equations on an ordered ring is #**P**-hard. In Theorem 6.3, we show that the problem of computing the number of **T**-valued roots of a quadratic multiple variable polynomial on an ordered ring is #**P**-hard. Finally in Theorem 6.4, we show that the problem of determining if a quadratic multiple variable polynomial on an ordered ring has a unique $\{0, 1\}$-valued root is **coNP**-hard.

THEOREM 6.1. *Let* $S = (\mathbf{S}, +, -, \cdot, 0)$ *be a unitary ring. Then,*

(1) *The problem of computing the number of* $\{0, 1\}$*-valued solutions of a system of quadratic equations on S is* #**P**-*hard*;

(2) *The problem of determining if a system of quadratic equations on S, that has a* $\{0, 1\}$*-valued solution, has a unique* $\{0, 1\}$*-valued solution is* **coNP**-*hard.*

Proof. The proof of (1) follows from part (2) of Proposition 2.1 and Proposition 2.3 by a proof closely analogous to that of Theorem 3.1. Let F be a monotone 2 CNF Boolean formula. Let $x_1, \cdots$, and $x_n (n \geqq 1)$ be the variables appearing in F. Let $E[F]$ equal to $E_F^1, \cdots, E_F^m$ be the system of equations on **BIN**, associated with F, as in the proof of Proposition 2.3. Let $v_F^1, \cdots, v_F^m$ be the variables appearing on the left-hand sides of the corresponding equations of $\mathbf{E}[F]$. As in the proof of Proposition 2.3, we assume that each of the variables $x_1, \cdots, x_n, v_F^1, \cdots$, and v_F^m are distinct and that v_F^m is the variable v_F. Let $\mathbf{EQ}[F]$ be the following system of equations on S:

(1) For $1 \leqq i \leqq n$, $x_i(x_i - \mathbf{1}) = \mathbf{0}$;

(2) For $1 \leqq j \leqq m$, the equation $E_F^{j'}$ is defined as in the proof of Theorem 3.1;

(3) $v_F^m = \mathbf{1}$.

For any solution of the system $\mathbf{EQ}[F]$, the values of the variables $v_F^i (1 \leqq i \leqq m)$ are uniquely determined by the values of the variables $x_j (1 \leqq j \leqq n)$. Moreover if

$x_1, \cdots, x_n \in \{0, 1\}$, then $v_F^1, \cdots, v_F^m \in \{0, 1\}$. Thus, the number of $\{0, 1\}$-valued solutions of the system $\mathbf{EQ}[F]$ equals the number of assignments $\mathbf{v}$ of values from $\{0, 1\}$ to the variables of F such that $\mathbf{v}[F] = 1$. (The reduction is parsimonious.)

The proof of (2) follows from Propositions 2.2 and 2.3 by an analogous argument. □

THEOREM 6.2. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be any ring with no nonzero zero divisors. Then, the problem of computing the number of solutions of a system of quadratic equations on S is $\#\mathbf{P}$-hard.*

Proof. The theorem follows from part (2) of Proposition 2.1, Proposition 2.7 and Theorem 6.1. There are two cases to consider.

Case 1. There exists $a \in \mathbf{S}$ with $a + a \neq 0$ or there exists $a \in \mathbf{S}$ with $a \neq a^2$. Let $F = c_1$ **and** $\cdots$ **and** $c_m (m \geqq 1)$ be a monotone 2 CNF Boolean formula. Let $x_1, \cdots$, and x_n be the variables appearing in F. For each clause c_i, let $V_i = \{x_{i1}, x_{i2}\}$ be the set of variables appearing in c_i.

Suppose there exists $a \in \mathbf{S}$ with $a + a \neq 0$. Let $\mathbf{a}$ be a constant symbol denoting a. Let the system $\mathbf{EQ}[F]$ consist of the following equations:

$$\text{(6.2.1)} \qquad \text{for } 1 \leqq i \leqq n \quad x_i \cdot (x_i - \mathbf{a}) = 0,$$

$$\text{(6.2.2)} \qquad \text{for } 1 \leqq j \leqq m \quad u_j = x_{j1} + x_{j2} \text{ and } (u_j - \mathbf{a}) \cdot (u_j - (\mathbf{a} + \mathbf{a})) = 0.$$

Otherwise, let $a \in \mathbf{S}$ such that $a \neq a^2$. Let $\mathbf{a}$ be a constant symbol denoting a. Let the system $\mathbf{EQ}[F]$ consist of (6.2.1) together with the following:

$$\text{(6.2.3)} \qquad \begin{aligned} &\text{for } 1 \leqq j \leqq m,\ u_j = x_{j1} + x_{j2}^2 \\ &v_j = (u_j - \mathbf{a}) \cdot (u_j - \mathbf{a}^2), \\ &w_j = v_j \cdot (u_j - \mathbf{a} - \mathbf{a}^2), \\ &w_j = \mathbf{0}. \end{aligned}$$

For any solution $\mathbf{w}$ of $\mathbf{EQ}[F]$, the values taken on by the variables of $\mathbf{EQ}[F]$ under $\mathbf{w}$ are uniquely determined by the values taken on by the variables $x_1, \cdots$, and x_n under $\mathbf{w}$. Thus, it is easily verified that the number of solutions of F equals the number of solutions of $\mathbf{EQ}[F]$. Hence, the conclusion of the theorem for the ring S follows from Proposition 2.1.

Case 2. For all $a \in \mathbf{S}$, $a + a = 0$ and $a = a^2$. Then, S is a Boolean ring with no nonzero zero divisors. Thus by Proposition 2.7, the ring S is isomorphic to the field $\mathbf{Z}_2$. Thus, the conclusion of the theorem for the ring S follows from Theorem 6.1. □

Computing the number of $\mathbf{T}$-valued roots is $\#\mathbf{P}$-hard for quadratic, cubic, and quartic polynomials on any ordered ring.

THEOREM 6.3. *Let $S = (\mathbf{S}, +, -, \cdot, 0)$ be an ordered ring. Let $\mathbf{T}_1$ be any subset of $\mathbf{S}$ of cardinality $\geqq 2$. Let $\mathbf{T}_2$ and $\mathbf{T}_3$ be any subsets of $\mathbf{S}$ of cardinalities $\geqq 2$ such that*

(i) *There exists $a \in \mathbf{T}_2$ for which either $a \leqq x$ for all $x \in \mathbf{T}_2$ or $x \leqq a$ for all $x \in \mathbf{T}_2$;*

(ii) *There exists $a, b \in \mathbf{T}_3$ for which $a \leqq x$ and $x \leqq b$ for all $x \in \mathbf{T}_3$.*

Then, the following problems are $\#\mathbf{P}$-hard:

(1) *The problem of computing the number of $\mathbf{T}_1$-valued roots of a quartic polynomial on S;*

(2) *The problem of computing the number of $\mathbf{T}_2$-valued roots of a cubic polynomial on S;*

(3) *The problem of computing the number of $\mathbf{T}_3$-valued roots of a quadratic polynomial on S.*

Proof. The theorem follows from part (2) of Proposition 2.1 by an argument similar to that of the proof of Theorem 5.2. We only sketch the proof of (3).

Let $F = c_1$ **and** $\cdots$ **and** $c_m (m \geqq 1)$ be a monotone 2 CNF Boolean formula. Let $x_1, \cdots,$ and x_n be the variables appearing in F. For $1 \leqq j \leqq m$, let V_j be the set of variables appearing in clause c_j. Without loss of generality, we assume that $|V_j| = 2$ for $1 \leqq j \leqq m$. Let **a** and **b** be distinct constant symbols denoting a and b, respectively. Let $p_F(x_1, \cdots, x_n)$ be the quadratic polynomial on S

$$\sum_{i=1}^{n} (x_i - \mathbf{a}) \cdot (\mathbf{b} - x_i) + \sum_{j=1}^{m} (y - \mathbf{a}) \cdot (\mathbf{b} - y_j) + \sum_{j=1}^{m} \left\{ \sum_{x \in V_j} x - (\mathbf{b} + y_j) \right\}^2.$$

Then, the number of $\mathbf{T}_3$-valued roots of $p_F(x_1, \cdots, x_n)$ equals the number of assignments **v** of values from $\{0, 1\}$ to the variables $x_1, \cdots,$ and x_n of F such that $\mathbf{v}[F] = 1$. □

Finally for ordered unitary rings S and for the normed rings as **C** and **Qu**, the problem of determining if a quadratic polynomial on **S** has a unique $\{0, 1\}$-valued root is **coNP**-hard.

THEOREM 6.4. *Let $S = (S, +, -, \cdot, 0, 1)$ be an ordered unitary ring. Then, the set*

$$\{f | f \text{ is a quadratic polynomial on } S \text{ that has a unique } \{0, 1\}\text{-valued root}\}$$

is **coNP**-*hard.*

Proof. We actually show the following.

Claim 6.4.1. There exists a deterministic polynomially time-bounded algorithm A such that A, given a Boolean formula f as input, outputs a quadratic p_f on S with integer coefficients such that the number of different assignments **v** of values from $\{0, 1\}$ to the variables of f such that $\mathbf{v}[f] = 1$ equals the number of different assignments **w** of values from $\{0, 1\}$ to the variables of p_f such that $\mathbf{w}[p_f] = 0$.

Once Claim 6.4.1 is proven correct, the theorem follows directly from Proposition 2.2.

Let $f(x_1, \cdots, x_n) (n \geqq 1)$ be a Boolean formula. For all subformulas g of f, let the system of equations $\mathbf{E}[g]$ be defined recursively by:

If g is a variable x, then $\mathbf{E}[g]$ is the equation $v_x = x$;

If g is (**not** g_1), then $\mathbf{E}[g]$ is the system of equations

$$\mathbf{E}[g_1],\ v_g = 1 + -v_{g_1};$$

If g is (g_1 **or** g_2), then $\mathbf{E}[g]$ is the system of equations

$$\mathbf{E}[g_1], \mathbf{E}[g_2],\ v_g = v_{g_1} + v_{g_2} + -v_{g_1} \cdot v_{g_2};$$

If g is (g_1 **and** g_2), then $\mathbf{E}[g]$ is the system of equations

$$\mathbf{E}[g_1], \mathbf{E}[g_2],\ v_g = v_{g_1} \cdot v_{g_2}.$$

Let $\mathbf{E}'[f]$ be the system of equations $\mathbf{E}[f]$, $v_f = 1$. Clearly, the system of equations $\mathbf{E}'[f]$ is constructible from **f** deterministically in polynomial time. We leave it to the reader to verify the following.

Claim 6.4.2. The number of different assignments **v** of values from $\{0, 1\}$ to the variables of f such that $\mathbf{v}[f] = 1$ equals the number of different assignments **w** of values from $\{0, 1\}$ to the variables of the system $\mathbf{E}'[f]$ that satisfy it.

Let m be the number of equations in the system $\mathbf{E}'[f]$. For each of these equations EQ_i, $1 \leqq i \leqq m$, let EQ_i' be the formula

$$\text{left-hand side of } EQ_i + -\text{right-hand side of } EQ_i.$$

Then for all assignments **w** of values from $\{0, 1\}$ to the variables of $\mathbf{E}'[f]$, $\mathbf{w}[E_i'] \in \{0, 1, -1\}$, for $1 \leqq i \leqq m$. Let p_f be the quadratic polynomial with integer coefficients

on l

$$\sum_{i=1}^{m} 2^i \cdot EQ'_i.$$

Then, we claim the following.

Claim 6.4.3. For all assignments $\mathbf{w}$ of values from $\{0, 1\}$ to the variables of p_f, $\mathbf{w}[p_f] = 0$ if and only if, for $1 \leqq i \leqq m$, $\mathbf{w}[EQ'_i] = 0$ if and only if $\mathbf{w}$ satisfies the system $\mathbf{E}'[f]$.

To verify Claim 6.4.3, it suffices to verify that

$$\sum_{i=1}^{m} 2^i \cdot c_i = 0 \quad \text{where for } 1 \leqq i \leqq m \quad c_i \in \{0, 1, -1\}$$

if and only if

$$\text{for } 1 \leqq i \leqq m \quad c_i = 0.$$

But this follows easily by proof by contradiction from the formula for the sum of a finite geometric series.

Finally, the correctness of Claim 6.4.1 follows directly from the correctness of Claims 6.4.2 and 6.4.3. □

7. Conclusions. We conclude with three additional observations on the results of this paper. First, assuming that $\mathbf{P} \neq \mathbf{NP}$, we have shown that the SQES-problem and several additional nonlinear problems are computationally intractable for very simple systems of equations for most of the rings studied in mathematics and/or computer science. The power of linear algebra is that it provides deterministic polynomially time-bounded algorithms for solving systems of linear equations. The results of this paper show why little analogous success has been obtained for solving nonlinear systems of equations on rings. Moreover even if $\mathbf{P} = \mathbf{NP}$, the proofs in this paper can be used to show that several of the nonlinear problems studied here are still likely to be computationally infeasible. In particular in [SH], we use the proofs in this paper to show the following.

> For all reals $r > 1$, if the Satisfiability problem for Boolean formulas requires time $\Omega(n^r)$ i.o. on any deterministic Turing machine, then the SQES-Problem also requires time $\Omega(n^r)$ i.o. on any deterministic Turing machine, for each commutative ring satisfying condition (1a) and for each ring satisfying condition (1b).

Second, nonlinear algebra, especially the SQES-Problem, can be shown to be "hard" for a number of additional algebraic structures including many of the semirings and lattices studied in mathematics and computer science [HS2].

Third, the SQES-Problem is decidable deterministically in polynomial time for many rings S for which, for all x, y in S, $x \cdot y = 0$. We do not know if the SQES-Problem is decidable deterministically in polynomial time for any other rings.

Appendix (Proof of Proposition 2.6). We prove the induction step only when $x, y \in D_{i+1}$, where $x = \alpha + \beta + -(\alpha \cdot \beta)$ and $y = \gamma + \delta + -(\gamma \cdot \delta)$, $\alpha, \beta, \gamma, \delta \in D_i$ and, for all $u, v \in D_i$, $u = u \cdot u$ and $u \cdot v = v \cdot u$. The remaining cases of the induction step are similar and are left to the reader.

By repeated applications of the associativity of $\cdot$ and of the idempotency and commutativity of $\cdot$ for the elements of D_i, for all $\alpha, \beta, \gamma, \delta \in D_i$,

(a) $\alpha \cdot (-\alpha \cdot \beta) = (-\alpha \cdot \beta) \cdot \alpha = -\alpha \cdot \beta = -(\alpha \cdot \beta)$,

(b) $\beta \cdot (-\alpha \cdot \beta) = (-\alpha \cdot \beta) \cdot \beta = -\alpha \cdot \beta = -(\alpha \cdot \beta)$,

(c) $(-\alpha \cdot \beta) \cdot (-\alpha \cdot \beta) = \alpha \cdot \beta$,

(d) $\alpha \cdot (-\gamma \cdot \delta) = (-\gamma \cdot \delta) \cdot \alpha$,

(e) $(-\alpha \cdot \beta) \cdot (-\gamma \cdot \delta) = (-\gamma \cdot \delta) \cdot (-\alpha \cdot \beta)$.

Thus,

$$\begin{aligned} x \cdot x &= [\alpha + \beta + -(\alpha \cdot \beta)] \cdot [\alpha + \beta + -(\alpha \cdot \beta)] \\ &= \alpha + \alpha \cdot \beta + (-\alpha \cdot \beta) + \alpha \cdot \beta + \beta + (-\alpha \cdot \beta) + (-\alpha \cdot \beta) + (-\alpha \cdot \beta) + (\alpha \cdot \beta) \\ &= \alpha + \beta + (-\alpha \cdot \beta) \\ &= x, \end{aligned}$$

and

$$\begin{aligned} x \cdot y &= [\alpha + \beta + -(\alpha \cdot \beta)] \cdot [\gamma + \delta + -(\gamma \cdot \delta)] \\ &= \alpha \cdot \gamma + \alpha \cdot \delta + \alpha \cdot (-\gamma \cdot \delta) + \beta \cdot \gamma + \beta \cdot \delta + \beta \cdot (-\gamma \cdot \delta) \\ &\qquad + (-\alpha \cdot \beta) \cdot \gamma + (-\alpha \cdot \beta) \cdot \delta + (-\alpha \cdot \beta) \cdot (-\gamma \cdot \delta) \\ &= [\gamma + \delta + -(\gamma \cdot \delta)] \cdot [\alpha + \beta + -(\alpha \cdot \beta)] \\ &= y \cdot x. \end{aligned}$$

REFERENCES

[Ab] J. C. ABBOTT, *Sets, Lattices, and Boolean Algebras*, Allyn and Bacon, Boston, MA, 1969.

[AM] L. ADLEMAN AND K. MANDERS, *Computational complexity of decision procedures for polynomials*, Proc. 16th Annual IEEE Symposium on Foundations of Computer Science, Berkeley, CA, 1975, pp. 169-177.

[Bi] G. BIRKHOFF, *Lattice Theory*, 3rd ed., American Mathematical Society, Providence, RI, 1967.

[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.

[Co] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.

[D] M. DAVIS, *Computability and Unsolvability*, Dover, New York, 1982.

[FY] A. S. FRAENKEL AND Y. YESHA, *Complexity of problems in games, graphs, and algebraic equations*, Discrete Appl. Math., 1 (1979), pp. 15-30.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[HS1] H. B. HUNT III AND R. E. STEARNS, *Distributive lattices, and the complexity of logics and probability*, submitted for publication.

[HS2] ———, *Nonlinear algebra and optimization on semirings and lattices are "hard,"* in preparation.

[IM] O. H. IBARRA AND S. MORAN, *Probabilistic algorithms for deciding equivalence of straight-line programs*, J. Assoc. Comput. Mach., 30 (1983), pp. 217-228.

[J] R. G. JEROSLOW, *There cannot be any algorithm for integer programming with quadratic constraints*, Oper. Res., 21 (1973), pp. 221-224.

[L] S. LANG, *Algebra*, Addison-Wesley, Reading, MA, 1965.

[MB] S. MACLANE AND G. BIRKHOFF, *Algebra*, Macmillan, New York, 1967.

[Ma] YU. V. MATIJASEVIC, *Enumerable sets are diophantine*, Soviet Math. Dokl., 11 (1970), pp. 354-357 (English translation).

[Sa] S. SAHNI, *Computationally related problems*, this Journal, 3 (1974), pp. 262-279.

[Sc] T. J. SCHAEFER, *The complexity of satisfiability problems*, Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 216-226.

[Si] J. SIMON, *On the difference between the one and the many (preliminary version)*, in Automata, Languages, and Programming, Lecture Notes in Computer Science, 52, Springer, Berlin, pp. 480-491.

[SH] R. E. STEARNS AND H. B. HUNT III, *On the complexity of the satisfiability problem and the structure of NP*, in preparation.

[Sz] J. T. SCHWARTZ, *Fast probabilistic algorithms for verification of polynomial identities*, J. Assoc. Comput. Mach., 27 (1980), pp. 701-717.

[Va] L. G. VALIANT, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410-421.

[van] B. L. VAN DER WAERDEN, *Modern Algebra*, Vol. 1, Frederick Ungar, New York, 1953.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987
© 1987 Society for Industrial and Applied Mathematics
009

COMPUTING POWERS IN PARALLEL*

JOACHIM VON ZUR GATHEN†

Abstract. Fast parallel computations are presented for large powers modulo an element that has only small prime factors. They work for integers and polynomials over small finite fields.

Key words. parallel processing, circuit depth, algebraic computing, symbolic manipulation, powers of integers, powers of polynomials

AMS(MOS) subject classifications. Primary 68C20; secondary 10A30, 12C05

1. Introduction. Consider the problem of computing $a^b \bmod m$ in parallel, where a, b, m are n-bit integers. This problem and related tasks come up as a subroutine in many computational problems, e.g., factoring integers, primality tests, cryptographic protocols, and factoring polynomials over finite fields. The method of "repeated squaring" does not yield fast parallel computations, i.e., of parallel time $(\log n)^{O(1)}$.

We present Boolean circuits for this problem. Provided that the modulus m has only small prime factors $p \leqq n$ ("m is n-smooth"), the depth is $O(\log^2 n \operatorname{loglog} n)$ with polynomial size for log-space uniform families, and the depth is of the optimal order $O(\log n)$ for P-uniform families.

In § 2, we use the analogue of Boolean circuits for parallel computations over fields, called arithmetic circuits. We determine exactly the parallel complexity of the powering problem over infinite fields in this model. In particular, no arithmetic circuits of poly-logarithmic depth exist for this problem. We also present the basic technique used for the fast parallel algorithms of the next sections; of course, it works only in a special case.

Section 3 deals with the case of computing $a^b \bmod m$, where m is a power of a small prime number; this is the core of the algorithm. Section 4 starts with some auxiliary parallel algorithms, for factoring numbers and Chinese remaindering, and then describes the powering algorithm. The fact that factoring can (trivially) be achieved fast in parallel puts the constraint of smoothness into perspective: it is a severe restriction, and for many applications the more interesting case is that of large prime factors. However, the present result is the first one that provides an exponential parallel speed-up of the sequential methods for modular integer powering. In particular, it disproves a conjecture in the first version of Reif [17]. A slightly different problem—computing in parallel an approximation to the high-order bits of a large power of an integer—is considered in Alt [1].

In §§ 5 and 6 we translate the approach into the setting of polynomials over finite fields. Under the corresponding constraint that m have only small irreducible factors, we get shallow Boolean circuits of polynomial size for the powering problem. The depth is $O(\log^2 n \operatorname{loglog} n)$ under log-space uniformity, and $O(\log n)$ under P-uniformity.

Subsequent to the present paper, Fich and Tompa [6] obtained a fast parallel powering algorithm in large finite fields of small characteristic. This leads to the

* Received by the editors January 4, 1985; accepted for publication (in revised form) December 15, 1986. A preliminary version of this work appeared in the Proceedings of the 25th Annual IEEE Symposium on the Foundations of Computer Science, Singer Island, Florida, 1984. Part of this work was done while the author was visiting Universität Zürich, Zürich, Switzerland. This work was supported by Schweizerischer Nationalfonds grant 2175-0.83, and by Natural Sciences and Engineering Research Council of Canada grant 3-650-126-40.

† Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada.

surprising observation that for certain polynomial computations, Boolean circuits are exponentially more powerful than arithmetic circuits (von zur Gathen and Seroussi [10]). The problem of computing large powers discriminates between the two models, also other natural problems like testing for quadratic residuosity.

2. Powers over fields. In this section, we consider the standard model for parallel computations over fields: arithmetic circuits. It is easy to determine the parallel complexity (=depth) of the powering problem exactly over infinite fields, and almost exactly over finite fields (Theorem 2.5). In particular, in this model no computation of depth $(\log n)^{O(1)}$ for powers with an n-bit exponent is possible, except (trivially) when the field is very small, say has at most n elements. The reader interested only in the algorithms of §§ 3 through 6 may skip this section.

It is always interesting to have problems whose computational complexity we can determine exactly. The main motivation here is, however, to provide some justification for the assumption that we make later, namely that the ground field be very small.

Let F be an arbitrary field. The model of computation in this section are Strassen's [19] arithmetic circuits (or "straight-line programs"). We only need to consider special arithmetic circuits, with one input variable and one specially designated output node. (See von zur Gathen [9] for general parallel arithmetic computations.) Such an arithmetic circuit α over $F \cup \{x\}$ takes as inputs x and constants from F, performs arithmetic operations $+, -, *$, and $/$, and produces an output. At each node of the computation graph a rational function from $F(x)$ is computed. $\psi_\alpha \in F(x)$ is the rational function computed at the output node. On substituting some $a \in F$ for x, field elements are computed. We stipulate that in α no division by the rational function zero occurs. However, for certain substitutions $a \in F$, a division by the field element zero may occur.

$$R_\alpha = \{a \in F: \text{no division by zero occurs for } x \leftarrow a\}$$

is the domain of definition of α. $F \setminus R_\alpha$ is finite. If $A \subseteq F$ and $f: A \to F$ is a total function, then α computes f if and only if $\psi_\alpha(a) = f(a)$ for all $a \in A \cap R_\alpha$. If A is infinite and α computes f, then $f = \psi_\alpha \restriction A$ is a rational function. (We ignore issues of "uniformity" for arithmetic circuits, i.e., how the individual circuits α_n of a family $(\alpha_n)_{n \in \mathbb{N}}$ can be computed, given n.)

As usual, the size (=sequential time) $S(\alpha)$ of α is the total number of arithmetic operations in α, and the depth (=parallel time) $D(\alpha)$ is the length of a longest chain of subsequent operations. For a function $f: A \to F$, where $A \subseteq F$, the depth $d(f)$ is the smallest depth of arithmetic circuits α over $F \cup \{x\}$ which compute f. (If there is no α computing f, then $d(f) = \infty$.) We also denote by $d^+(f)$ the smallest depth of arithmetic circuits α over $F \cup \{x\}$ which compute f everywhere, i.e., with $A \subseteq R_\alpha$. Clearly $d \leqq d^+$, and if A is infinite, then $d = d^+$. For $A \subseteq F$ and $b \in \mathbb{N}$, let

$$\pi_A^b: A \to F, \qquad a \mapsto a^b$$

be the powering function. When F is finite and $A \subseteq F$ "large", then a computation for π_A^b is an "approximate" computation for π_F^b. For a finite field $F = GF(p^l)$, with p prime, the functions $\pi_F^{p^e}$ are important for extracting pth roots in F [10] and squarefree decomposition in $F[x]$ [7]. Well-known algorithms give the following upper bounds on the depth. (Throughout the paper "log" means $\log_2$.)

FACT 2.1. *Let F be a field, $A \subseteq F$ finite with s elements, $b, c, k \in \mathbb{N}$, $b \equiv c \bmod k$, $b \geqq 1$, $1 \leqq c < k$ and $f: A \to F$ any function. Then*

(i) $d^+(f) \leqq \log s + (2 \log s)^{1/2} + 3$,

(ii) *F algebraically closed* $\Rightarrow d^+(f) \leqq \lceil \log s \rceil + 2$,

(iii) $d^+(\pi_F^b) \leqq \lceil \log b \rceil$,

(iv) $A \subseteq \{a \in F: a^{k+1} = a\} \Rightarrow d^+(\pi_A^b) \leqq \lceil \log c \rceil$,

(v) $A \subseteq \{a \in F: a^k = 1\} \Rightarrow d^+(\pi_A^b) \leqq \lceil \log (k-c) \rceil + 1$.

Proof. (i) and (ii) follow from Munro and Paterson [16], applied to the polynomial in $F[x]$ interpolating f over A. The method of "repeated squaring" (see [13, § 4.6.3]) proves (iii). In (iv), we have $\pi_A^b = \pi_A^c$, and in (v), $\pi_A^b = 1/\pi_A^{k-c}$. □

An upper bound $d(f) \leqq \lceil \log s \rceil + 2$ holds in (i) for any field, with a rather unproductive circuit α computing $\prod_{a \in A} (x-a)^{-1}$, so that $A \cap R_\alpha = \varnothing$. If we allow additions for free, then Kung [14] has a surprising algorithm that uses depth 2.

To put the subsequent results into perspective, we next note some lower bounds on $d(\pi_A^b)$ and $d^+(\pi_A^b)$. The degree $\deg f$ of a rational function $f \in F(x)$ is $\max \{\deg g, 1 + \deg h\}$, if $f = g/h$ and $g, h \in F[x]$ with $\gcd(g, h) = 1$. Kung [14] proves the following lower bound for any arithmetic circuit α.

FACT 2.2. $\deg \psi_\alpha \leqq 2^{D(\alpha)}$. □

For a polynomial $f \in F[x]$ and $A \subseteq F$, we denote by $f \restriction A: A \to F$ the map induced by f. We use the monus function $a \dot{-} b = \max \{a - b, 0\}$.

LEMMA 2.3. *Let F be an arbitrary field, $A \subseteq F$ be finite with $s \geqq 2$ elements, $f \in F[x] \setminus \{0\}$ of degree $b \geqq 1$, α an arithmetic circuit over $F \cup \{x\}$ computing $f \restriction A$, and $d = D(\alpha)$.*

(i) *If $A \subseteq R_\alpha$ then $d \geqq \min \{\log b, \log (s \dot{-} b + 1)\}$.*

(ii) *If $A \cap R_\alpha = \varnothing$, then $d \geqq \log s - \operatorname{loglog} s$.*

Proof. (i) Suppose that the function computed by α is $\psi_\alpha = g_1/g_2$, with $g_1, g_2 \in F[x]$, $g_2 \neq 0$, $\gcd(g_1, g_2) = 1$, and let $h = g_1 - g_2 f$. Since $A \subseteq R_\alpha$ and $\psi_\alpha(a) = f(a)$ for all $a \in A$, we have $h(a) = 0$ for $a \in A$, and

$$\prod_{a \in A} (x-a) \mid h.$$

If $h = 0$, then $\deg g_1 \geqq \deg f = b$. If $h \neq 0$, then either $\deg g_1 \geqq s$ or $\deg g_2 \geqq s \dot{-} b$. The claim follows from Fact 2.2.

(ii) For a vertex v of the graph of α, let $D(v)$ denote the depth of v. Thus an input or constant vertex v has $D(v) = 0$, and the output vertex v has $D(v) = d$. For $0 \leqq i \leqq d$, let L_i' consist of the vertices v with $D(v) = i$, and $L_i \subseteq L_i'$ consist of those v at which a division is performed. For $v \in L_i$, let $\eta(v)$ be the numerator of the rational function which is the divisor at v. Then $\deg \eta(v) \leqq 2^{i-1}$ for each $v \in L_i$. We may assume that every vertex is connected to the output vertex, so that

$$\# L_i \leqq \# L_i' \leqq 2^{d-i}, \qquad \sum_{v \in L_i} \deg \eta(v) \leqq 2^{i-1} \# L_i \leqq 2^{d-1}.$$

Since $A \cap R_\alpha = \varnothing$, we have

$$\prod_{a \in A} (x-a) \mid \prod_{\substack{1 \leqq i \leqq d \\ v \in L_i}} \eta(v), \qquad s = \# A \leqq \sum_{\substack{1 \leqq i \leqq d \\ v \in L_i}} \deg \eta(v) \leqq d 2^{d-1},$$

which implies (ii). □

In view of Fact 2.2, one might hope to improve the lower bound in (ii) to $\log s$. Somewhat surprisingly, this is not possible. The following is an example of an arithmetic circuit α over $\mathbb{C} \cup \{x\}$ of depth d and with $\# A \geqq d 2^{d-3}$, where $A = \mathbb{C} \setminus R_\alpha$.

Example 2.4. Let $d \in \mathbb{N}$, $n = 2^{d-1}$, $F = \mathbb{C}$, and $a_1, \cdots, a_n \in \mathbb{C}$ algebraically independent over $\mathbb{Q}$. Let α be a complete binary tree of depth d, with $x, a_1, x, a_2, \cdots, x, a_n$ at the leaves, and alternating levels of +- and /-nodes. For example, the following are among the functions computed at levels 1, 2 and 3, respectively:

$$x + a_1, \quad \frac{x+a_1}{x+a_2}, \quad \frac{x+a_1}{x+a_2} + \frac{x+a_3}{x+a_4}.$$

One can show that the new denominator polynomials introduced by division steps have no common zeros with the previous ones, and that the sum of their degrees is $d2^{d-3}$, which equals $\#(\mathbb{C} \setminus R_\alpha)$.

THEOREM 2.5. *Let $b \in \mathbb{N}$, $b \geqq 1$, F be a field, and $F^\times = F \setminus \{0\}$.*

(i) *If F is infinite, then*

$$d(\pi_F^b) = d^+(\pi_F^b) = \lceil \log b \rceil.$$

(ii) *If $A \subseteq F$ is finite with $s \geqq 2$ elements, then*

$$\min\left\{\log b, \log\left(\frac{s}{2} \dot{-} b + 1\right), \log s - \operatorname{loglog} s - 1\right\} \leqq d(\pi_A^b) \leqq d^+(\pi_A^b) \leqq \lceil \log b \rceil,$$

$$\min\{\log b, \log(s \dot{-} b + 1)\} \leqq d^+(\pi_A^b).$$

(iii) *If F is finite with q elements, $1 \leqq b < q$ and $m = \min\{\lceil \log b \rceil, \lceil \log(q - b) \rceil\}$, then*

$$\min\left\{\log b, \log\left(\frac{q}{2} \dot{-} b + 1\right), \log q - \operatorname{loglog} q - 1\right\} \leqq d(\pi_F^b) \leqq d^+(\pi_F^b) \leqq \lceil \log b \rceil,$$

$$m \leqq d^+(\pi_{F^\times}^b) \leqq m + 1.$$

If $q - b - 1$ is a power of two or $b \leqq q/2$, then $d^+(\pi_{F^\times}^b) = m$.

Proof. (i) follows from Facts 2.1(iii) and 2.2. For (ii), let α be an arithmetic circuit computing π_A^b, and $S = A \cap R_\alpha$. If $\#S \geqq s/2$, we find from Lemma 2.3(i)

$$D(\alpha) \geqq \min\left\{\log b, \log\left(\frac{s}{2} \dot{-} b + 1\right)\right\}.$$

If $\#S < s/2$, then Lemma 2.3(ii) implies that

$$D(\alpha) \geqq \log(\#(A \setminus S)) - \operatorname{loglog}(\#(A \setminus S)) \geqq \log\frac{s}{2} - \operatorname{loglog} s.$$

In (iii), we use Lemma 2.3(i) and Fact 2.1(iii) and (v). □

Now consider the following four problems, where $u \in F$, and $a, m \in F[x]$ of degree n are inputs, and $b \in \mathbb{N}$ with $2^{n-1} < b \leqq 2^n$.

POWER$_1$: Compute u^b.

POWER$_2$: Compute $a^b \bmod m$.

POWER$_3$: Compute $a^b \bmod x^n$.

POWER$_4$: Compute $a^b \bmod x^n$, if a has constant term 1.

The coefficient vectors of a and m form the input, and one obtains a coefficient vector as output; thus we now have to allow arithmetic circuits with many inputs and outputs.

COROLLARY 2.6. *Let $1 \leqq i \leqq 3$ and α be an arithmetic circuit computing* POWER$_i$ *everywhere.*

(i) *If F is infinite, then $D(\alpha) \geqq n$.*

(ii) *If F is finite with q elements and $1 \leqq b < q$, then $D(\alpha) \geqq \min\{n, \lceil \log(q - b) \rceil\}$.*

Proof. For POWER_1, this is Theorem 2.5(iii). By using input $a \in F$, any circuit for POWER_2 or POWER_3 also solves POWER_1. □

So much for the negative results that show that computing bth powers, where b is an n-bit integer, requires depth $\Omega(n)$ on arithmetic circuits. In only one case can we hope for shallow arithmetic circuits over F, namely when F is small, say $q = \# F \leqq n$, so that Corollary 2.6(ii) only gives a lower bound of order $\log n$. Subsequent to this paper, Fich and Tompa [6] gave fast parallel Boolean algorithms over nonprime large finite fields of small characteristic.

The basis for the fast parallel powering algorithms in the next sections is the following binomial expansion, which provides an algorithm for POWER_4 (where $a \equiv 1 \bmod x$):

$$a^b = (1+(a-1))^b = \sum_{0 \leqq i \leqq b} \binom{b}{i} (a-1)^i \equiv \sum_{0 \leqq i < n} \binom{b}{i} (a-1)^i \bmod x^n.$$

PROPOSITION 2.7. POWER_4 *can be computed by an arithmetic circuit of depth* $O(\log^2 n)$ *and size* $n^{O(1)}$.

Proof. The constants $\binom{b}{i}$ can be "hard-wired" into the circuit, and one multiplication of polynomials modulo x^n can be performed in depth $O(\log n)$. □

In characteristic zero, we could use the formula $a^b = \exp(b \log a)$ to obtain the same result.

The algorithms to be discussed below use the Boolean circuit families for elementary arithmetic. Although a priori the individual circuits (one for each relevant input length) of such a family are not required to be related to each other, it is convenient to know that a description of the circuit for input size parameter n can be constructed, say, by a (deterministic) Turing machine M on input n in unary. If M is polynomial-time bounded, then the family is P-uniform, and if the stronger requirement that M be logarithmic-space bounded holds, then the family is log-space uniform. Ruzzo [18] and Cook [4] discuss uniformity and parallel Boolean computations.

We avoid reference to the various notions of uniformity by using $\text{div}(n)$ to denote some function such that there exist Boolean circuits of depth $\text{div}(n)$ and size $n^{O(1)}$ that compute the division with remainder for n-bit integers. "Long division" yields the trivial bound $\text{div}(n) = O(\log^2 n)$. By Beame, Cook and Hoover [2], $\text{div}(n) = O(\log n)$ for P-uniform circuits, and by Reif [17], $\text{div}(n) = O(\log n \operatorname{loglog} n)$ for log-space uniform circuits. The "iterated" product of n n-bit integers can be computed in depth $O(\text{div}(n))$ [2].

Note that the upper bounds in Theorem 2.5 are achieved by log-space uniform families while the lower bounds make no uniformity assumption at all.

The depth bound of Proposition 2.7 can be improved by considering arithmetic networks, a more general model of computation that encompasses both arithmetic circuits and Boolean circuits. The interface between the two types of circuits is provided in the one direction by testing an arithmetic value $a \stackrel{?}{=} 0$ and producing a Boolean value. In the other direction, one can select one of two arithmetic values according to the value of a Boolean variable.

Reif [17] has shown that division with remainder of polynomials of degree at most n can be calculated on arithmetic networks of depth $O(\log n)$, provided the ground field F contains the roots of unity required for a Fast Fourier Transform. Eberly [5] generalized this to arbitrary fields, by constructing "log-space uniform"

families of arithmetic networks of depth $O(\log n \operatorname{loglog} n)$, and "$P$-uniform" families of depth $O(\log n)$ for division with remainder of polynomials. (These notions of uniformity for arithmetic networks are defined in von zur Gathen [9].) If we use $\operatorname{div}_F(n)$ to denote the maximum of the Boolean $\operatorname{div}(n)$ and these arithmetic depth functions over F, under both notions of uniformity, then also the iterated product of n polynomial with degree at most n can be computed in depth $O(\operatorname{div}_F(n))$. For all families considered, the size is polynomial in n.

The Boolean component of arithmetic networks allows us to solve a slightly stronger version of POWER_4: The binary representation of b can now be considered to be part of the input. In the algorithm, even the constants $\binom{b}{i}$ need not be hard-wired, but can be computed in depth $O(\operatorname{div}_F(n))$ using arithmetic in F. This is clear if $\operatorname{char} F = 0$: one simply computes the iterated products for numerator and denominator of the binomial coefficients, and then their quotient. If $\operatorname{char} F = p > 0$, we can use Lucas' [15] formula:

$$\binom{b}{i} \equiv \binom{b_0}{i_0}\binom{b_1}{i_1} \cdots \bmod p$$

where

$$b = b_0 + b_1 p + \cdots, \qquad i = i_0 + i_1 p + \cdots$$

are the p-ary expansions of b and i, with $0 \leqq b_j, i_j < p$. Given $j \leqq n$, and p, b, we compute $b_j = \lfloor b/p^j \rfloor - p\lfloor b/p^{j+1} \rfloor$ in depth $O(\operatorname{div}(n))$. The individual binomial coefficients can now be calculated as for characteristic zero; no division by zero occurs.

COROLLARY 2.8. POWER_4 *can be computed on arithmetic networks of depth* $O(\operatorname{div}_F(n))$. □

This statement comprises two results: log-space uniform families of arithmetic networks of depth $O(\log n \operatorname{loglog} n)$ for POWER_4, and P-uniform families of depth $O(\log n)$.

3. Integer powers modulo a prime power. In this section, we study the base case for computing large powers of integers modulo another integer, namely with the modulus being a prime power. In the next section we will use the Chinese remainder algorithm for a more general case.

Throughout this section, n is an input size parameter, a and b in $\mathbb{N}$ are n-bit numbers (i.e. $1 \leqq a, b < 2^n$), $p \leqq n$ is a prime, and $1 \leqq e \leqq n$. We want to compute $a^b \bmod p^e$. The fast parallel algorithm we get for the computation $\bmod p^e$ contrasts with the fact that by Theorem 2.5 no arithmetic circuits over F of poly-logarithmic depth exist for the same computation over a field F with p^e elements. (However, Fich and Tompa [6] present an arithmetic network over the prime field $\mathbb{Z}_p \subseteq F$ of depth $O(\log^2 n)$ that computes a^b.)

Let us first consider the case $p \geqq 3$. We use the well-known decomposition $\mathbb{Z}_{p^e}^{\times} = G \times H$ (see e.g. [11, Chap. 4]), with the injective group homomorphism

$$\alpha : \mathbb{Z}_p^{\times} \to \mathbb{Z}_{p^e}^{\times}, \qquad r \bmod p \mapsto r^{p^{e-1}} \bmod p^e,$$

$$G = \operatorname{im} \alpha, \qquad H = \{r \bmod p^e : r \equiv 1 \bmod p\}.$$

Thus $G \cong \mathbb{Z}_p^{\times}$, $\# G = p - 1$, and $\# H = p^{e-1}$. We will split the problem of computing powers in $\mathbb{Z}_{p^e}^{\times}$ into the corresponding problems in G and H. Our first task is to compute α.

LEMMA 3.1. *Given r with $1 \leqq r < p \leqq n$ and $p \geqq 3$ prime, one can compute s such that*

$$s \equiv \alpha(r) \equiv r^{p^{e-1}} \bmod p^e$$

by a Boolean circuit of depth $O(\log n \cdot \operatorname{div}(n))$ and size $n^{O(1)}$.

Proof. Since $\# \mathbb{Z}_{p^e}^{\times} = (p-1)p^{e-1}$, we have

$$s^{p-1} \equiv 1 \bmod p^e \text{ and } s \equiv r \bmod p$$

for $s = r^{p^{e-1}}$. A quadratic Newton method for solving the first equation, with the second equation providing an initial value, would yield the following recursion:

$$\begin{aligned} &s'_0 = r, \\ &s'_i = s'_{i-1} - \frac{(s'_{i-1})^{p-1} - 1}{(p-1)(s'_{i-1})^{p-2}} \bmod p^{2^i}, \\ &1 \leqq s'_i < p^{2^i}. \end{aligned}$$

In order to avoid divisions, we consider the following division-free variant of this Newton iteration:

$$\begin{aligned} &s_0 = r, \\ &s_i \equiv s_{i-1} + (s_{i-1}^{p-1} - 1)(1 + p + p^2 + \cdots + p^{2^i-1}) s_{i-1}^{(p-2)^2} \bmod p^{2^i}, \\ &1 \leqq s_i < p^{2^i}. \end{aligned}$$

For completeness, we check that the formula is correct, i.e., that

$$s_i^{p-1} \equiv 1 \bmod p^{2^i},$$

by induction on i. The case $i = 0$ being clear, we set $t = s_{i-1}^{p-1}$, so that

$$\begin{aligned} s_i^{p-1} - 1 &\equiv t + (p-1) s_{i-1}^{p-2} (t-1)(1 + \cdots + p^{2^i-1}) s_{i-1}^{(p-2)^2} - 1 \\ &= (t-1)(1 + (-1 + p^{2^i}) t^{p-2}) \\ &\equiv (t-1)(1 - t^{p-2}) = (t-1)^2(-t^{p-3} - \cdots - 1) \equiv 0 \bmod p^{2^i}. \end{aligned}$$

Set $l = \lceil \log e \rceil$. Then

$$s_l^{p-1} \equiv 1 \bmod p^e,$$

and the uniqueness of Newton iteration guarantees that

$$\alpha(r) \equiv s_l \bmod p^e.$$

The computation of s_i from s_{i-1} can be done in depth $O(\operatorname{div}(n))$, and the total depth of the resulting circuit is $O(\log n \operatorname{div}(n))$. □

Algorithm INTEGER POWERS MODULO A PRIME POWER.

Input: An input size parameter n, integers $a, b, p, e \in \mathbb{N}$ such that $p, e \leqq n$; $a, b < 2^n$, and $p \geqq 3$ is prime.

Output: $c \in \mathbb{N}$ such that $a^b \equiv c \bmod p^e$.

1. If $a = 0$, then return $c = 0$ and stop. Compute $l \in \mathbb{N}$ such that $p^l \nmid a$ and $p^{l+1} \nmid a$. If $lb \geqq e$, return $c = 0$ and stop. Otherwise replace a by a/p^l. [We can now assume $a \not\equiv 0 \bmod p$.]
2. Compute r such that

$$a \equiv r \bmod p, \qquad 1 \leqq r < p.$$

3. Compute s such that

$$s \equiv \alpha(r) \equiv r^{p^{e-1}} \bmod p^e, \qquad 1 \leqq s < p^e.$$

4. Compute h and u such that

$$h \equiv b \bmod p-1, \qquad 0 \leqq h < p-1,$$
$$u \equiv s^h \bmod p^e, \qquad 1 \leqq u < p^e.$$

5. Compute the inverse t of s such that

$$st \equiv 1 \bmod p^e, \qquad 1 \leqq t < p^e.$$

6. Compute v and w such that

$$v \equiv at \bmod p^e,$$
$$w \equiv \sum_{0 \leqq i < e} \binom{b}{i} (v-1)^i \bmod p^e,$$
$$1 \leqq v, w < p^e.$$

7. Return $c = p^{lb} uw$.

THEOREM 3.2. *Algorithm* INTEGER POWERS MODULO A PRIME POWER *works correctly as described. For input size parameter n, it can be implemented on a Boolean circuit of depth $O(\log n \operatorname{div}(n))$ and size $n^{O(1)}$.*

Proof. We can assume that $a \not\equiv 0 \bmod p$, i.e., $l = 0$. Note that $s \equiv r \equiv a \bmod p$ and $v \equiv 1 \bmod p$, and therefore

$$v^b = (1+(v-1))^b \equiv \sum_{0 \leqq i < e} \binom{b}{i} (v-1)^i \equiv w \bmod p^e.$$

Since $\# \mathbb{Z}_p^\times = p-1$, we also have

$$uw \equiv s^h v^b \equiv \alpha(r)^h v^b \equiv \alpha(r^h) v^b \equiv \alpha(r^b) v^b \equiv \alpha(r)^b v^b \equiv a^b \bmod p^e.$$

Thus the algorithm works correctly. We use a quadratic Newton iteration to compute t in step 5:

$$t_0 = r, \quad t_i \equiv t_{i-1} - (-t_{i-1} + t_{i-1}^2 s) \bmod p^{2^i}, \quad 1 \leqq t_i < p^{2^i}.$$

Each iteration step can be performed in depth $O(\operatorname{div}(n))$. This depth is also sufficient for the iterated products required for the binomial coefficients (see end of § 2) and powers of $v-1$ in step 6. Thus the depths are:

steps 1, 2, 4, 6, 7: $O(\operatorname{div}(n))$,

steps 3, 5: $O(\log n \operatorname{div}(n))$. □

By exploiting the power of P-uniformity, we can actually get Boolean circuits of optimal (up to constant factors) depth.

COROLLARY 3.3. *There exists a P-uniform family of Boolean circuits of depth $O(\log n)$, whose nth circuit computes $a^b \bmod p^e$ for all $a, b, p, e \in \mathbb{N}$ with $a, b < 2^n$; $e, p \leqq n$, and $p \geqq 3$ prime.*

Proof. Since $\operatorname{div}(n) = O(\log n)$ for P-uniformity, it is sufficient to implement steps 3 and 5 of INTEGER POWERS MODULO A PRIME POWER in depth $O(\log n)$. In the circuit for input size parameter n, we hard-wire for every (r, p, e) with $1 \leqq r < p \leqq n$, $e \leqq n$, and p prime, a table of constants s_{rpe} and t_{rpe} such that

$$s_{rpe} \equiv r^{p^{e-1}} \bmod p^e, \qquad t_{rpe} \cdot s_{rpe} \equiv 1 \bmod p^e, \quad 1 \leqq s_{rpe}, t_{rpe} < p^e.$$

These are less than n^3 constants, and they can be computed in (sequential) polynomial time. Then steps 3 and 5 can be replaced by a table look-up in depth $O(\log n)$. □

Remark 3.4. For $r \equiv a \not\equiv 0 \bmod p$, we have

$$a^b \equiv r^{b-e+1} \sum_{0 \leqq i < e} \binom{b}{i} r^{e-i-1} (a-r)^i \bmod p^e.$$

By hard-wiring r^{2^j} for all $r < p$ and $0 \leqq j < n$, this formula yields another P-uniform family of Boolean circuits for our powering problem, with depth $O(\log n)$. It also works when p is 2 or not prime, so that Corollary 3.3 also holds for $p = 2$. For log-space uniformity, we obtain the following improvement.

COROLLARY 3.5. *For $p = 2$ and $p = 3$, there exist log-space uniform families of Boolean circuits of depth $O(\log n \operatorname{loglog} n)$ and size $n^{O(1)}$ that compute $a^b \bmod p^e$ for all $a, b < 2^n$ and $e \leqq n$.*

Proof. For $p = 2$, we have $r = s = t = 1$ in the algorithm. For $p = 3$, we use the residue system $\{1, -1\}$ for $\mathbb{Z}_3^\times$, so that $r = s = t \in \{1, -1\}$. □

In the first version of Reif [17], it was conjectured that no Boolean circuits of depth $(\log n)\, O(1)$ exist to compute $a^b \bmod 2^n$; this conjecture is now disproven.

4. Chinese remaindering. In order to generalize the fast parallel algorithm for powering with prime power moduli p^e to more general moduli, we use, of course, a Chinese remainder algorithm. It turns out that we first have to present Boolean circuits for some other problems, which may be of independent interest. Inputs are the various n-bit numbers denoted as $a, b, a_1, \cdots, a_k, m, m_1, \cdots, m_k$ with $k \leqq n$, and output an n-bit number c as below:

POWER: $c \equiv a^b \bmod m$.

GCD: $c = \gcd(a, m) \geqq 1$.

MODINV:
$$\begin{cases} ac \equiv 1 \bmod m & \text{if } \gcd(a, m) = 1, \\ c = 0 & \text{otherwise.} \end{cases}$$
("modular inverse")

LAGRANGE: $c \equiv 0 \bmod m_2 \cdots m_k$,
$$c \equiv \begin{cases} 1 \bmod m_1 & \text{if } \gcd(m_1, m_2 \cdots m_k) = 1, \\ 0 \bmod m_1 & \text{otherwise.} \end{cases}$$
("Lagrange interpolation coefficient"),

CHINREM: If $\gcd(m_i, m_j) = 1$ for all $i \neq j$,
then $c \equiv a_i \bmod m_i$ for all i.
Otherwise $c = 0$.
("Chinese remainder algorithm")

FACTOR: compute the prime factors of m (with multiplicities).

Using DIV for the problem of integer division with remainder, we have NC^1-reductions (see Cook [4] for terminology):

$$\text{CHINREM} \leqq \text{LAGRANGE} + \text{GCD} \text{ and } \text{LAGRANGE} \leqq \text{MODINV} + \text{DIV}.$$

The second reduction follows by using an inverse d of $m_2 \cdots m_k \bmod m_1$; then $c = d \cdot (m_2 \cdots m_k)$ is sufficient. The iterated product $m_2 \cdots m_k$ is reducible to division (Beame, Cook and Hoover [2]).

At the present time, none of these problems is known to be in NC—i.e., solvable in depth $(\log n)^{O(1)}$—or log-space complete for P. All problems, except possibly FACTOR, are in P.

We need fast parallel algorithms for these problems in special cases. We use "SMOOTH" to denote the condition that the moduli $m, m_1, \cdots, m_k$ are n-smooth, i.e., that $p \leqq n$ for every prime factor p of m or m_i. Note that the condition depends on the parameter n; where this parameter is clear, we will not mention it explicitly. The results would also hold if we used $p \leqq n^c$ for some fixed c. The number of smooth integers is a well-studied topic in analytic number theory (see e.g. Hildebrand [12]).

THEOREM 4.1. *The following problems can be computed by Boolean circuits of polynomial size and the stated depth*:

(i) SMOOTH-FACTOR *and* SMOOTH-GCD *in depth* $O(\operatorname{div}(n))$,

(ii) SMOOTH-MODINV, SMOOTH-LAGRANGE, *and* SMOOTH-CHINREM *in depth* $O(\log n \cdot \operatorname{div}(n))$.

Proof. We leave away the "SMOOTH-". First consider the trivial algorithm for FACTOR: For each number $p \leqq n$, test whether p is prime (by a sieve method) and determine its multiplicity in m. This can be performed in depth $\operatorname{div}(n)$, by computing $p, p^2, p^3, \cdots, p^n$ and testing for each i whether p^i divides m. Now GCD is trivial.

Now consider MODINV in the special case where $m = p^e$, p prime. We have the following algorithm:

1. Test if $p|a$, and return $c = 0$ if "yes."
2. Find by exhaustive search $c_0 \in \mathbb{N}$ such that
$$1 \leqq c_0 < p, \qquad ac_0 \equiv 1 \bmod p.$$
3. Compute $c \equiv a^{-1} \bmod p^e$ by a quadratic Newton iteration as in step 5 of INTEGER POWERS MODULO A PRIME POWER.

The circuit depth for this algorithm is dominated by step 3:
$$O(\log e \cdot \operatorname{div}(n)) = O(\log n \operatorname{div}(n)).$$

Next we consider LAGRANGE. If $m_1 = p^e$ for a small prime p, then we can solve LAGRANGE in depth $O(\log n \operatorname{div}(n))$ by the reduction to MODINV+DIV. For the general case $m_1 = p_1^{e_1} \cdots p_r^{e_r}$, with small pairwise distinct primes $p_1, \cdots, p_r$ (and no p_i dividing $m_2 \cdots m_k$), we compute $c_1, \cdots, c_r$ such that
$$c_i \equiv \begin{cases} 1 \bmod p_{p_i}^{e_i}, \\ 0 \bmod p_{p_1}^{e_1} \cdots p_{i-1}^{e_{i-1}} p_{i+1}^{e_{i+1}} \cdots p_r^{e_r} m_2 \cdots m_k. \end{cases}$$
The power products can be computed in depth $O(\operatorname{div}(n))$. Each c_i is given by one of the special LAGRANGE problems already solved, and
$$c = c_1 + \cdots + c_r$$
solves the current problem. Therefore LAGRANGE and CHINREM can be solved in depth $O(\log n \operatorname{div}(n))$.

Now for the general case of MODINV, where $m = p_1^{e_1} \cdots p_r^{e_r}$ with small primes p_i, we have the algorithm:

1. For all i, $1 \leqq i \leqq r$, compute c_i such that $ac_i \equiv 1 \bmod p_i^{e_i}$,
2. Compute c such that for all $i \leqq r$, $c \equiv c_i \bmod p_i^{e_i}$.

Then
$$\forall i \leqq r \quad ac \equiv ac_i \equiv 1 \bmod p_i^{e_i} \Rightarrow ac \equiv 1 \bmod m. \qquad \square$$

We now have a fast parallel algorithm for powering.

Algorithm INTEGER POWERS.
Input: an integer n (the input size parameter), and n-bit integers a, b, m, where $b \in \mathbb{N}$, and $m \in \mathbb{N}$ is n-smooth.
Output: $c \in \mathbb{Z}$ such that $c \equiv a^b \bmod m$.
1. Call FACTOR with input m to compute $p_1, \cdots, p_r, e_1, \cdots, e_r$ such that $m = p_1^{e_1} \cdots p_r^{e_r}$ is the prime factorization of m.
2. For all i, $1 \leqq i \leqq r$, call algorithm INTEGER POWERS MODULO A PRIME POWER with input $n, a, b, p = p_i$, and $e = e_i$ to obtain output $c_i \in \mathbb{Z}$. [Then $c_i \equiv a^b \bmod p_i^{e_i}$.]
3. Call CHINREM with input $a_1, \cdots, a_r, m_1, \cdots, m_r$, where $m_i = p_i^{e_i}$, $a_i \equiv c_i \bmod m_i$, $0 \leqq a_i < m_i$, and return the output c.

THEOREM 4.2. *Algorithm* INTEGER POWERS *can be performed by a family of Boolean circuits of depth* $O(\log n \operatorname{div}(n))$ *and size* $n^{O(1)}$.

Proof. Correctness of the algorithm is clear. By Theorem 4.1, steps 1 and 3 can be performed in depth $O(\log n \operatorname{div}(n))$ and size $n^{O(1)}$, and similarly for step 2 by Theorem 3.2. □

Note that on input of arbitrary n-bit numbers a, b, m, step 1 can be used to determine whether m is n-smooth.

COROLLARY 4.3. *There exists a P-uniform family of Boolean circuits of depth* $O(\log n)$, *whose nth circuit computes* $a^b \bmod m$ *for all n-bit integers a, b, m, where m is n-smooth.*

Proof. In view of Theorem 4.1(i) and Corollary 3.3 it is sufficient to implement step 3 of algorithm INTEGER POWERS in depth $O(\log n)$. For any p, e, q with $1 \leqq p, e, q \leqq n$ and $p \neq q$ prime, we hard-wire the constant u_{peq} such that

$$u_{peq} \cdot q \equiv 1 \bmod p^e, \qquad 1 \leqq u_{peq} < p^e,$$

in the circuit for input size parameter n. This is a table of less than n^3 constants, and any u_{peq} can be looked up in depth $O(\log n)$. At a call of CHINREM with moduli $m_1 = p_1^{e_1}, \cdots, m_r = p_r^{e_r}$, with $2 \leqq p_i \leqq n$, we compute the inverses

$$v_{ij} = u^{e_j}_{p_i, e_i, p_j},$$

for $i \neq j$, so that

$$v_{ij} p_j^{e_j} \equiv 1 \bmod p_i^{e_i}.$$

This can be done in depth $O(\log n)$, and using the reduction CHINREM $\leqq$ MODINV + DIV, step 3 can be performed in depth $O(\log n)$. □

5. Polynomial powers modulo a prime power. In this and the next section, we translate the development of §§ 3 and 4 for integers into the setting of polynomials over a finite field F. The main result is a Boolean circuit of depth $O(\log^2 n \operatorname{loglog} n)$ that computes $a^b \bmod f^e$, where $a, f \in F[x]$, a has degree at most n, f is irreducible and such that $(\# F)^{\deg f} \leqq n$, b is an n-bit number, and $e \leqq n$. We also get an arithmetic circuit over F of the same depth. The above circuits are log-space uniform; we also obtain P-uniform circuit families of optimal depth $O(\log n)$.

So let F be a finite field of characteristic p and with $q = p^l$ elements. We assume that F is represented as (p, g), where $g \in \mathbb{Z}_p[t]$ is an irreducible polynomial of degree l such that $F = \mathbb{Z}_p[t]/(g)$. Elements of F are represented by polynomials from $\mathbb{Z}_p[t]$ of degree less than l, and an element of $\mathbb{Z}_p$ is given by the binary representation of an integer between 0 and $p-1$.

Now let $f \in F[x]$ be monic irreducible of degree d, $e \geqq 1$, and $R = F[x]/(f^e)$. We first consider a decomposition $R^\times = G \times H$ of the group of units of R, analogous to that for $\mathbb{Z}_{p^e}^\times$. Note that $\# R = q^{de}$ and $\# R^\times = (q^d - 1)q^{d(e-1)}$, so that we expect the "small" group G to have $q^d - 1$ elements, and the "large" H to have $q^{d(e-1)}$ elements. We consider the field $K = F[x]/(f)$, and the mapping

$$\alpha : K^\times \to R^\times, \qquad r \bmod f \mapsto r^{q^{d(e-1)}} \bmod f^e.$$

We abbreviate the exponent as $\varepsilon = q^{d(e-1)}$. If $r \in F[x]$ and $r \not\equiv 0 \bmod f$, then r^ε is a unit mod f^e. α is well defined, since $r^\varepsilon - s^\varepsilon = (r-s)^\varepsilon$ for every $r, s \in F[x]$, and $e \leqq 2^{e-1} \leqq 2^{d(e-1)} \leqq q^{d(e-1)} = \varepsilon$; thus $r \equiv s \bmod f$ implies that $r^\varepsilon \equiv s^\varepsilon \bmod f^e$. Obviously, α is a (multiplicative) group homomorphism. (It is also additive where defined.) Since

$$r^\varepsilon \equiv r \bmod f$$

for all $r \in F[x]$, α is injective. We now let

$$G = \operatorname{im} \alpha, \qquad H = \{g \bmod f^e : g \equiv 1 \bmod f\}.$$

These are two subgroups of $R^\times$, with $\# G = q^d - 1$ and $\# H = q^{d(e-1)}$. Since $G \cap H = \{1\}$, we have the desired decomposition $R^\times = G \times H$. The two groups can also be characterized as

$$G = \{u \in R^\times : u^{q^d - 1} = 1\}, \qquad H = \{u \in R^\times : u^{q^{d(e-1)}} = 1\}.$$

Just as in Lemma 3.1, we first describe a fast parallel computation of α. We again denote by n the input size parameter, and assume that $q^d \leqq n$, $de \leqq n$. The second constraint is quite natural, since the usual representatives for R have degree up to $de - 1$. We assume that there exist Boolean circuits of depth $\operatorname{div}(n)$ and size $n^{O(1)}$ that compute the division with remainder, iterated product, and inverse mod x^n for polynomials of degree at most n. Then $\operatorname{div}(n) = O(\log n)$ for P-uniform circuits, and $\operatorname{div}(n) = O(\log n \operatorname{loglog} n)$ for log-space uniform circuits, by Eberly [5]. As of now, no Boolean circuits of depth $(\log n)^{O(1)}$ are known to perform arithmetic in $\mathbb{Z}_p$ using some standard representatives if p has about n bits; inversion mod p is the problem. This can be avoided by using a redundant representation u/v for elements of $\mathbb{Z}_p$, with $u, v \in \mathbb{Z}$, $0 \leqq u, v < p$ and $v \neq 0$; the upper bounds quoted above hold for this representation. But since in our case $p \leqq n$, we can actually find the inverse of $v \bmod p$ by exhaustive search in depth $O(\log n)$.

LEMMA 5.1. *Given $r \in F[x]$ with $0 \leqq \deg r < \deg f$, one can compute $s \in F[x]$ such that*

$$s \equiv r^{q^{d(e-1)}} \bmod f^e$$

by Boolean circuits of depth $O(\log n \operatorname{div}(n))$ and size $n^{O(1)}$.

Proof. We solve the equation

$$s^{q^d - 1} \equiv 1 \bmod f^e, \qquad s \equiv r \bmod f,$$

by a quadratic Newton iteration. The depth of the iteration is $\lceil \log e \rceil$. Each step

$$s_i \equiv s_{i-1} + (s_{i-1}^{q^d - 1} - 1) s_{i-1}^{(q^d - 2)^2} \bmod f^{2^i}$$

can be performed in depth $O(\operatorname{div}(n))$. Comparing with Lemma 3.1, one factor $q^d - 1 = -1$ in the denominator becomes trivial. □

We use P-POWER to denote the polynomial analogue of POWER$_1$: with input size parameter $n \in \mathbb{N}$, we are given p, g, a, b, m such that $p, b \in \mathbb{N}$, p is prime, $b < 2^n$, $g \in \mathbb{Z}_p[t]$ is irreducible of degree $l \leqq n$, $a, m \in \mathbb{Z}_p[t, x]$, $\deg_t(a), \deg_t(m) < l$, $\deg_x(a)$,

$\deg_x(m) \leqq n$, and have to compute some $c \in \mathbb{Z}_p[t, x]$ such that $c \equiv a^b \bmod m$ in $F[x]$, where $F = \mathbb{Z}_p[t]/(g)$. Let $q = p^l$. We say that m is n-smooth if there exist $f_1, \cdots, f_r \in F[x]$ irreducible of degree at most d, and $e_1, \cdots, e_r \in \mathbb{N}$ such that $f = f_1^{e_1} \cdots f_r^{e_r}$ and $q^d \leqq n$.

The following is the translation of algorithm INTEGER POWERS MODULO A PRIME POWER to the setting of polynomials.

Algorithm POLYNOMIAL POWERS MODULO A POWER OF AN IRREDUCIBLE.

Input: An input size parameter $n \in \mathbb{N}$, and p, g, a, b, m as in P-POWER, where $m = f^e$ is a power of an irreducible $f \in F[x]$.

Output: $c \in F[x]$ such that $c \equiv a^b \bmod f^e$.

1. If $a = 0$, then return $c = 0$ and stop. Compute $l \in \mathbb{N}$ such that f^l / a and $f^{l+1} \nmid a$. If $lb \geqq e$, return $c = 0$ and stop. Otherwise replace a by a/f^l. (We can now assume $a \not\equiv 0 \bmod f$.)
2. Set $d = \deg f$, and compute $r \in F[x]$ such that
$$a \equiv r \bmod f, \qquad 0 \leqq \deg r < d.$$
3. Compute $s \in F[x]$ such that
$$s \equiv \alpha(r \bmod f) \equiv r^{q^{d(e-1)}} \bmod f^e,$$
$$0 \leqq \deg s < de.$$
4. Compute $h \in \mathbb{N}$ and $u \in F[x]$ such that
$$h \equiv b \bmod q^d - 1, \qquad 0 \leqq h < q^d - 1,$$
$$u \equiv s^h \bmod f^e, \qquad 0 \leqq \deg u < de.$$
5. Compute the inverse $t \in F[x]$ of s such that
$$st \equiv 1 \bmod f^e, \qquad 0 \leqq \deg t < de.$$
6. Compute $v, w \in F[x]$ such that
$$v \equiv at \bmod f^e,$$
$$w \equiv \sum_{0 \leqq i < e} \binom{b}{i} (v-1)^i \bmod f^e,$$
$$0 \leqq \deg v, \deg w < de.$$
7. Return $c = f^{lb} uw$.

THEOREM 5.2. *The above algorithm solves* P-POWER. *If m is a power of an irreducible polynomial f with $q^{\deg f} \leqq n$, it can be implemented on a Boolean circuit of depth $O(\log n \operatorname{div} (n))$ and size $n^{O(1)}$.*

Proof. Everything follows just as for Theorem 3.2. The inverse t in step 5 is calculated by a quadratic Newton iteration. The initial value t_0 such that $st_0 \equiv 1 \bmod f$ can be found by exhaustive search in depth $O(\log n)$, since $q^d \leqq n$; alternatively, one can solve the corresponding $d \times d$-system of linear equations over F, in depth $O(\log^2 d \operatorname{div} (n))$ or $O((\operatorname{loglog} n)^2 \operatorname{div} (n))$. All steps except steps 3 and 5 require only depth $O(\operatorname{div} (n))$. □

COROLLARY 5.3. *There exists a P-uniform family of Boolean circuits of depth $O(\log n)$ that computes* P-POWER *for inputs with m a power of a small irreducible polynomial.*

Proof. Given n, for every p, g, f, e as in P-POWER with input size parameter n and $p^{l \deg f} \leqq n$, and for every polynomial $r \in (\mathbb{Z}_p[t]/(g))[x] \setminus \{0\}$ of degree less than d, let $s_{rfeg}, t_{rfeg} \in (\mathbb{Z}_p[t]/(g))[x]$ be the unique polynomials such that

$$s_{rfeg} \equiv r^{p^{ld(e-1)}} \bmod f^e, \quad s_{rfeg} t_{rfeg} \equiv 1 \bmod f^e, \quad 0 \leqq \deg s_{rfeg}, \deg t_{rfeg} < de.$$

There are less than $n^5 \log n$ possible choices of (p, l, g, d, f, e, r), and therefore all s_{rfeg} and t_{rfeg} can be hard-wired into the nth circuit of a P-uniform circuit. Given these constants, steps 3 and 5 and therefore the whole of algorithm of POLYNOMIAL POWERS MODULO A POWER OF AN IRREDUCIBLE can be performed in depth $O(\log n)$. □

The following is an improvement analogous to Corollary 3.5.

COROLLARY 5.4. *There exists a log-space uniform family of Boolean circuits of depth* $O(\log n \operatorname{loglog} n)$ *and polynomial size, which computes* P-POWER *for all* $m = f^e$ *with* f *linear and* $p^l \leqq n$.

Proof. On input $a, b, f = x - z, e$ with $z \in F$, we can assume that $a(z) \neq 0$, and then the algorithm simplifies to

$$c \equiv a(z)^b \sum_{0 \leqq i < e} \binom{b}{i} \left(\frac{a}{a(z)} - 1\right)^i$$

$$\equiv a(z)^{b-e+1} \sum_{0 \leqq i < e} \binom{b}{i} a(z)^{e-i-1} (a - a(z))^i \bmod f^e. \qquad \square$$

For arithmetic circuits, the field F and the exponents b and e have to be fixed. The appropriate modification of the algorithm above yields the following result.

COROLLARY 5.5. *For any finite field* F, *any* $b < 2^n$ *and* $e \leqq n$ *there exists an arithmetic circuit over* F *of depth* $O(\log^2 n \operatorname{loglog} n)$ *and polynomial size that computes* $a^b \bmod f^e$ *for all inputs* $a, f \in F[x]$ *of degree at most* n *such that* f *is irreducible and* $(\# F)^{\deg f} \leqq n$.

This algorithm can be implemented on a log-space uniform family of arithmetic circuits. For P-uniform families of arithmetic networks, one obtains optimal depth $O(\log n)$.

6. Polynomial powers over finite fields. In this section, we make a rather obvious adaptation of the algorithm from § 5 to more general moduli, via Chinese remaindering. A fast parallel powering algorithm results, for smooth moduli. In particular, we obtain a P-uniform family of Boolean circuits of optimal depth $O(\log n)$.

As usual, F is a finite field with $\operatorname{char} F = p$ and $\# F = q = p^l$, and n is an input size parameter. We consider problems P-CHINREM, P-FACTOR defined by the same conditions as the corresponding problems in § 4, except that now $a_1, \cdots, a_k, m_1, \cdots, m_k, c \in F[x]$ are polynomials of degree at most n; b is an n-bit integer. Polynomials are represented by their coefficient sequences.

For any field F, P-CHINREM can be solved by arithmetic networks over F of depth $O(\log^2 n)$ (von zur Gathen [8]). For finite F, this can also be performed by Boolean circuits of depth $O(\log^2 n)$, using redundant notation for elements of $\mathbb{Z}_p$. (See Borodin, Cook and Pippenger [3] for Boolean circuits for linear algebra; also Eberly [5].) Clearly P-FACTOR can be computed on Boolean circuits of depth $O(\operatorname{div}(n))$, if m is n-smooth.

THEOREM 6.1. P-POWER *can be computed on Boolean circuits of depth* $O(\log n \operatorname{div}(n))$ *and size* $n^{O(1)}$, *for inputs with* m *being* n-*smooth.*

Proof. In order to compute $a^b \bmod m$ for $a, m \in F[x]$, we first factor $m = f_1^{e_1} \cdots f_r^{e_r}$ where $f_1, \cdots, f_r$ are the irreducible factors of m. By Theorem 5.2, we can compute

$c_i \in F[x]$ such that $c_i \equiv a^b \bmod f_i^{e_i}$ for each i. Then we interpolate $c_1, \cdots, c_r$ by P-CHINREM to get the desired $c \in F[x]$ with $c \equiv a^b \bmod m$. □

COROLLARY 6.2. *There exists a P-uniform family of Boolean circuits of depth* $O(\log n)$, *whose nth circuit computes P-*POWER *if m is n-smooth.*

Proof. By Corollary 5.3 and the fact that P-FACTOR can now be solved in depth $O(\log n)$, it is sufficient to consider P-CHINREM. As for Corollary 4.3, we simply hard-wire the required modular inverses $u_{fef'} \in (\mathbb{Z}_p[t]/(g))[x]$ such that

$$u_{fef'} \cdot f' \equiv 1 \bmod f^e, \qquad 0 \leqq \deg u_{fef'} < e \deg f,$$

where p is prime, $g \in \mathbb{Z}_p[t]$ is monic irreducible of degree l, $f \neq f' \in (\mathbb{Z}_p[t]/(g))[x]$ are irreducible monic of degree at most d, $p^{ld} \leqq n$, and $1 \leqq e \leqq n$. This yields a table of polynomial size, which can be constructed in polynomial time. □

7. Conclusion. We have shown that large powers of integers or polynomials modulo an element can be computed fast in parallel, provided the modulus is smooth, i.e., has only small prime factors. The basic question, however, remains open:

Open Question 7.1. Can $a^b \bmod m$, for n-bit integers a, b, m, be computed on Boolean circuits of depth $(\log n)^{O(1)}$ (and size $n^{O(1)}$)?

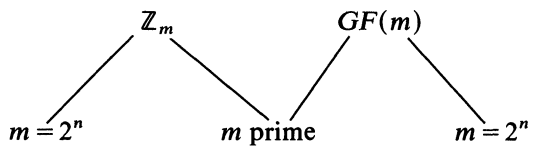
One can hope for further positive results in special cases; a candidate seems to be the case where m is prime and $m-1$ is smooth. No successful algorithm exists that makes use only of arithmetic mod m for prime m (Theorem 2.5).

The number a^b is too long to be computed even in polynomial time, but Allan Borodin has asked the following.

Open Question 7.2. Given n-bit integers a, b, i, can the ith bit of a^b be computed in depth $(\log n)^{O(1)}$ (and size $n^{O(1)}$)?

It is not even clear how to do this in (sequential) polynomial size. The corresponding questions can be asked for polynomials over finite fields. Fich and Tompa [6] give positive answers to both questions for polynomials over finite fields of small characteristic. For computations by arithmetic circuits over large finite fields, the answer is negative to both questions.

The two most basic finite arithmetic structures with m elements (where m is a "large" integer) are $\mathbb{Z}_m$ and $GF(m)$. The extreme cases are when m is a power of 2 (or a power of a small prime, or smooth for $\mathbb{Z}_m$), and when m is prime, in which case $\mathbb{Z}_m = GF(m)$.



The parallel powering problem is solved in both cases with $m = 2^n$.

REFERENCES

[1] H. ALT, *Comparison of arithmetic functions with respect to Boolean circuit depth*, Proceedings 16th Annual ACM Symposium on the Theory of Computing, Washington, DC, 1984, pp. 466–470.

[2] P. W. BEAME, S. A. COOK AND H. J. HOOVER, *Log depth circuits for division and related problems*, this Journal, 15 (1986), pp. 994–1003.

[3] A. BORODIN, S. COOK AND N. PIPPENGER, *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Inform. and Control, 58 (1983), pp. 113–136.

[4] S. A. COOK, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control, 64 (1985), pp. 2–22.

[5] W. EBERLY, *Very fast parallel matrix and polynomial arithmetic*, Proceedings 25th Annual IEEE Symposium on the Foundations of Computer Science, Singer Island, FL, 1984, pp. 21-30.

[6] F. FICH AND M. TOMPA, *The parallel complexity of exponentiating polynomials over finite fields*, Proceedings 17th Annual ACM Symposium on the Theory of Computing, Providence, RI, 1985, pp. 38-47; J. Assoc. Comput. Mach., to appear.

[7] J. VON ZUR GATHEN, *Parallel algorithms for algebraic problems*, this Journal, 13 (1984), pp. 802-824.

[8] ———, *Representations and parallel computations for rational functions*, this Journal, 15 (1986), pp. 432-452.

[9] ———, *Parallel arithmetic computations: a survey*, Proceedings 12th Internat. Symposium on the Math. Foundations of Computer Science, Bratislava, Springer Lecture Notes in Computer Science, 233, 1986, pp. 93-112.

[10] J. VON ZUR GATHEN AND G. SEROUSSI, *Boolean circuits versus arithmetic circuits*, Proceedings 6th Internat. Conf. Computer Science, Santiago, Chile, 1986, pp. 171-184.

[11] H. HASSE, *Number Theory*, Grundlehren der math. Wiss., 229, Springer-Verlag, New York-Berlin-Heidelberg, 1980.

[12] A. HILDEBRAND, *On the number of positive integers $\leqq x$ and free of prime factors $> y$*, J. Number Theory, 22 (1986), pp. 289-307.

[13] D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[14] H. T. KUNG, *New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences*, J. Assoc. Comput. Mach., 23 (1976), pp. 252-261.

[15] E. LUCAS, *Sur les congruences des nombres eulériens et des coefficients différentiels des fonctions trigonométriques, suivant un module premier*, Bull. Soc. Math. France, 6 (1877/78), pp. 49-54.

[16] I. MUNRO AND M. PATERSON, *Optimal algorithms for parallel polynomial evaluation*, J. Comput. System Sci., 7 (1973), pp. 189-198.

[17] J. REIF, *Logarithmic depth circuits for algebraic functions*, this Journal, 15 (1986), pp. 231-242. Extended Abstract in Proceedings 24th Annual IEEE Symposium on the Foundations of Computer Science, Tucson, AZ, 1983, pp. 138-145.

[18] W. L. RUZZO, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365-383.

[19] V. STRASSEN, *Berechnung und Programm.* I, Acta Inform., 1 (1972), pp. 320-335.

SIAM J. COMPUT.
Vol. 16, No. 5, October 1987
© 1987 Society for Industrial and Applied Mathematics
010

ON THE OPTIMAL LAYOUT OF PLANAR GRAPHS WITH FIXED BOUNDARY*

B. BECKER† AND G. HOTZ†

Abstract. The optimal planar layout of planar graphs with respect to the L_1- or L_2-metric leads to NP-hard problems, if one assumes the nodes of the graph to be fixed in the plane (see [FiPa], [Be]).

In this paper we consider the (optimal) layout of graphs with fixed boundary (i.e., graphs, where only the nodes of a given cycle of the graph have fixed positions in the plane). The investigated layouts are straight line embeddings in a continuous part of the plane; the cost of a layout is calculated with help of very general cost functions including the pth power of the usual Euclidean distance metric for $p = 2, 3, \cdots$ (for short, l_p-metric).

For a large class of graphs, which, for example, occur in chip layout problems as the abstract structure of switching circuits, we show the existence and uniqueness of the optimal layout.

The main part of the paper is concerned with planar graphs. We get an interesting characterization of nonplanar layouts of planar graphs, which shows that the optimal layout of a planar graph is planar or at least "quasiplanar." This property makes it possible to decompose the general layout problem into two independent problems:

(i) Find a layout of a circuit that is not necessarily planar, but that has an "allowed crossing behaviour."

(ii) Fix the crossing points and then optimize the layout. Our theorems show that no new contacts (crossing points) will be generated.

In the Appendix we outline some (efficient, polynomial time) methods for the construction of optimal layouts and give as an example the optimal layouts of a recursively defined n-bit adder and multiplier.

Key words. graph with fixed boundary, l_p-metric, (optimal) layout, (quasi) planar, star, convex point

AMS(MOS) subject classifications. 05C10, 57M15, 68C05

1. Introduction.

In this paper we develop the concept of "graph with fixed boundary" and present some results on the (optimal) layout of those graphs in the Euclidean plane (with respect to a given cost function).

A graph is a commonly used model to represent the abstract structure of a switching circuit (nodes of the graph equal modules (function blocks or basic cells) of the circuit, edges of the graph equal signal nets interconnecting the modules). If this circuit is to be realized on a (VLSI-) chip, there are some modules, the input- and output-pins, which have to be located on the boundary of the chip area. In our model we therefore assume that a cycle of the graph is given and embedded in the plane which corresponds to the boundary of the chip and whose nodes are identified with the i/o-pins of the circuit. We call this structure "graph with fixed boundary." So a graph with fixed boundary models the abstract structure of a circuit together with topological constraints given by the boundary of the chip and fixed i/o-ports.

For the actual construction of a circuit on a chip it is necessary to place the (remaining) modules of the circuit on the chip area, too, and to route the wires interconnecting the modules according to given design and optimization criteria. These problems are well known in the area of design automation as *placement and routing problems* [Br]. A layout of a graph with fixed boundary in the Euclidean plane within the area prescribed by the fixed boundary may be interpreted as a first placement and

* Received by the editors January 2, 1984; accepted for publication (in revised form) December 23, 1986.
† Fachbereich 10, Universität des Saarlandes, D-6600 Saarbrücken, West Germany.

global routing of the corresponding circuit. This is done (and the results of this paper are used) in a CAD-system for the design of integrated circuits being developed at the Universität des Saarlandes. (For detailed information see [HBKM].)

We now give a brief review of other results and methods for placement and routing, in order to relate them to our approach. As pointed out at the beginning, the graph-theoretic model is commonly adopted. Placement and routing then mainly consist of constructing a good layout with respect to a given measure. There are many goals which have to be considered:

- Chip area should be minimized.
- Routing must be possible (according to given design rules).
- Testing has to be possible.
- Heat dissipation levels must be preserved, etc. $\cdots$.

It seems impossible to directly incorporate all these goals into an algorithm, but minimizing a total weighted edge length has turned out to best reflect all the goals and therefore is widely used (see [Br]).

We consider cost functions including the pth power of the usual Euclidean distance metric. Minimizing with respect to the square of the Euclidean length reflects the following idea: the modules are joined together by lines of force, similar to springs or rubber bands, that attract the modules to one another. We then construct an optimal layout (with respect to the l_2-metric), i.e., a layout where the force on each module (not on the boundary) is equal to 0. This is done to produce a balanced distribution of all modules on the chip area. In existing design automation systems placement heuristics are often used which are based on similar ideas (see e.g., "force-directed placement" in [Fi]). We also consider l_p-optimal layouts, which yield an even more balanced distribution and converge to a layout minimizing the longest wire (see [BeOs]).

Today's design automation systems use heuristics (as indicated above) since it has turned out during the last years that many problems in this area are NP-hard (see [SaBh], [Do]). Nevertheless we can construct optimal layouts for a graph with fixed boundary in polynomial time because of the following:

- Layouts are not considered in a discrete part of the plane (=grid) as, for example, in [Sto], but in an area of the plane given by a fixed boundary. The fixed boundary together with the i/o-pins of the circuit function as a "frame spanning the remaining nodes and edges." If the graph is not spanned by the boundary (i.e., the inputs and outputs are not fixed on the boundary) and the nodes are only allowed to be embedded on different grid points inside the boundary, the construction of the optimal layout corresponds to the solution of an NP-complete problem, the "Quadratic Assignment Problem" [SaGo]. On the other hand, the optimal layout for the graph with fixed boundary can be constructed in time less than $O(n^3)$ (n = number of nodes), as is shown in the Appendix.

- The locations of the modules not on the fixed boundary are freely moveable on the chip area. If all locations are fixed in the plane, the problem of finding a planar layout of minimal total weighted edge length with respect to the L_1- or L_2-metric is NP-hard (see [FiPa], [Be]). In our model we show as a main result of this paper that an optimization of a given planar graph does not destroy planarity (i.e., the placement algorithm does not destroy topological properties of the circuit, which have been transferred to the placement and routing phase from the logical topological level of the system (see [BHKM])).

- The resulting optimal layouts have wires running in any direction of the plane. This is not possible in current technologies; however, efforts to generalize the concept of strictly horizontal or vertical wires can be observed (see e.g., cover and preface in

[Bry]). On the other hand, some work has been done to embed the resulting optimal layouts in a grid [Schm].

There is no doubt that a graph with fixed boundary is a very idealized model of a real circuit. In order to construct a realistic physical circuit the optimal layout of the idealized graph has to be transformed into a layout, which realizes the edges and nodes by geometric configurations of given dimensions and thus separates edges and nodes according to given design rules. This can be done with help of an algorithm called "logarithmic pumping," which assigns the necessary configurations to nodes and edges on one side and minimizes the layout area in an iterative process on the other side. For detailed information see [BHKM] and [Schw].

In this paper we restrict ourselves to the study of "graphs with fixed boundary" and their layouts.

We finish the Introduction by outlining the remainder of the paper: First we give the basic definitions and show the existence and uniqueness of the optimal layout of a graph with fixed boundary with respect to a general cost function f including the l_p-metric for $p=2, 3, \cdots$. (In the case $p=1$ we still have the existence but not the uniqueness of the optimal layout.) The main part of the paper is concerned with connected planar graphs with fixed boundary and their layouts. We give a classification of layouts and show that some forms of layouts can never be optimal: Nonplanar layouts of planar graphs, which are not even "quasiplanar," have "convex points," but this contradicts optimality. This result is first proved for triangulated graphs (Lemma 6) and then generalized to planar graphs (Lemmas 7 and 8). Although the lemmas are intuitively clear, their proof is rather lengthy. It requires an exact classification of nonplanarity (Lemmas 2, 3 and 5) and finishes with the calculation of an invariant (Lemma 4). In a last part we show that the optimal layout of triconnected planar graphs is not only quasiplanar but also planar. This is done mainly with help of a technical lemma, the proof of which uses the properties of the cost function f together with analytical tools. In the Appendix we introduce methods to construct an optimal layout and present pictures of the optimal layout of a recursively defined adder and multiplier.

2. Basic definitions. Let $G=(V, E)$ be an undirected simple graph.

A *layout* χ of G in the plane is given by a mapping $\chi: V \to \mathbb{R}^2$: i.e., the edges $e=(v_1, v_2)$ of G are mapped on the straight line segments $\overline{\chi(v_1)\chi(v_2)} =: \chi(e)$. (Note that in this general definition different nodes may be mapped on the same point in the plane.)

$G_{\rho(c)} := (G, c, \rho)$ is called *a graph with fixed boundary*, iff the following hold:

- $G=(V, E)$ is a (undirected, simple) graph with $V = V_{\text{in}} \cup V_c$.
- $c=(V_c, E_c)$ is a nontrivial simple cycle of G.
- $\rho: V_c \to \mathbb{R}^2$ is a layout of c, which maps c onto a convex polygon $\rho(c)$.

$$C_{\rho(c)} := \left\{ x \in \mathbb{R}^2 \,\middle|\, x = \sum_{v \in V_c} \lambda_v \cdot \rho(v) \text{ with } \lambda_v \in [0, 1], \sum_{v \in V_c} \lambda_v = 1 \right\}$$

is a convex set, called the *convex closure* of $\rho(c)$ and denotes $\rho(c)$ together with its interior.

A layout χ of G, denoted by $\chi(G)$, is called $\rho(c)$-respecting, iff $G_{\rho(c)} = (G, c, \rho)$ is a graph with fixed boundary and $\chi|c = \rho$, i.e., the cycle c is mapped on $\rho(c)$ in the layout χ. We give an example of a $\rho(c)$-respecting layout in Fig. 1.

In the following we investigate

$$L(G, c, \rho) := \{\chi \mid \chi \text{ is a } \rho(c)\text{-respecting layout of } G\},$$

the *set of $\rho(c)$-respecting layouts of G.*

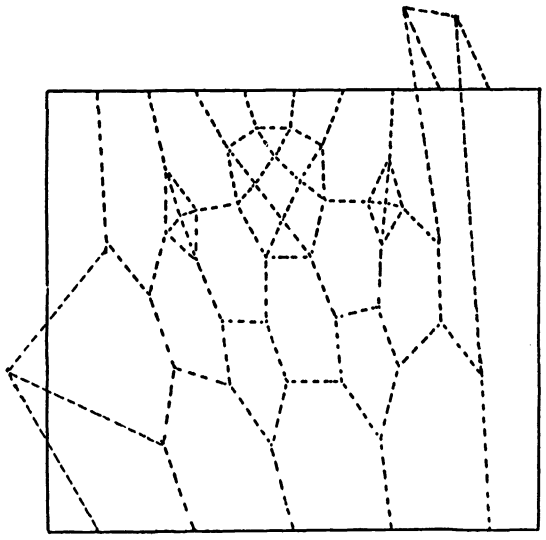


FIG. 1. —— *denotes the fixed boundary*; ---- *denotes the remaining edges.*

Remark. If we have $V_{in} = \{v_1, v_2, \cdots, v_n\}$ we get the following bijection:

$$\begin{array}{ccl} L(G, c, \rho) & \rightarrow & \mathbb{R}^{2n} \\ \chi & \rightarrow & (x_1, y_1, x_2, y_2, \cdots, x_n, y_n) \\ & & \text{with } (x_i, y_i) := \chi(v_i) \\ & & \text{for } i = 1, \cdots, n. \end{array}$$

We therefore identify a $\rho(c)$-respecting layout $\chi(G)$ with a point in $\mathbb{R}^{2n}$ and vice versa.

Next we define some special kind of $\rho(c)$-respecting layouts, where all nodes are placed inside the area prescribed by the boundary. (These layouts "correspond" to layouts of a given circuit on a given chip area.)

A layout $\chi(G)$ is called $\rho(c)$-*bounded*, iff χ is $\rho(c)$-respecting and $\chi(V) \subset C_{\rho(c)}$. We give an example in Fig. 2.

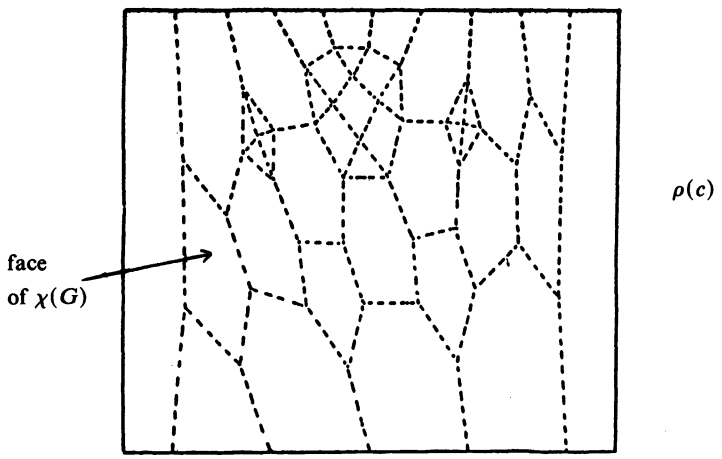


FIG. 2

In the example of Fig. 2 we have a planar embedding inside $\rho(c)$. (Different nodes are mapped on different points, different edges are mapped on disjoint straight line segments (up to common endpoints).)

Henceforth we call a layout $\chi(G)$ $\rho(c)$-*planar*, iff χ is $\rho(c)$-bounded and planar.

A graph G with fixed boundary $\rho(c)$ is called $\rho(c)$-*planar*, iff there exists a layout $\chi(G)$, which is $\rho(c)$-planar.

Remark. There does not exist a $\rho(c)$-planar layout for each nontrivial cycle c of a planar graph. This easily follows from the uniqueness theorem for planar embeddings of triconnected planar graphs [Ho]. Furthermore, one sees from the construction given in [Ho] which cycles c of a planar graph yield planar embeddings and how many "topologically" different layouts exist.

Let $\chi(G)$ be a $\rho(c)$-planar layout of a biconnected graph G. "Removal of $\chi(e)$ for all edges e of G from the plane" splits the plane into connected open regions. Exactly one region ("the outer region") is unbounded. A finite region together with its boundary is called a *face* of $\chi(G)$. Let F be a face, then int (F) denotes the interior of F and ∂F the boundary of F. (See Fig. 2.)

A layout $\chi(G)$ of a biconnected graph G is called $\rho(c)$-planar convex, iff χ is $\rho(c)$-planar and all faces of $\chi(G)$ are convex regions.

We will need these definitions later when we try to find minimal elements of $L(G, c, \rho)$ with respect to certain cost functions. The cost of a layout $\chi(G)$ is defined as follows: Let $f: \mathbb{R}^+ \to \mathbb{R}^+$ be nondecreasing and continuous, then $|\chi|_f := \sum_{e \in E} f(|\chi(e)|)$, where $|\chi(e)| :=$ Euclidean length of $\chi(e)$ is the *cost of* χ *with respect to the cost function* f. In the case $f(x) = x^p (p = 2, 3, \cdots,)$ we call $|\cdot|_f$ l_p-metric. We try to find a layout χ_{opt} with $|\chi_{\text{opt}}|_f = \inf\{|\chi|_f \mid \chi \in L(G, c, \rho)\}$. χ_{opt} is then called *optimal layout* of G with respect to $\rho(c)$ and f.

3. Existence and uniqueness of the optimal layout. We first show the existence of an optimal layout χ_{opt} with respect to a given cost function f and will then see that with some additional assumptions for G and f χ_{opt} is unique.

Let $G' = (V', E')$ be a component of G. If $V' \cap V_c = \varnothing$, then an optimal layout $\chi(G')$ is given by $\chi(v') = (x, y) \in \mathbb{R}^2$, $\forall v' \in V'$. If $V' \supset V_c$, we conclude with the help of the monotony of f and the convexity of $C_{\rho(c)}$.

Remark. Let $\chi(G')$ be a $\rho(c)$-respecting layout which is not $\rho(c)$-bounded. Then there exists a $\rho(c)$-bounded layout $\bar{\chi}(G')$ with $|\chi(G')|_f \geqq |\bar{\chi}(G')|_f$.

This means that if there exists an optimal layout of G, we may assume that it is $\rho(c)$-bounded. Together with the compactness of $C_{\rho(c)}$ and the bijection between $L(G, c, \rho)$ and $\mathbb{R}^{2n}$ we get the following theorem.

THEOREM 1 (Existence). *Let $G_{\rho(c)} = (G, c, \rho)$ be a graph with fixed boundary. Then there exists an optimal layout χ_{opt} with respect to $\rho(c)$ and f, which is $\rho(c)$-bounded.*

If we assume connectedness for G and strict convexity for f (as is the case for $f(x) = x^p$ with $p \geqq 2$, but not for $f(x) = x$), we get the uniqueness of χ_{opt}.

THEOREM 2 (Uniqueness). *Let $G_{\rho(c)} = (G, c, \rho)$ be a connected graph with fixed boundary and $f: \mathbb{R}^+ \to \mathbb{R}^+$ a continuous, nondecreasing and strictly convex function. Then the optimal layout of G with respect to $\rho(c)$ and f is unique.*

Proof of Theorem 2. For the proof we consider the cost $|\cdot|_f$ and show that $|\cdot|_f$ is strictly convex on $\mathbb{R}^{2n}$. With the help of Theorem 1 we then get the following: The unique minimum of $|\cdot|_f$ defines the optimal $\rho(c)$-bounded layout χ_{opt}.

We now give the exact proof. Let $G = (V, E)$ with $V = V_{\text{in}} \cup V_c$ and $V_{\text{in}} = \{v_1, \cdots, v_n\}$, $V_c = \{v_{n+1}, \cdots, v_m\}$. Then $\rho(c)$ may be identified with $P_c := (a_{n+1}, b_{n+1}, \cdots, a_m, b_m) \in \mathbb{R}^{2(m-n)}$ with $(a_i, b_i) := \rho(v_i)$, $\forall i \in \{n+1, \cdots, m\}$. A layout

$\chi \in L(G, c, \rho)$ is given by $P = (x_1, y_1, \cdots, x_n, y_n) \in \mathbb{R}^{2n}$ and if we define the Euclidean metric by

$$\begin{array}{rccc} h: & \mathbb{R}^4 & \to & \mathbb{R} \\ & (x_1, y_1, x_2, y_2) & \to & ((x_1-x_2)^2+(y_1-y_2)^2)^{1/2}, \end{array}$$

the cost function $|\cdot|_f$ is given as the following real-valued function

$$\begin{array}{rccl} |\cdot|_f: & \mathbb{R}^{2n} & \to & \mathbb{R} \\ & (x_1, y_1, \cdots, x_n, y_n) & \to & \displaystyle\sum_{\substack{i,j\in\{1,\cdots,n\} \\ (v_i,v_j)\in E}} f(h(x_i, y_i, x_j, y_j)) \\ & & & \quad + \displaystyle\sum_{\substack{i\in\{1,\cdots,n\} \\ j\in\{n+1,\cdots,m\} \\ (v_i,v_j)\in E}} f(h(x_i, y_i, a_j, b_j)) \\ & & & \quad + \displaystyle\sum_{\substack{i,j\in\{n+1,\cdots,m\} \\ (v_i,v_j)\in E}} f(h(a_i, b_i, a_j, b_j)). \end{array}$$

A short calculation shows that

(i) h is a convex function, and

(ii) if $((x = (x_1, x_2, x_3, x_4) \neq (x_1, x_2, \bar{x}_3, \bar{x}_4) = \bar{x})$ and $h(x) = h(\bar{x})$), then h is strictly convex on $\{y \mid y = \lambda x + (1-\lambda)\bar{x}, \lambda \in [0, 1]\}$.

On the other hand, $|\cdot|_f$ is convex as the sum of convex functions $f \cdot h$. In order to show that $|\cdot|_f$ is strictly convex, it suffices to show that one term of the above sum is strictly convex. Because of the connectivity of G we succeed in finding i, j with the following:

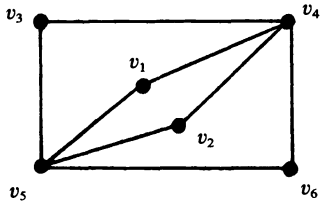
(a) $(x_i, y_i, a_j, b_j) \neq (\bar{x}_i, \bar{y}_i, a_j, b_j)$, or

(b) $(x_i, y_i, x_j, y_j) \neq (\bar{x}_i, \bar{y}_i, \bar{x}_j, \bar{y}_j)$ but $((x_i, y_i) = (\bar{x}_i, \bar{y}_i)) \vee ((x_j, y_j) = (\bar{x}_j, \bar{y}_j))$

for any two layouts $(x_1, y_1, \cdots, x_n, y_n) \neq (\bar{x}_1, \bar{y}_1, \cdots, \bar{x}_n, \bar{y}_n)$. Because of (a), (b) and since f is nondecreasing strictly convex, the corresponding term of $|\cdot|_f$ is strictly convex. So we have: $|\cdot|_f$ is strictly convex. Together with Theorem 1 we get the proof of Theorem 2. □

4. Planarity of the optimal layout. In the following we investigate the optimal layout of a connected $\rho(c)$-planar graph with fixed boundary. We would like to show that an optimization does not destroy the topological properties of the circuit, i.e., the optimal layout of a $\rho(c)$-planar graph is $\rho(c)$-planar, also. But this is not always true, as it is shown in [Gr] for the example in Fig. 3. (We only give the picture.)

We prove the following: Optimal layouts of $\rho(c)$-planar graphs are "quasi-$\rho(c)$-planar." The intuitive meaning of "χ is quasi-$\rho(c)$-planar" is: "If one looks at χ, χ seems to be $\rho(c)$-planar, for example, there is no crossing over of edges." In Fig. 3 the optimal layout χ_{opt} is quasi-$\rho(c)$-planar, but not $\rho(c)$-planar, since $\chi_{\text{opt}}(v_2) = \chi_{\text{opt}}(v_1)$. (See Fig. 4.)



$V_{\text{in}} = \{v_1, v_2\}$
$V_c = \{v_3, v_4, v_5, v_6\}$

FIG. 3

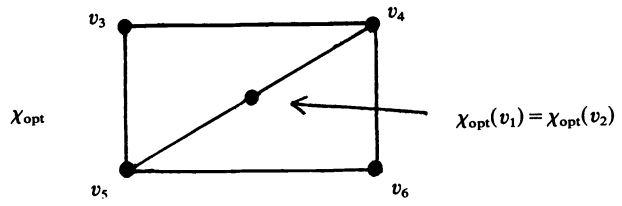


FIG. 4

If we assume that G is triconnected and that the fixed boundary allows a planar layout, we can prove a still better result: the optimal layout is $\rho(c)$-planar.

The subsequent definition of "convex point" is of great importance if one wants to prove the nonoptimality of a given layout.

DEFINITION. A straight line g in $\mathbb{R}^2$ divides the plane into two infinite regions $H_1(g)$, $H_2(g)$ with $H_1(g) \cap H_2(g) = g$. For $v \in V$ the *edge set of* v is defined as $E(v) := \{e \mid e \in E, e = (v, w)\}$. The *node set of* v is defined as $V(v) := \{w \mid w \in V, e = (v, w) \in E(v)\}$. Let χ be a $\rho(c)$-bounded layout of G. $v \in V_{in}$ is called a *convex point of* χ, iff the following hold:

(i) $\{\chi(v)\} \subsetneq \chi(E(v))$;

(ii) There exists a straight line g with $g \cap \chi(E(v)) = \{\chi(v)\}$ and $\chi(E(v)) \subset H_1(g)$.

Remark. χ has a convex point, iff χ looks locally like the example in Fig. 5.

The following lemma proves that an optimal layout can never have convex points.

LEMMA 1. *Let χ be a $\rho(c)$-bounded layout of G with a convex point v, f continuous and increasing. Then χ is not optimal with respect to f.*

Proof. According to the definition of convex point we have the following situation. (See Fig. 6.)

Because of the strict monotony of f there exists an $\varepsilon > 0$, such that a change of $\chi(v)$ vertical to g by ε to $\bar{\chi}(v)$ results in a better layout. (See Fig. 7.)

We have $\sum_{i=1}^{n} f(g(\chi(v), \chi(v_i))) > \sum_{i=1}^{n} f(g(\bar{\chi}(v), \chi(v_i)))$ because $g(\chi(v), \chi(v_i)) > g(\bar{\chi}(v), \chi(v_i))$, $\forall i = 1, \cdots, n$. □

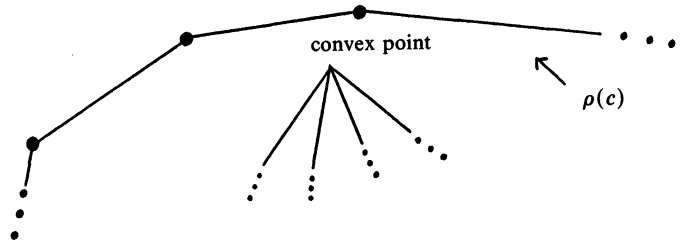


FIG. 5

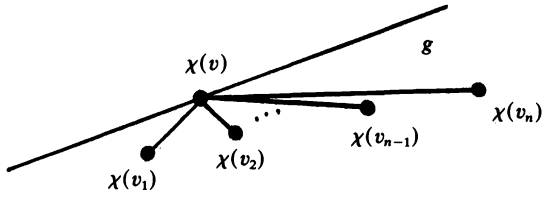


FIG. 6

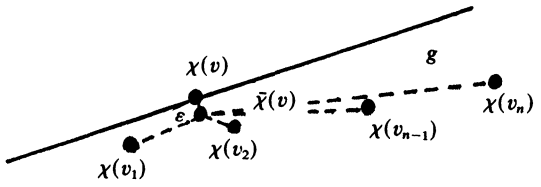


FIG. 7

4.1. Quasiplanarity and optimality. In this part of the paper we show that optimal layouts of $\rho(c)$-planar graphs are quasi-$\rho(c)$-planar. The results are first shown for triangulated graphs, then the generalization to any $\rho(c)$-planar graph is given.

DEFINITION. A graph G with fixed boundary $\rho(c)$ is called $\rho(c)$-triangulated, iff

(i) G is biconnected and $\rho(c)$-planar;

(ii) For any $\rho(c)$-planar layout $\chi_0(G)$ all faces of $\chi_0(G)$ are triangles.

With the help of [Wh] one easily concludes that the faces of a $\rho(c)$-triangulated graph G are uniquely determined. For the following let G be a $\rho(c)$-triangulated graph and let χ_0 be a fixed $\rho(c)$-planar layout of G. The set of triangles of χ_0 is denoted by

$$T(\chi_0) := \{\Delta_{\chi_0}(v_1, v_2, v_3) \mid \Delta_{\chi_0}(v_1, v_2, v_3) \text{ is a face of } \chi_0 \text{ and is "formed" by the points } \chi_0(v_1), \chi_0(v_2), \chi_0(v_3)\}.$$

In order to determine planarity, respectively, nonplanarity, any other layout $\chi(G)$ will be compared with χ_0 locally: for example, we consider the sequence of edges emanating from $\chi_0(v)$ and $\chi(v)$ (called the "stars in $\chi_0(v)$ and $\chi(v)$"). The instrument we use to do this is a mapping $\varphi(\chi_0, \chi)$: $C_{\rho(c)} \to C_{\rho(c)}$. φ maps the triangles of $T(\chi_0)$ onto the corresponding triangles in χ. More precisely, if one considers χ_0 and any point $P \in C_{\rho(c)}$, we have exactly one of the following three possibilities:

(i) $P = \chi_0(v)$ for exactly one $v \in V$.

(ii) There exists exactly one edge $\overline{\chi_0(v_1)\chi_0(v_2)}$ with the following properties: $\overline{\chi_0(v_1)\chi_0(v_2)}$ is an edge of a triangle of $T(\chi_0)$ and P has a unique representation $P = \lambda_1\chi_0(v_1) + \lambda_2\chi_0(v_2)$ with $\lambda_1, \lambda_2 \in (0, 1)$, $\lambda_1 + \lambda_2 = 1$.

(iii) There exists exactly one triangle $\Delta_{\chi_0}(v_1, v_2, v_3)$ of $T(\chi_0)$ and P lies in the interior of $\Delta_{\chi_0}(v_1, v_2, v_3)$. Then P has a unique representation $P = \sum_{i=1}^{3} \lambda_i\chi_0(v_i)$ with $\lambda_i \in (0, 1)$ and $\sum_{i=1}^{3} \lambda_i = 1$.

If χ is any $\rho(c)$-bounded layout of G, $\varphi(:= \varphi(\chi_0, \chi)): C_{\rho(c)} \to C_{\rho(c)}$ is defined by the following:

$$\varphi(P) = \begin{cases} \chi(\chi_0^{-1}(P)) & P \text{ belongs to case (i)}, \\ \lambda_1\varphi(\chi_0(v_1)) + \lambda_2\varphi(\chi_0(v_2)) & P \text{ belongs to case (ii)}, \\ \sum_{i=1}^{3} \lambda_i\varphi(\chi_0(v_i)) & P \text{ belongs to case (iii)}. \end{cases}$$

We call φ a deformation of χ_0 with respect to χ. The following remark is well known from topology [Re].

Remark. φ is well defined, continuous and surjective.

We illustrate the mapping φ with an example (see Fig. 8), which will give an intuitive idea how to prove the quasiplanarity of the optimal layout.

If we look at the example in Fig. 8, we see that φ changes the "stars" in $\chi_0(v_4)$ and $\chi_0(v_5)$. If we measure the angles of the triangles meeting in $\chi(v_4)$, we have a sum

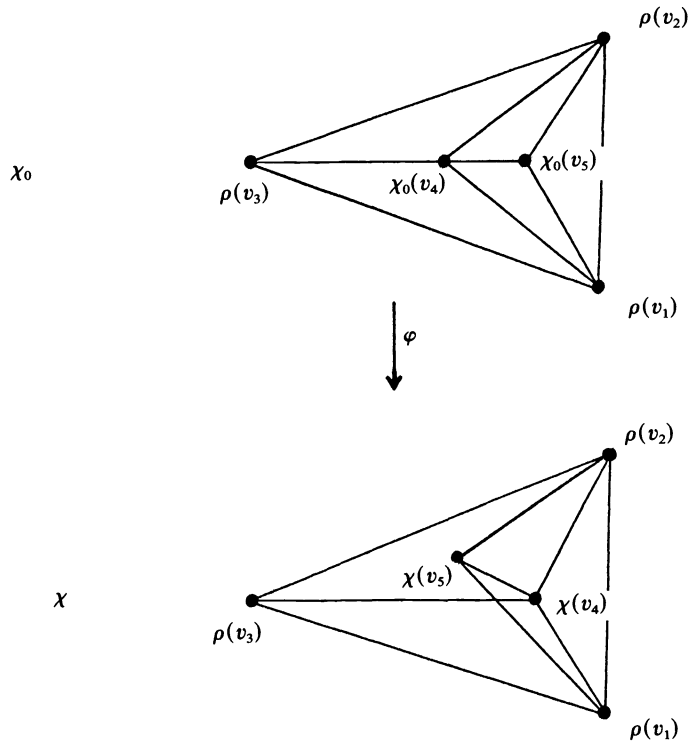


FIG. 8. *χ_0 is a $\rho(c)$-planar layout of a graph G; χ is a non-$\rho(c)$-planar layout of the same graph.*

of more than 2π, if we measure in $\chi(v_5)$, we have less than 2π, whereas for all points of V_{in} in χ_0 we get exactly 2π. This gives the motivation for the following idea: If a layout is not (quasi)-$\rho(c)$-planar, we try to find a point $v \in V_{\text{in}}$, where the star in $\chi_0(v)$ is changed by φ. If there exists such a point, we conclude that there exists a point where the sum of all angles is less than 2π; this point then will be a convex point. ($\chi(v_5)$ in the example of Fig. 8 is a convex point.) The existence of a convex point contradicts optimality according to Lemma 1.

In order to prove this exactly, we first have to give the formal definition of star. For the following let χ be a layout with $|\chi(e)| \neq 0$, $\forall e \in E$, i.e., $\chi(e)$ is not a point for all $e \in E$. For edges of $E(v)$, $v \in V$, we define an equivalence relation "$\sim_{\chi(v)}$":

$e_1 \sim_{\chi(v)} e_2 :\Leftrightarrow \text{card}\,(\chi(e_1) \cap \chi(e_2)) > 1$,
$[e_1]$ then is the equivalence class of e_1 in $\chi(v)$,
i.e., the set of all edges in $E(v)$, which overlap with e_1.

If $e_1, e_2 \in E(v)$ with $\chi(e_1) \cap \chi(e_2) = \{\chi(v)\}$,

$\sphericalangle(\chi(e_1), \chi(e_2)) :=$ the angle in $\chi(v)$ between $\chi(e_1)$ and $\chi(e_2)$
(in clockwise direction from $\chi(e_1)$ to $\chi(e_2)$).

$S_\chi(v) = ([e_1], [e_2], \cdots, [e_n])$ is called *the star in $\chi(v)$* iff:
(i) $\cup_{i=1}^{n} [e_i] = E(v)$, $(n > 1)$;

(ii) $\forall i, j \in \{1, \cdots, n\}, i \neq j$:

$$(e \in [e_i], e' \in [e_j]) \Rightarrow (\chi(e) \cap \chi(e') = \{\chi(v)\});$$

(iii) For $i = 1, \cdots, n$: $\chi(e_i)$ is the direct left neighbour of $\chi(e_{i+1})$ and $\sphericalangle(\chi(e_i), \chi(e_{i+1})) \leqq \pi$ for $\{e_i, e_{i+1}\} \not\subset E_c$ (we define $e_{n+1} := e_1$!) (i.e., if there exists i with $\sphericalangle(\chi(e_i), \chi(e_{i+1})) > \pi$ (see $S_\chi(v_5)$ in Fig. 8), then we have a convex point and the star is not defined). If $S_\chi(v)$ is defined, it is unique up to cyclic permutation. (Hence we call two stars equal, iff they are equal up to cyclic permutation.)

Let $S_\chi(v) = ([e_1], \cdots, [e_n])$ $(v \in V)$. Then $([e_1], \cdots, [e_n])$ is called a *clockwise enumeration of* $E(v)$ *in* $\chi(v)$, $([e_n], [e_{n-1}], \cdots, [e_1])$ a *counterclockwise enumeration.* If card $([e_i]) = 1$, $\forall i = 1, \cdots, n$ (i.e., no edges lie upon each other), the star in $\chi(v)$ is called *simple.*

Remark. For χ_0 $S_{\chi_0}(v)$ is defined and simple for all $v \in V$.

We now consider the "image" of $S_{\chi_0}(v)$ for $v \in V$ in χ, which is obtained by collapsing edges adjacent in χ_0 that are in the same equivalence class with respect to "$\sim_{\chi(v)}$". More precisely, let $S_{\chi_0}(v) = (e_1, \cdots, e_n)$. Then (with respect to "$\sim_{\chi(v)}$") there exists $l \in [1, n]$ and $n_1 < n_2 < \cdots < n_l$ $(n_i \in [1, n]$ for $i = 1, \cdots, l)$ with:

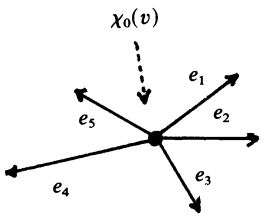
$$[e_{n_i}] \neq [e_{n_{i+1}}] \quad \text{for } i = 1, \cdots, l \quad ([e_{n_{l+1}}] := [e_{n_1}]),$$

$$E_{n_{i+1}} := \{e_{n_i+1}, e_{n_i+2}, \cdots, e_{n_{i+1}}\} \subset [e_{n_{i+1}}] \quad \text{for } i = 1, \cdots, l-1,$$

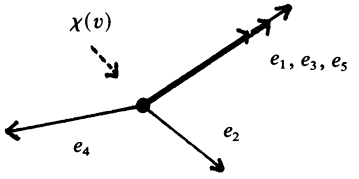
$$E_{n_1} := \{e_1, \cdots, e_{n_1}, e_{n_l+1}, \cdots, e_n\} \subset [e_{n_1}].$$

We call $\chi(S_{\chi_0}(v)) := (E_{n_1}, \cdots, E_{n_l})$ the *image of* $S_{\chi_0}(v)$ *in* χ. We say $S_{\chi_0}(v)$ *induces* $S_\chi(v)$, iff $\chi(S_{\chi_0}(v))$ is equal to $S_\chi(v)$ (up to cyclic permutation), for short, $\chi(S_{\chi_0}(v)) = S_\chi(v)$. For an example see Fig. 9.

$S_\chi(v)$ is called a *full star,* iff $S_{\chi_0}(v)$ induces $S_\chi(v) := ([e_1], \cdots, [e_n])$ and $\sphericalangle(\chi(e_i), \chi(e_{i+1})) < \pi$ for $[e_i] \cap E_c = \emptyset$ or $[e_{i+1}] \cap E_c = \emptyset$. $S_\chi(v)$ is called a *half star* (along g), iff $S_{\chi_0}(v)$ induces $S_\chi(v)$ and looks like the example in Fig. 10. $S_\chi(v)$ is called an *inverse half star* (along g), iff $\chi(S_{\chi_0}(v))$ is a counterclockwise enumeration of $S_\chi(v)$ and looks like the example in Fig. 10. $S_\chi(v)$ is called a *degenerated star* (along g), iff $S_{\chi_0}(v)$ induces $S_\chi(v)$ and looks like the example in Fig. 11.



(a) $S_{\chi_0}(v) = (\{e_1\}, \{e_2\}, \{e_3\}, \{e_4\}, \{e_5\})$.



(b) $S_\chi(v) = (\{e_1, e_3, e_5\}, \{e_2\}, \{e_4\})$,
$\chi(S_{\chi_0}(v)) = (\{e_1, e_5\}, \{e_2\}, \{e_3\}, \{e_4\})$.

FIG. 9

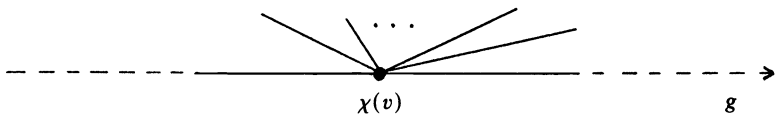


FIG. 10

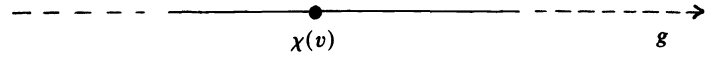


FIG. 11

In the following we give a characterization lemma for planar and quasiplanar layouts.

LEMMA 2. *Let χ be a $\rho(c)$-bounded layout of G. Then we have:*

$$\forall v \in V;\ S_\chi(v) \text{ is defined, simple and induced by } S_{\chi_0}(v)$$
$$\Leftrightarrow$$
$$\chi \text{ is } \rho(c)\text{-planar.}$$

An exact proof of this lemma can be found in [Gr]. Lemma 2 gives the possibility of being able to decide the planarity of χ only by looking at all nodes $v \in V$ and comparing the stars in $\chi(v)$ and $\chi_0(v)$.

We prove an analogous property for the quasiplanarity of a layout χ in Lemma 3. At first we give the formal definition of "quasi-$\rho(c)$-planar."

DEFINITION. Let G be $\rho(c)$-triangulated, $\chi_0(G)\rho(c)$-planar, χ a layout of G (not necessarily $|\chi(e)|>0, \forall e \in E$). $\chi(v)(v \in V)$ is called a *peak*, iff $\exists e \in E(v)$: $|\chi(e)|>0$, $\forall e, e' \in E(v)$ with $|\chi(e)|, |\chi(e')|>0$: $\operatorname{card}(\chi(e) \cap \chi(e'))>1$. χ is called *quasi-$\rho(c)$-planar*, iff

(i) $\forall \Delta_1, \Delta_2 \in T(\chi_0)$: $\varphi(\Delta_1) \cap \operatorname{int}(\varphi(\Delta_2)) = \varnothing$,

(ii) $\forall v \in V$: $\chi(v)$ is not a peak.

Point (i) of the definition corresponds to the intuitive idea of quasiplanar given at the beginning of this chapter: χ is planar up to triangles collapsed to a line or a point. Point (ii) makes sure that there are no "hidden" convex points in regions where triangles are collapsed.

LEMMA 3. *Let χ be a $\rho(c)$-bounded layout of G with $|\chi(e)|>0, \forall e \in E$. Then we have:*

$$\forall v \in V:\ S_\chi(v) \text{ is defined and induced by } S_{\chi_0}(v)$$
$$\Rightarrow$$
$$\chi \text{ is quasi-}\rho(c)\text{-planar.}$$

Proof. The proof is a general version of that of Lemma 2 (see [Gr]). We prove 5 facts that will finally give the complete proof.

FACT 1. *There exist no peaks in χ (since $S_\chi(v)$ is defined for all $v \in V$).*

Next we consider neighbourhoods of nodes and edges in χ, χ_0. For a point $v \in V$ we define the following:

The *neighbourhood of v for χ_0*:

$$U_{\chi_0}(v) := \bigcup_{\substack{\Delta_{\chi_0}(v_1, v_2, v_3) \in T(\chi_0),\\ v_i = v \text{ for } i = 1 \vee 2 \vee 3}} \Delta_{\chi_0}(v_1, v_2, v_3).$$

The *neighbourhood of v for χ*:

$$U_\chi(v) := \bigcup_{\substack{\Delta_{\chi_0}(v_1, v_2, v_3) \in T(\chi_0),\\ v_i = v \text{ for } i = 1 \vee 2 \vee 3}} \Delta_\chi(v_1, v_2, v_3), \quad \text{with } \Delta_\chi(v_1, v_2, v_3) := \varphi(\Delta_{\chi_0}(v_1, v_2, v_3)).$$

In the following we consider an edge $e_0 = (v_0, w_0)$ with $\chi(e_0) \not\subset \rho(c)$ and construct a *generalized neighbourhood* $U_\chi(e_0)$ of e_0 in χ, which consists of several neighbourhoods "surrounding" e_0. Here there are two cases of interest. Assume that e_0 is an edge of a triangle $\Delta \in T(\chi_0)$, which does not degenerate in χ (i.e., $\operatorname{int}(\varphi(\Delta)) \neq \varnothing$):

(i) If $[e_0] = \{e_0\}$ in $\chi(v_0)$, then ($[e_0] = \{e_0\}$ in $\chi(w_0)$ and) we define $U_\chi(e_0) := U_\chi(v_0) \cup U_\chi(w_0)$. (See Fig. 12.)

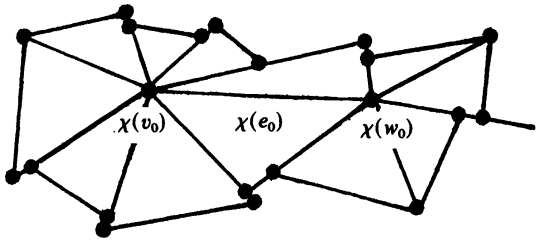


FIG. 12

(ii) If $[e_0] \neq \{e_0\}$, we can show the existence of a cycle in G (resp. $\chi_0(G)$), which collapses (together with its interior) to a straight line segment in the layout χ. This cycle together with its interior and its "surrounding" triangles will give the neighbourhood of e_0 in this case. (For an example, see Fig. 16.)

More precisely we give Fact 2.

FACT 2. *There exists a unique simple cycle* $c_0 = (V_0, E_0)$ *in* G *with the following properties*:

(a) $\{v_0, w_0\} \subset V_0$, $e_0 \in E_0$.

(b) $\chi(c_0)$ *lies on a straight line* g.

(c) *There are exactly two points* $u_1, u_2 \in V_0$ *where* $S_\chi(u_1)$, $S_\chi(u_2)$ *are not degenerated and not half stars along* g (u_1, u_2 *are called endpoints of* c_0 *in* χ), $\forall v \in V_0$, $v \neq u_1, u_2$ *we have*: $S_\chi(v)$ *is a half star* (*along* g).

(d) $\forall v \in V$ "*inside* c_0" (*with respect to* χ_0): $\chi(v)$ *lies on* $\chi(c_0)$ *and has a degenerated star.*

Proof of Fact 2. We start with the construction of the endpoints of c_0. Without loss of generality assume a situation as in Fig. 13. Let g be the straight line defined by e_0. According to the direction of g the relation $P < P'$ is well defined for two points P, P' on g. (For example, $\chi(v_0) < \chi(w_0)$ in Fig. 13.)

Either $S_\chi(v_0)$ is not a half star along g:

– Then v_0 is one of the endpoints of c_0;

or $S_\chi(v_0)$ is a half star along g:

– Then we can continue our considerations with $e_{v_1} := (v_1, v_0)$ instead of e_0, and e_{v_1} is an edge of a triangle $\Delta_{v_1} \in T(\chi_0)$ with $\text{int}(\varphi(\Delta_{v_1})) \neq \emptyset$. (See Fig. 14.)

The node $\chi(w_0)$ is treated analogously.

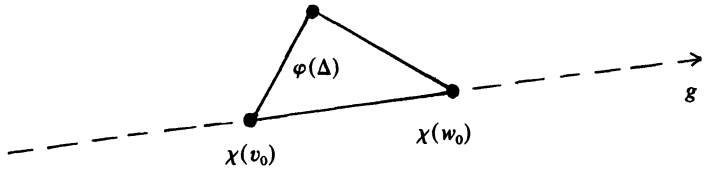


FIG. 13

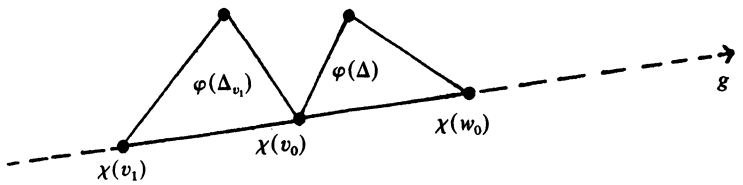


FIG. 14

By induction (see Fig. 15) we finally get a sequence $(v_0, v_1, \cdots, v_n)$ (resp. $(w_0, \cdots, w_m)$) of nodes as follows:

$S_\chi(v_i)$ (resp. $S_\chi(w_j)$) is a half star along g for $i = 1, \cdots, n-1$ (resp. $j = 1, \cdots, m-1$);

$S_\chi(v_n)$ (resp. $S_\chi(w_m)$) is not a half star along g and not degenerated.

If we consider $[e_{v_n}]$ $(e_{v_n} := (v_n, v_{n-1}))$ in $\chi(v_n)$, we find an edge $e_{v_{n+1}} = (v_n, v_{n+1}) \in [e_{v_n}]$ as follows:

$e_{v_n} \neq e_{v_{n+1}}$ and $e_{v_{n+1}}$ is an edge of a triangle $\Delta_{v_{n+1}} \in T(\chi_0)$ with $\operatorname{int}(\varphi(\Delta_{v_{n+1}})) \neq \varnothing$.

Analogously, we get the existence of an edge $e_{w_{m+1}}$ with the corresponding properties.

The edges e_{v_n}, $e_{v_{n-1}} := (v_{n-1}, v_{n-2})$, $\cdots$, e_{v_1}, e_0, $e_{w_1} := (w_1, w_0)$, $e_{w_2} := (w_2, w_1)$, $\cdots$, $e_{w_m} := (w_m, w_{m-1})$ form a first part of the cycle c_0. In a next step we deform the sequence $c := (u_1, \cdots, u_1) := (v_n, v_{n-1}, \cdots, v_0, w_0, \cdots, w_m)$, until we get the second part of the cycle c_0. The following two deformations are applied repeatedly: ($V(c_0)$ ultimately contains the nodes $v_{n-1}, \cdots, v_0, w_0, \cdots, w_{m-1}$ and the nodes inside c_0. At the beginning $V(c_0)$ is empty.)

(i) If there exists $i \in \{2, \cdots, l-1\}$, such that $\chi(u_{i-1}) < \chi(u_i) < \chi(u_{i+1})$ and $\Delta_{\chi_0}(u_{i-1}, u_i, u_{i+1}) \in T(\chi_0)$, then change c to $(u_1, \cdots, u_{i-1}, u_{i+1}, \cdots, u_1)$ and set $V(c_0) := V(c_0) \cup \{u_i\}$.

(ii) If there exist $i \in \{1, \cdots, l-1\}$ and $u \notin V(c)$, such that $\chi(u_i) < \chi(u) < \chi(u_{i+1})$ and $\Delta_{\chi_0}(u_i, u, u_{i+1}) \in T(\chi_0)$, then change c to $(u_1, \cdots, u_i, u, u_{i+1}, \cdots, u_l)$.

We apply (i) and (ii) as long as possible. Assume that $c = (u'_1, \cdots, u'_k)$. Since the stars in χ are defined and induced by χ_0, the following holds: If there exist $i \in \{1, \cdots, k-1\}$ and $u \notin V(c_0)$, such that $\chi(u)$ lies on g and $\Delta_{\chi_0}(u'_i, u'_{i+1}, u) \in T(\chi_0)$, then there also exists a triangle, such that the application of deformation (i) or (ii) is possible. That is: if neither (i) nor (ii) can be applied, the nodes on $c = (u'_1, \cdots, u'_k)$ are half stars along g for $i = 2, \cdots, k-1$, and $u'_1 = v_n$, $u'_k = w_m$ are no half stars and not degenerated. c defines the second part of the cycle c_0; this finishes the proof of Fact 2.

The construction of c_0 is illustrated in Fig. 16. $U_\chi(e_0)$ is now defined as

$$U_\chi(e_0) := \Big(\bigcup_{v \in V_0} U_\chi(v) \Big) \cup \Big(\bigcup_{v \text{ inside } c_0} U_\chi(v) \Big).$$

The following is clear without proof.

FACT 3. *Let* $\Delta \in T(\chi_0)$ *with* $\operatorname{int}(\varphi(\Delta)) \neq \varnothing$. *Then* $\varphi | \Delta$ *is bijective.*

In the next remark we consider curves in $C_{\rho(c)}$ and their inverse images under φ.

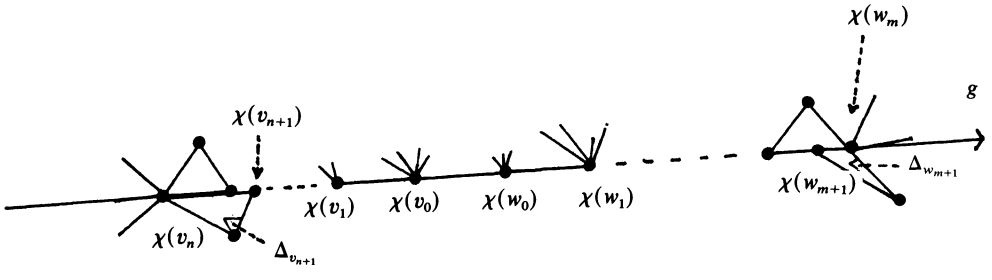


FIG. 15

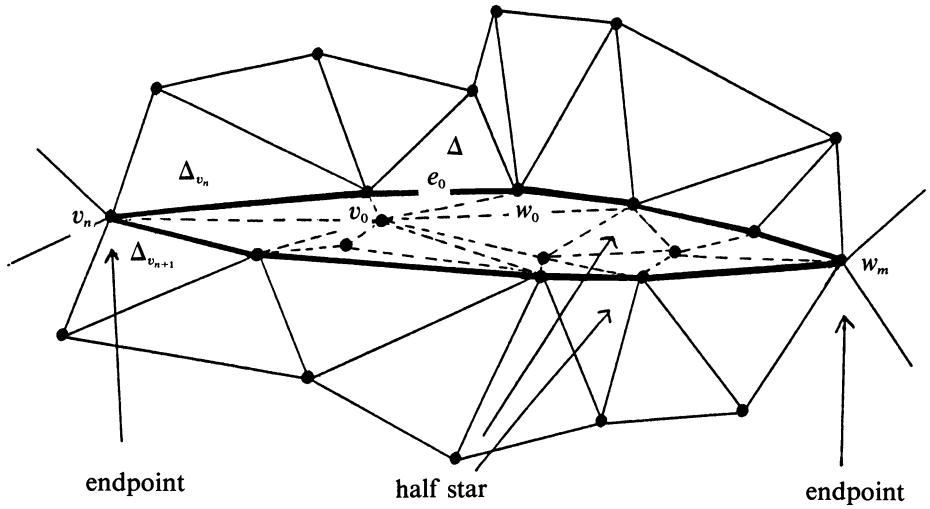


FIG. 16. --- *denotes the triangles, which collapse on g in* χ; —— *denotes the cycle* c_0.

With the help of Facts 2 and 3 we conclude the following:

FACT 4. *Let* $\bar{\gamma}$ *be a simple curve in the interior of* $C_{\rho(c)}$ *from* $\bar{x}$ *to* $\bar{y}$ *with the following properties*:

(i) $\{\bar{x}, \bar{y}\} \cap \chi(E) = \emptyset$ *and* $\bar{\gamma} \cap \chi(V) = \emptyset$.

(ii) *Let* $\Delta \in T(\chi_0)$ *with* $\operatorname{int}(\varphi(\Delta)) = \emptyset$, *then* $\operatorname{card}(\bar{\gamma} \cap \varphi(\Delta)) \leqq 1$.

Let $x \in \varphi^{-1}(\bar{x})$. *Then there exists exactly one simple curve* γ *with starting point* x *and* $\varphi(\gamma) = \bar{\gamma}$.

Proof of Fact 4. Let $x \in \Delta_1 \in T(\chi_0)$ $(\varphi(x) = \bar{x})$. Then we have a situation as in Fig. 17.

Consider $U_\chi(e_0)$ and use Fact 2 to get the following. There exists exactly one triangle $\Delta_2 \in T(\chi_0)$ with

$$\varphi(\Delta_2) \in U_\chi(e_0), \quad \operatorname{int}(\varphi(\Delta_2)) \neq \emptyset, \quad \varphi(\Delta_2) \cap \varphi(\Delta_1) \cap \bar{\gamma} = \bar{z} \neq \emptyset.$$

γ starts in x and is uniquely determined in Δ_1 (see Fact 3); we call this part of γ γ_1. γ_1 runs from x to a point $z_1 (\not\in \chi_0(V))$ on the boundary of Δ_1 with $\varphi(z_1) = \bar{z}$. There exists exactly one point z_2 on the boundary of Δ_2 with $\varphi(z_2) = \bar{z}$. The only way to continue Δ_1 is to use the uniquely determined simple curve γ_2, which starts in z_1, runs inside the cycle c_0 (which contains collapsing triangles according to Fact 2), ends in z_2 and whose image is $\bar{z}$. In Δ_2 we are then able to continue the curve without any problems. So Fact 4 is proved for the initial case. By induction we finish the proof.

FACT 5. *There exists a point* $\bar{y} \in C_{\rho(c)}$ *with the following properties*:

(i) $\operatorname{card}(\varphi^{-1}(\bar{y})) = 1$;

(ii) $\bar{y} \cap \chi(E) = \emptyset$.

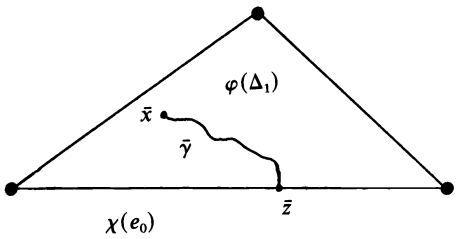


FIG. 17

Proof of Fact 5. Choose $v \in V_c$ with $e_1, e_2 \in E_c \cap E(v)$ and $\measuredangle(\chi(e_1), \chi(e_2)) \neq \pi$. Then there exists $\varepsilon > 0$ with: $\forall \bar{y} \in C_{\rho(c)}$ with $|\bar{y} - \chi(v)| < \varepsilon$ and $\bar{y} \cap \chi(E) = \emptyset$: card $(\varphi^{-1}(\bar{y})) = 1$.

With Facts 1, 4 and 5 we are able to prove Lemma 3.

Assume the following (in contradiction to the definition of quasiplanar): Let $\bar{x} \in C_{\rho(c)}$. There are $\Delta_1, \Delta_2 \in T(\chi_0)$ with $\bar{x} \in \text{Int}(\varphi(\Delta_i))$, $x_i \in \text{Int}(\Delta_i)$, $\varphi(x_i) = \bar{x}$ $(i = 1, 2)$. (We may assume w.l.o.g., that $\bar{x} \cap \chi(E) = \emptyset$.) Choose a point $\bar{y} \in C_{\rho(c)}$ according to Fact 5 and then a curve $\bar{\gamma}$ from $\bar{x}$ to $\bar{y}$, which fulfills the conditions of Fact 4. (This is possible without any difficulties!) According to Fact 4 there exists exactly one simple curve γ_i with $\varphi(\gamma_i) = \bar{\gamma}$, which starts in x_i $(i = 1, 2)$. On the other hand, there exists exactly one curve γ starting in y with $\varphi(\gamma) = \bar{\gamma}$. This gives $\gamma_1 = \gamma_2$, hence $x_1 = x_2$; that is, the interior of two triangles, which do not degenerate, is disjoint.

This concludes the proof of Lemma 3 together with Fact 1. □

Remark. In Lemma 4 we will implicitly prove the inverse direction of Lemma 3, but this is not necessary for the proof of the nonoptimality of nonquasiplanar layouts.

With the help of the characterization Lemma 3 and an invariant $I(G_{\rho(c)})$ of G, which may be computed by calculating the angles of all triangles meeting in a point $\chi(v)$, we will show that "nonquasiplanarity" implies the existence of convex points.

The *angle* $\measuredangle(\chi, v)$ *in* v *for* χ is $\measuredangle(\chi, v) := \sum_{\Delta \in U_\chi(v)} \measuredangle(\Delta, v)$ with $\measuredangle(\Delta, v)$ is the angle of Δ in $\chi(v)$, if int $(\Delta) \neq \emptyset$; if int $(\Delta) = \emptyset$, $\measuredangle(\Delta, v)$ is defined as follows: Let $\chi(e_1), \chi(e_2), \chi(e_3)$ compose Δ

$$\measuredangle(\Delta, v) = \begin{cases} \pi & \text{if } \exists i, j \in \{1, 2, 3\},\ i \neq j \text{ with } \chi(e_i) \cap \chi(e_j) = \chi(v), \\ 0 & \text{otherwise.} \end{cases}$$

Remark. For $v \in V_{\text{in}}$ we get: $\measuredangle(\chi_0, v) = 2\pi$.

If we look at the example at the beginning of this section, we see that $\measuredangle(\chi, v)$ may be greater or less than 2π for points v in V_{in}.

We are now ready to define the *invariant* $I(G_{\rho(c)})$:

$$I(G_{\rho(c)}) := (2n + k - 2)\pi \quad \text{where } k = \text{card}(V_c),\ n = \text{card}(V_{\text{in}}).$$

$I(G_{\rho(c)})$ may be computed in different ways, as we will see in the following claim.

CLAIM 1. *Let* $N = \text{card}(T(\chi_0))$. *Then*

$$\begin{aligned} I(G_{\rho(c)}) &= \sum_{v \in V_{\text{in}}} 2\pi + (k-2)\pi \\ &= \sum_{v \in V_{\text{in}}} \measuredangle(\chi_0, v) + \sum_{v \in V_c} \measuredangle(\chi_0, v) \\ &= \sum_{v \in V} \measuredangle(\chi_0, v) \\ &= N\pi \\ &= \sum_{v \in V} \measuredangle(\chi, v). \end{aligned}$$

The correctness of Claim 1 is clear, since the number of "triangles" of χ_0 and χ is the same. (Note that for a collapsed triangle exactly one vertex is charged π and the others are charged 0.)

We make the following observations:

CLAIM 2.

(i) $\measuredangle(\chi_0, v) \leqq \measuredangle(\chi, v)$ *for* $v \in V_c$.

(ii) *If there exists a* $v_1 \in V$ *with* $\sphericalangle(\chi_0, v_1) < \sphericalangle(\chi, v_1)$, *then there exists a* $v_2 \in V_{\text{in}}$ *with* $2\pi = \sphericalangle(\chi_0, v_2) > \sphericalangle(\chi, v_2)$.

(iii) *If* $\sphericalangle(\chi, v) < 2\pi$ *for* $v \in V_{\text{in}}$, *then* $\chi(v)$ *is a convex point.*

Proof.

(i) Clear;

(ii) Follows from Claim 1 and (i);

(iii) Follows directly from a consideration of the triangles composing $U_\chi(v)$.

With the help of Claim 2 and Lemma 3 we succeed in finding convex points.

LEMMA 4. *Let* χ *be a* $\rho(c)$*-bounded layout of* G *with* $|\chi(e)| > 0$, $\forall e \in E$. *Then we have*:

$$\chi \text{ is not quasi-}\rho(c)\text{-planar}$$
$$\Rightarrow$$
$$\chi \text{ has a convex point.}$$

Proof. Apply Lemma 3 to get the following: If there exists $v \in V$ with "$S_\chi(v)$ not defined," we have the existence of a convex point in v. (See (iii) in the definition of "star.") So we may assume: there exists $v \in V$ with the following: $S_{\chi_0}(v)$ does not induce $S_\chi(v)$.

Case 1: $v \in V_c$. We conclude $\sphericalangle(\chi_0, v) < \sphericalangle(\chi, v)$ and with Claim 2 we get the existence of a convex point.

Case 2: $v \in V_{\text{in}}$.

(i) $\sphericalangle(\chi, v) \neq 2\pi \Rightarrow_{\text{Claim 2}}$ there exists a convex point;

(ii) $\sphericalangle(\chi, v) = 2\pi$.

Without loss of generality assume that in v there is no convex point of χ.

(a) φ "folds" $U_{\chi_0}(v)$ along a line segment of a straight line g. The situation is illustrated in Fig. 18.

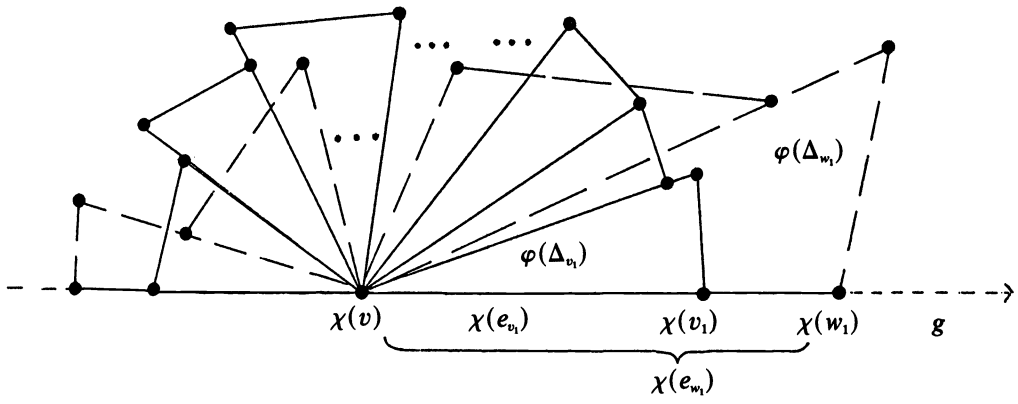


FIG. 18. $\chi(v)$ *is called a folding point along* g.

Consideration of the angle at $v_1(w_1)$ for χ either directly implies the existence of a convex point or we get the following: v_1 is a half star along g and w_1 is an inverse half star along g with respect to χ (or vice versa). In this case we start with the construction of a cycle c_0, analogously to the cycle c_0 in the proof of Fact 2 (for

illustration see Fig. 19): at the beginning we have the point $\chi(v)$, the edges $e_{v_1} = (v, v_1)$, $e_{w_1} = (v, w_1)$ and the triangles Δ_{v_1}, Δ_{w_1}, during the construction we find:

– Either a node $v' \in V_{\text{in}}$ with: v' is a convex point or $\sphericalangle(\chi, v') \neq 2\pi$ ($\Rightarrow$ existence of a convex point);

– Or $v' \in V_c$ with: $\sphericalangle(\chi_0, v') < \sphericalangle(\chi, v')$ ($\Rightarrow$ existence of a convex point);

– Or we inductively get sequences $(v, v_1, v_2, \cdots, v_n)$ and $(v, w_1, w_2, \cdots, w_m)$ with: $v, v_n = w_m$ are folding points along g, $v_1, v_2, \cdots, v_{n-1}$ are half stars along g, and $w_1, w_2, \cdots, w_{m-1}$ are inverse half stars along g (with respect to χ). The cycle c_0 is given by $(v, v_1, \cdots, v_n = w_m, w_{m-1}, \cdots, w_2)$ and all nodes inside c_0 have degenerated stars in χ.

Since χ has a convex boundary $\rho(c)$, we inductively find (by construction of new cycles c_0):

– Either a point $v' \in V_{\text{in}}$ with: v' is a convex point or $\sphericalangle(\chi, v') \neq 2\pi$;

– Or a point $v' \in V_c$ with: $\sphericalangle(\chi_0, v') < \sphericalangle(\chi, v')$.

So we finally get the existence of a convex point in Case (a).

(b) There is the following case left: $\sphericalangle(\chi, v) = 2\pi$ and $\chi(S_{\chi_0}(v))$ is a counterclockwise enumeration of the edges in $\chi(v)$. Choose a path of edges from $\chi(v)$ to the boundary $\rho(c)$. Then one finds a node w on this path such that: $\chi(S_{\chi_0}(w)$ is neither a clockwise nor a counterclockwise enumeration of the edges in $\chi(w)$. This gives the existence of a convex point according to Case 1, 2(i), 2(ii)(a). □

With the help of Lemma 4 we get the inverse direction of Lemma 3 and therefore have the following lemma.

LEMMA 5. *Let χ be a $\rho(c)$-bounded layout of G with $|\chi(e)| > 0$, $\forall e \in E$. Then we have:*

$$\chi \text{ is quasi-}\rho(c)\text{-planar,}$$
$$\Leftrightarrow$$
$$\forall v \in V\colon S_\chi(v) \text{ is defined and } \chi(S_{\chi_0}(v)) = S_\chi(v).$$

We generalize Lemma 4 to layouts χ, where $|\chi(e)|$ is not necessarily greater than 0, $\forall e \in E$.

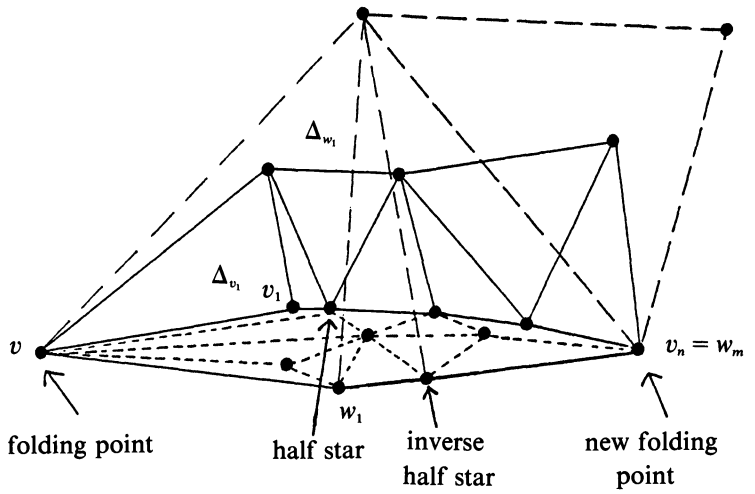


FIG. 19. — *denotes the triangles collapsed on g*; --- *denotes the "folded" triangles.*

LEMMA 6. *Let χ be a $\rho(c)$-bounded layout of a $\rho(c)$-triangulated graph G. Then we have*:

$$\chi \text{ is not quasi-}\rho(c)\text{-planar,}$$
$$\Rightarrow$$
$$\chi \text{ has a convex point.}$$

Proof. If there exists an edge $e=(v, w)$, which collapses to a point under φ, we consider the $\rho(c)$-triangulated graph, which results from identifying v and w. With induction and Lemma 4 we finish the proof.

It remains to prove our results for general graphs with fixed boundary (not necessarily $\rho(c)$-triangulated). First we have to give the general definition of quasi-planar. Intuitively spoken quasiplanarity of a layout $\chi(G)$ in this case means the following: all connected components of G collapse to points, all biconnected components collapse to straight line segments, the layout of the "remaining" graph is quasiplanar in the sense of our former definition. More formally, let G be any $\rho(c)$-planar graph with fixed boundary, χ a layout of G.

χ has the *a-point property*, iff the following hold:

– Let v be any articulation point of G (then G is split into connected components by v), let $G_v=(V_v, E_v)$ be any of the resulting subgraphs with $c \not\subset G_v$. Then $\chi(w)=\chi(v)$, $\forall w \in V_v$. (See the example in Fig. 20!)

χ has the *a-pair property*, iff the following hold:

– Let (v_1, v_2) be any articulation pair of G (then G is split into connected components by (v_1, v_2)), let $G_{(v_1,v_2)}=(V_{(v_1,v_2)}, E_{(v_1,v_2)})$ be any of the resulting subgraphs with $c \not\subset G_{(v_1,v_2)}$. Then we have the following: If $E_{(v_1,v_2)} \cap E_c = \varnothing$ or $v_1, v_2 \in V_c$ with $\overline{\rho(v_1)\rho(v_2)} \subset \rho(c)$, then $\chi(w) \in \overline{\chi(v_1)\chi(v_2)}$, $\forall w \in V_{(v_1,v_2)}$. (See the example in Fig. 21.)

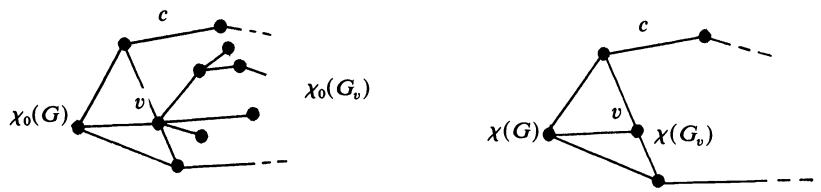


FIG. 20

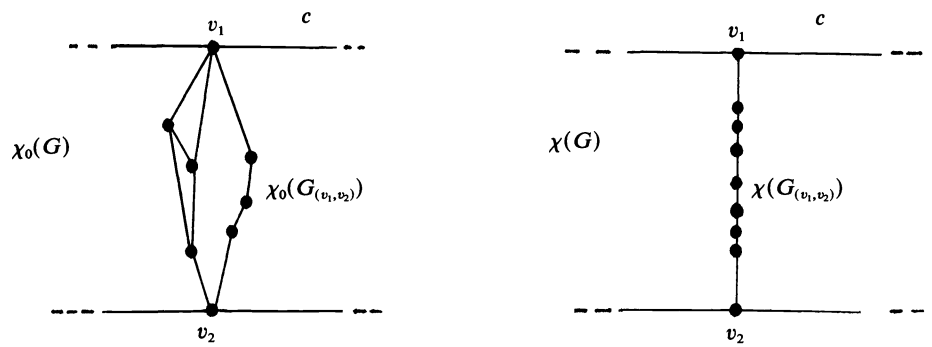


FIG. 21

With the help of these definitions we "construct" the reduced graph G_{red} of G in three steps:

(i) Define G_{red} to be the maximal subgraph of G with:

$- c \subset G_{red}$,

$- \exists$ articulation point in G_{red};

(ii) Let (v_1, v_2) be an articulation pair in G_{red}, consider the resulting components $G_{(v_1,v_2)}$ and replace all components with $E_{(v_1,v_2)} \cap E_c = \emptyset$ or $v_1, v_2 \in V_c$ and $\overline{\rho(v_1)\rho(v_2)}$ lies on $\rho(c)$ by an edge $e = (v_1, v_2)$, unless e is already an edge of G_{red};

(iii) Repeat (ii) as long as possible.

The resulting graph is called the *reduced graph* G_{red} *of* G. Any layout χ of G, which fulfills the a-point and a-pair property, may be interpreted as a layout of the reduced graph G_{red}. G_{red} is a $\rho(c)$-planar graph with fixed boundary, and the faces of G_{red} are uniquely determined.

DEFINITION. Let G be a $\rho(c)$-planar graph with fixed boundary. A $\rho(c)$-bounded layout χ of G is *nearly quasi-$\rho(c)$-planar*, iff:

(i) χ has the a-point and a-pair property;

(ii) Let F_1, F_2 be faces of G_{red} in χ, then $F_1 \cap \text{int}(F_2) = \emptyset$;

(iii) $\nexists v \in V$: $\chi(v)$ is a peak ("peak" is defined analogously to the triangulated case).

As a generalization of Lemma 6 we get Lemma 7.

LEMMA 7. *Let χ be a $\rho(c)$-bounded layout of a $\rho(c)$-planar graph G. Then we have*:

χ is not nearly quasi-$\rho(c)$-planar
$\Rightarrow$
χ has a convex point.

Proof.

Case 1. (i), (iii) of the above definition is not fulfilled $\Rightarrow$ we at once get the existence of a convex point.

Case 2. (ii) of the above definition is not fulfilled $\Rightarrow$ let G_T denote any triangulation of G_{red}, let χ_T be the layout of G_T induced by χ, then χ_T is not quasi-$\rho(c)$-planar, which gives the existence of a convex point in this case, also.

Nearly quasi-$\rho(c)$-planar layouts may have faces whose boundaries are not simple or do not define convex sets. These layouts cannot be optimal. Therefore we define:

DEFINITION. Let G be a $\rho(c)$-planar graph with fixed boundary. A $\rho(c)$-bounded layout χ of G is *quasi-$\rho(c)$-planar*, iff

(i) χ is nearly quasi-$\rho(c)$-planar;

(ii) Let F be a face of G_{red} with $\text{int}(F) \neq \emptyset$, then F is a convex set and ∂F is simple. Let F be a face of G_{red} with $\text{int}(F) = \emptyset$, then ∂F consists of two identical straight line segments or a point. (F is "degenerated convex.")

We get a more precise version of Lemma 7.

LEMMA 8. *Let χ be a $\rho(c)$-bounded layout of a $\rho(c)$-planar graph G. Then we have*:

χ is not quasi-$\rho(c)$-planar
$\Rightarrow$
χ has a convex point.

Proof. Since an exact proof requires the same methods as introduced before, we do not go into detail and only indicate the general proceeding: Assume that (ii) in the above definition of "quasiplanar" is not fulfilled. Then we either directly get the existence of a convex point or we can choose a triangulation of G, which after

application of Lemmas 4, 6 and 7 gives the existence of a convex point for the triangulated case. This immediately implies the existence of a convex point in the layout χ.

Lemma 8 gives an interesting characterization of nonplanar layouts of planar graphs (with fixed boundary). Either they are quasiplanar, or there exists a convex point.

We summarize our results in Theorem 3.

THEOREM 3. *Let $G_{\rho(c)}$ be a connected $\rho(c)$-planar graph with fixed boundary and $f: \mathbb{R}^+ \to \mathbb{R}^+$ continuous and increasing. Then an optimal layout of G with respect to $\rho(c)$ and f is quasi-$\rho(c)$-planar.*

4.2. Planarity and optimality. We restrict to triconnected planar graphs with fixed boundary and show that the optimal layout is planar convex. We need some preparations:

TECHNICAL LEMMA. *Assume f to be nondecreasing, strictly convex, twice differentiable, $f(0)=f'(0)=0$. Let $N>0$, $y>0$, $x\geqq 0$, $0<\alpha\leqq\pi/2$, for $\varepsilon>0$ define $x':=(x^2+\varepsilon^2)^{1/2}$, $y':=(y^2+\varepsilon^2-2y\varepsilon\cos\alpha)^{1/2}$. Assume a situation as in Fig. 22. Then there exists $\bar{\varepsilon}>0$, such that:*

$$\forall 0<\varepsilon\leqq\bar{\varepsilon}\colon N(f(x')-f(x))<f(y)-f(y').$$

Proof.

Case 1. $x>0$. If we choose ε small enough, we get with the help of the properties of f and the Mean Value Theorem

$$N(f(x')-f(x))<Nf'(2x)(x'-x), \qquad f'(y/2)(y-y')<f(y)-f(y').$$

Therefore it suffices to show that

$$\exists\bar{\varepsilon}>0\colon \forall\varepsilon\leqq\bar{\varepsilon}\colon M(x'-x)<y-y' \quad \text{with } M:=\frac{Nf'(2x)}{f'(y/2)}>0.$$

An exact calculation shows that there exists a polynomial P with

$$\begin{aligned} &M(x'-x)<y-y'\\ &\Leftrightarrow Mx+y>M(x^2+\varepsilon^2)^{1/2}+(y^2+\varepsilon^2-2y\varepsilon\cos\alpha)^{1/2}\\ &\Leftrightarrow Mx+y>\varepsilon\cdot P(\varepsilon). \end{aligned}$$

This finishes the proof of Case 1.

Case 2. $x=0$. We have to show that $Nf(\varepsilon)<f(y)-f(y')$. As in Case 1 we get, for small enough ε,

$$Nf(\varepsilon)<Nf'(\varepsilon)\varepsilon, \qquad f'(y/2)(y-y')<f(y)-f(y').$$

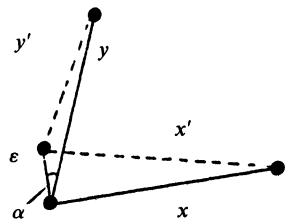


FIG. 22

Therefore it suffices to show that

$$\exists \bar{\varepsilon} > 0: \forall \varepsilon \leqq \bar{\varepsilon}: Mf'(\varepsilon)\varepsilon < y - y' \quad \text{with } M := \frac{N}{f'(y/2)} > 0.$$

We have the following:

$$\begin{aligned}
&Mf'(\varepsilon)\varepsilon < y - y' \\
&\Leftrightarrow y - Mf'(\varepsilon)\varepsilon > (y^2 + \varepsilon^2 - 2y\varepsilon \cos \alpha)^{1/2} \\
&\Leftrightarrow y^2 - 2Myf'(\varepsilon)\varepsilon + M^2(f'(\varepsilon))^2\varepsilon^2 > y^2 + \varepsilon^2 - 2y\varepsilon \cos \alpha \\
&\Leftrightarrow 2y \cos \alpha > \varepsilon(1 - M^2(f'(\varepsilon))^2) + 2Myf'(\varepsilon).
\end{aligned}$$

Because of $\lim_{\varepsilon \to 0} f'(\varepsilon) = 0$, the proof is finished. □

The technical lemma gives us a method to improve layouts. In order to be able to apply this lemma, we assume from now on, that f fulfills the following conditions as above: f is nondecreasing, strictly convex, twice differentiable and $f(0) = f'(0) = 0$.

LEMMA 9. *Let χ be $\rho(c)$-bounded layout of G, which is quasi-$\rho(c)$-planar but not $\rho(c)$-planar convex. Then χ is not optimal.*

Proof. We first consider two special layout situations, which will later help us to prove that a layout χ with degenerated convex faces cannot be optimal.

(i) Assume a situation as in Fig. 23. F is a face collapsed to a line and g is the straight line defined by F. There exists a point v on F with:

$$(*) \qquad \chi(E(v)) \cap (H_1(g) \backslash g) \neq \varnothing, \qquad \chi(E(v)) \cap (H_2(g) \backslash g) = \varnothing.$$

Now we use the technical lemma to show that in this situation the layout is not optimal. Let $\{v_1, \cdots, v_n\} \subset V(v)$ be the vertices with $\chi(v_i) \in (H_1(g) \backslash g)$. We know that $n \geqq 1$ because of $(*)$. Let $\{w_1, \cdots, w_m\} \subset V(v)$ be the vertices with $\chi(w_i) \in g$. We know that $m \geqq 2$ because of v on F.

We choose $\varepsilon > 0$ and change the placement of $\chi(v)$ by ε to a direction vertical to g. We call the new point in $H_1(g)$ $\bar{\chi}(v)$. (See Fig. 24.) Then we have

$$\begin{aligned}
&\sum_{i=1}^{m} (f(g(\chi(w_i), \bar{\chi}(v))) - f(g(\chi(w_i), \chi(v)))) \\
&\qquad < m \cdot \max_{i=1,\cdots,m} \{(f(g(\chi(w_i), \bar{\chi}(v))) - f(g(\chi(w_i), \chi(v)))\} \\
&\qquad < f(g(\chi(v_1), \chi(v))) - f(g(\chi(v_1), \bar{\chi}(v))) \\
&\qquad\qquad \text{(technical lemma with } \varepsilon \text{ sufficiently small)} \\
&\qquad < \sum_{i=1}^{n} (f(g(\chi(v_i), \chi(v))) - f(g(\chi(v_i), \bar{\chi}(v)))).
\end{aligned}$$

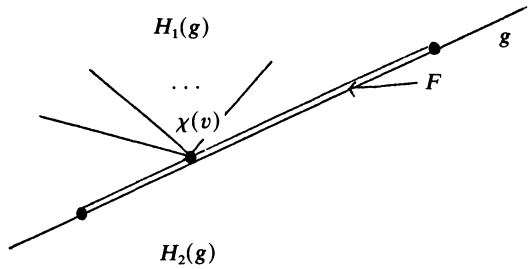


FIG. 23

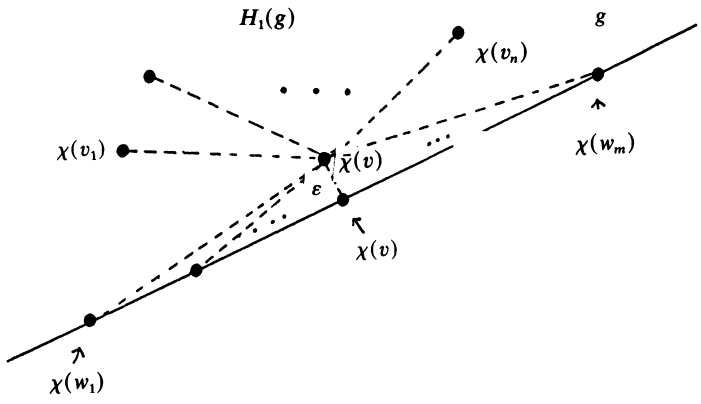


FIG. 24

This means that there exists an $\varepsilon > 0$ such that

$$\sum_{w \in V(v)} f(g(\chi(w), \bar{\chi}(v))) < \sum_{w \in V(v)} f(g(\chi(w), \chi(v))).$$

Therefore situation (i) can never occur in an optimal layout.

(ii) Assume the following situation:

$$\chi(v_1) = \chi(v_2) \quad \text{for } v_1 \in V(v_2)$$

and there does not exist a straight line g such that $\chi(E(v_1)) \subset g$ or $\chi(E(v_2)) \subset g$ (see the example in Fig. 25). The "part of the circle in $\chi(v_1)$ from $(\chi(w_1), \chi(v_1))$ in clockwise orientation to $(\chi(w_2), \chi(v_1))$" is called *sector* $S(w_1, v_1, w_2)$. Since χ is quasi-$\rho(c)$-planar, there exist points $\chi(w_1)$, $\chi(w_2)$, $\chi(w_3)$, $\chi(w_4)$, such that

(a) $w_1, w_2 \in V(v_1)$, $w_3, w_4 \in V(v_2)$,

(b) $\chi(E(v_1)) \subset S(w_1, v_1, w_2)$, $\chi(E(v_2)) \subset S(w_3, v_2, w_4)$,

(c) $\text{int}\,(S(w_1, v_1, w_2)) \cap \text{int}\,(S(w_3, v_2, w_4)) = \varnothing$.

With χ quasi-$\rho(c)$-planar it follows that the angle $\sphericalangle(w_1, v_1, w_2) \leqq \pi$, or $\sphericalangle(w_3, v_2, w_4) \leqq \pi$. Assume that $\sphericalangle(w_1, v_1, w_2) \leqq \pi$. With the help of the technical lemma we succeed in showing (analogously to (i)) that χ is not optimal. (See Fig. 26.)

It now remains to show the following for a quasi-$\rho(c)$-planar layout χ, which is not $\rho(c)$-planar convex: There exists a face F respectively, points v_1, v_2, such that (i)

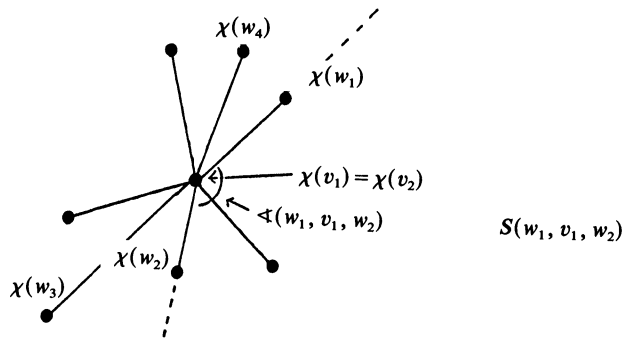


FIG. 25

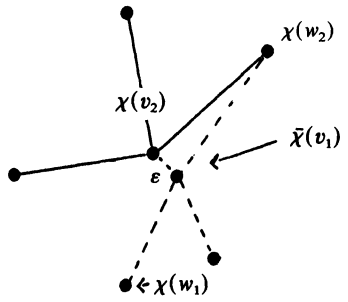


FIG. 26

or (ii) is applicable. Assume all faces of χ which are degenerated to a line do not allow application of (i). We have to show that there exist points which allow application of (ii). Let F be a face of χ collapsed to a line g. If the application of (ii) at one of the "endpoints" of F is possible, the proof is finished. Otherwise, we get the existence of a set S of faces (incl. F) degenerated to g and nested into each other. One of the endpoints of this set then allows the application of (ii). (See Fig. 27!) More exactly, S is defined as follows:

(i) $F \in S$;

(ii) All faces in S are degenerated to the line g;

(iii) $\bigcup_{F' \in S} F' := g_S$ is a (connected) straight line segment of g;

(iv) Exactly the endpoints P_1, P_2 of g_S correspond to nodes v_i $(i = 1, 2)$ of G with $\chi(E(v_i)) \not\subset g$ $(i = 1, 2)$.

From the quasiplanarity of χ we know that the endpoints of g_S do not correspond to peaks. We conclude that P_1 or P_2 must be the image of at least two vertices of G. Otherwise we would have a contradiction to the triconnectivity of G. So we conclude that (ii) is applicable. Therefore, we may assume that there are no faces of χ collapsed to a line. If there rest vertices v_1, v_2 with $\chi(v_1) = \chi(v_2)$, then situation (ii) is applicable. So the proof of Lemma 9 is finished. □

If we combine Theorem 3 and Lemma 9 we get Theorem 4.

THEOREM 4. *Let $G_{\rho(c)}$ be a triconnected $\rho(c)$-planar graph with fixed boundary and let $f: \mathbb{R}^+ \to \mathbb{R}^+$ be a nondecreasing, strictly convex, twice differentiable function with $f(0) = f'(0) = 0$. Then the optimal layout of G with respect to $\rho(c)$ and f is $\rho(c)$-planar convex.*

COROLLARY. *A careful reading of the proof of Lemma 9 gives the following: The triconnectivity of G is not necessary in Theorem 4; $G = G_{\text{red}}$ is sufficient.*

5. Concluding remark. (i) If one considers functions f_e, $\forall e \in E$ instead of one function f and defines the cost of a layout χ as $|\chi| := \sum_{e \in E} f_e(|\chi(e)|)$, all the results remain valid. (Especially layouts of "weighted graphs with fixed boundary" may be considered.)

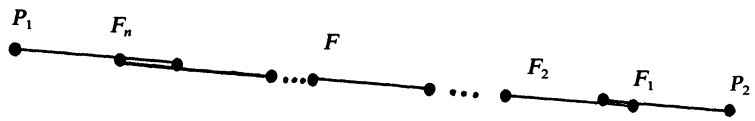


FIG. 27

(ii) If one considers "the limit" of the optimal layouts of a graph with respect to the l_p-metric for $p=2, 3, \cdots$, one gets a layout, which minimizes the maximal length of an edge of the graph and balances the length of the remaining ones. With the help of Theorems 3 and 4 we are able to prove the quasiplanarity of this layout, also (see [BeOs]).

Appendix. Apart from the existence and properties of optimal layouts their construction, too, is a point of interest. We studied approximation algorithms for the construction of the optimal layout with respect to cost functions $f(x)=x^p$. The cases $f(x)=x^2$ and $f(x)=x^p$ $(p>2)$ have to be distinguished.

In the case $p=2$ the solution of the optimization problem is equivalent to the solution of two (sparse) linear equation systems. This at once gives a polynomial time algorithm for the computation of the exact solution. On the other hand, Jacobi or Gauss–Seidel relaxation can be applied and converge to the optimal layout. (Iteration steps may yield nonplanar intermediate layouts. Therefore as termination criterion a planarity test for the approximation is needed. Such a test can be done in linear time by use of Lemma 2. For details see [GrHo].)

The complexity of the approximation algorithms depends on the spectral radius of the relevant matrix. There exists a class of graphs (see [StBu, p. 257]), for which the optimal layout can be computed in time $O(n^2)$ (n = number of the nodes), if the precision of the calculation is given. Experimental results yield linear time for the considered graphs, but we still did not succeed in the exact computation of the spectral radius for these cases. In the case $p>2$ methods of steepest descent were studied. Details for both cases ($p=2$ and $p>2$) together with the corresponding program listing can be found in [Gr]. With the help of multigrid methods it is possible to approximate the solution of the optimization problem for $p=2$ in linear time [St]. Whether this can be extended to $p>2$ is not known at this time. (For further information about multigrid methods see [HaTr].)

We finish with two examples: Consider the "net" in Fig. 28, which recursively defines an n-bit adder. We apply the given recursion for $n/2$ down to 2 (Ad_2 corresponds to a full adder) and thus finally we get a graph corresponding to the switching circuit

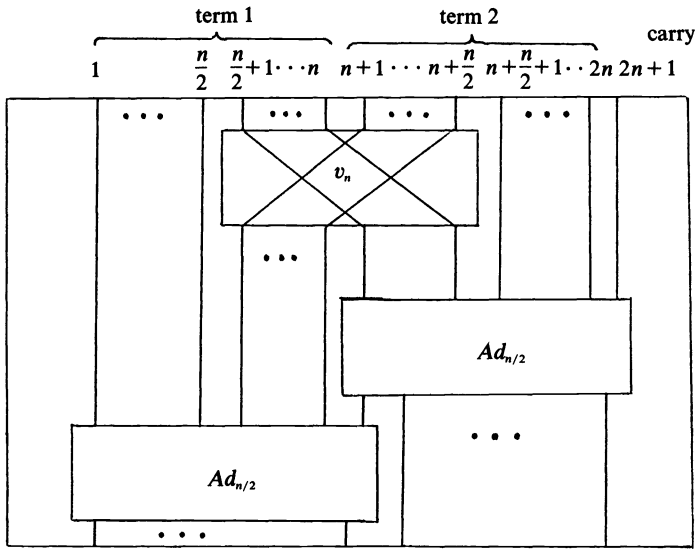


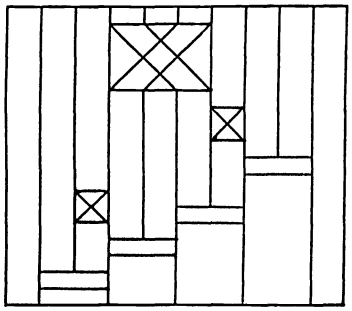
FIG. 28

of an n-bit adder. If the recursion is given, the above process is done automatically in the logical-topological level of the design system CADIC (for details see [BHKM]). The resulting graph is input for the optimization algorithms.

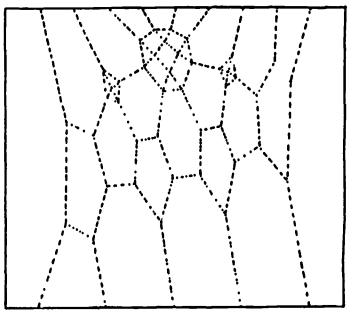
A first very small example is given by the optimal layouts of a 4-bit adder defined by the net of Fig. 28. (See Fig. 29.)

The approximation algorithms were tested for many (more complex) circuits (for example: carry-look-ahead adder, conditional-sum-adder, multiplier, memories, combinational logic, ...).

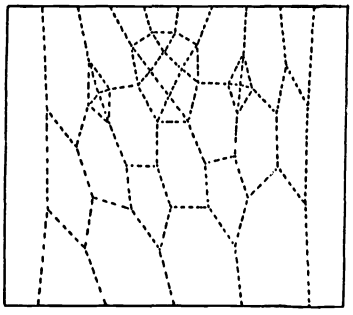
We finish by giving the l_2-optimal layout of a fast multiplier presented in [LuVu]. The multiplier can be described with the help of four recursive equations in a manner



(a)



(b)



(c)

FIG. 29. (a) *Layout of a graph with fixed boundary corresponding to a 4-bit adder*; (b) *Optimal layout with respect to* $f(x) = x^2$; (c) *Optimal layout with respect to* $f(x) = x^4$.

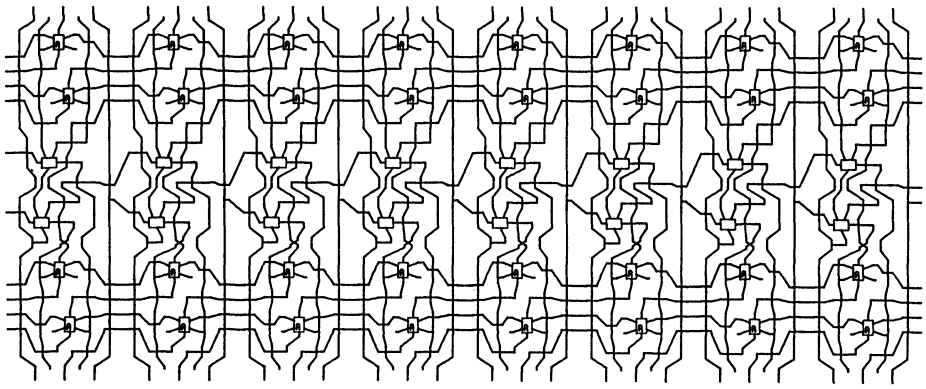


FIG. 30. *Expansion of the recursively defined network for the fast multiplier.*

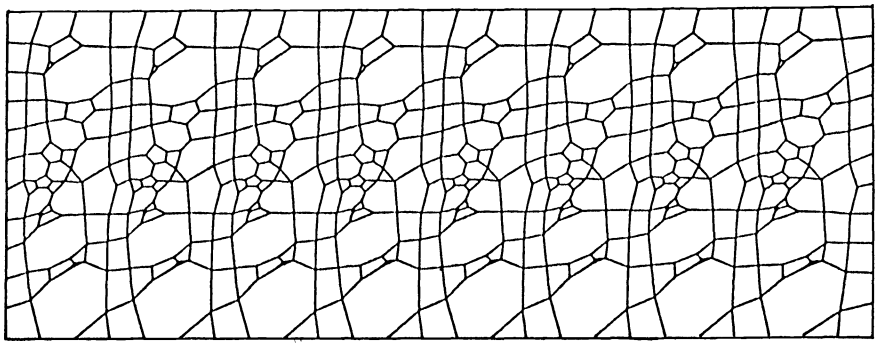


FIG. 31. *Optimal layout with respect to* $f(x) = x^2$.

analogous to the above adder (see [BHKM]). Here we start directly with the graph with fixed boundary constructed out of the equations (Figs. 30 and 31).

Acknowledgments. The authors thank U. Groh for many helpful comments and a careful reading of the manuscript.

REFERENCES

[Be] B. BECKER, *Über die kreuzungsfreie, rechtwinklige Einbettung von gewichteten Graphen in die Ebene*, Ph.D. dissertation, Universität des Saarlandes, Saarbrücken, 1982.

[BeOs] B. BECKER AND H. G. OSTHOF, *Layouts with wires of balanced length*, Proc. Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, 1985, pp. 21-31.

[BHKM] B. BECKER, G. HOTZ, R. KOLLA AND P. MOLITOR, *Ein* CAD-*System zum Entwurf integrierter Schaltungen*, Technical Report, 16/1984, SFB 124 B1, Universität des Saarlandes, Saarbrücken.

[Br] M. A. BREUER, *Design Automation of Digital Systems, Vol.* 1: *Theory and Techniques*, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Bry] R. BRYANT, ed., *Third Caltech Conference on* VLSI, Springer-Verlag, Berlin, 1983.

[Do] W. E. DONATH, *Complexity theory and design automation*, Proc. 17th Design Autom. Conf., New York, IEEE Computer Soc. Press, 1980, pp. 412-419.

[Fi] C. J. FISK, D. L. CASKEY AND L. L. WEST, *Automated circuit card etching layout*, Proc. IEEE, 55, 1967, pp. 1971-1982.

[FiPa] M. J. FISCHER AND M. S. PATTERSON, *Optimal tree layout* (*preliminary version*), Proc. 12th Annual ACM Symposium on the Theory of Computing, 1980, pp. 177-189.

[Gr] U. GROH, *Optimale Einbettungen von Graphen mit festem Rand*, Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1983.

[GrHo] U. GROH AND G. HOTZ, *Ein Planaritätstest für planar-konvexe Grapheneinbettungen mit linearer Komplexität*, Beiträge zur Algebra und Geometrie, 18 (1984), pp. 191-200.

[HaTr] W. HACKBUSCH AND U. TROTTENBERG, eds., *Multigrid methods. Proceedings of the conference held at Köln-Porz*, November 23-27, 1981, Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, 1981.

[Ho] G. HOTZ, *Einbettung von Streckenkomplexen in die Ebene*, Math. Annalen, 167 (1966), pp. 214-223.

[HBKM] G. HOTZ, B. BECKER, R. KOLLA AND P. MOLITOR, *Ein logisch-topologischer Kalkül zur Konstruktion von integrierten Schaltkreisen*, Informatik Forschung und Entwicklung, 1 (1986), pp. 38-47.

[LuVu] W. K. LUK AND J. VUILLEMIN, *Recursive Implementation of Optimal Time* VLSI *Integer Multipliers*, Proc. IFIP Congress 83, Amsterdam, North-Holland, 1983, pp. 155-168.

[Re] K. REIDEMEISTER, *Einführung in die kombinatorische Topologie*, Vieweg, Braunschweig, 1932.

[SaBh] S. SAHNI AND A. BHATT, *The complexity of design automation*, Proc. 19th Design Autom. Conf., New York, IEEE Computer Soc. Press, 1980, pp. 402-411.

[SaGo] S. SAHNI AND T. GONZALEZ, *P-complete approximation problems*, J. Assoc. Comput. Mach., 23 (1976), pp. 555-565.

[Schm] F. J. SCHMITT, *Rechtwinklige Verdrahtung optimaler Layouts*, Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1985.

[StBu] J. STOER AND R. BULIRSCH, *Einführung in die Numerische Mathematik* II, Springer-Verlag, Berlin, 1978.

[Schw] K. SCHWORM, *Layouts unter Berücksichtigung von Leitungs-breiten*, Diplomarbeit, Universität des Saarlandes, Saarbrücken, 1985.

[Sto] J. A. STORER, *The node cost measure for embedding graphs on the planar grid*, Proc. 12th Annual Symposium on the Theory of Computing, 1980, pp. 201-210.

[St] K. STÜBEN, *Algebraic Multigrid* (AMG): *Experiences and Comparisons*, Arbeitspapiere der GMD Bonn Nr. 23, März, 1983.

[Wh] H. WHITNEY, *A set of topological invariants for graphs*, Amer. J. Math., 55 (1933), pp. 231-235.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987

© 1987 Society for Industrial and Applied Mathematics

THREE PARTITION REFINEMENT ALGORITHMS*

ROBERT PAIGE† AND ROBERT E. TARJAN‡

Abstract. We present improved partition refinement algorithms for three problems: lexicographic sorting, relational coarsest partition, and double lexical ordering. Our double lexical ordering algorithm uses a new, efficient method for unmerging two sorted sets.

Key words. partition, refinement, lexicographic sorting, coarsest partition, double lexical ordering, unmerging, sorted set, data structure

AMS(MOS) subject classifications. 68P05, 68P10, 68Q25, 68R05

1. Introduction. The theme of this paper is partition refinement as an algorithmic paradigm. We consider three problems that can be solved efficiently using a repeated partition refinement strategy. For each problem, we obtain a more efficient algorithm than those previously known. Our computational model is a uniform-cost sequential random access machine [1]. All our resource bounds are worst-case. We shall use the following terminology throughout the paper. Let U be a finite set. We denote the size of U by $|U|$. A *partition* P of U is a set of pairwise disjoint subsets of U whose union is all of U. The elements of P are called its *blocks*. If P and Q are partitions of U, Q is a *refinement* of P if every block of Q is contained in a block of P. As a special case, every partition is a refinement of itself.

The problems we consider and our results are as follows:

(i) In § 2, we consider the *lexicographic sorting problem*: given a multiset U of n strings over a totally ordered alphabet Σ of size k, sort the strings in lexicographic order. Aho, Hopcroft and Ullman [1] presented an $O(m+k)$ time and space algorithm for this problem, where m is the total length of all the strings. We present an algorithm running in $O(m'+k)$ time and $O(n+k)$ auxiliary space (not counting space for the input strings), where m' is the total length of the *distinguishing prefixes* of the strings, the smallest prefixes distinguishing the strings from each other.

(ii) In § 3, we consider the *relational coarsest partition problem*: given a partition P of a set U and a binary relation E on U, find the coarsest refinement Q of P such that for each pair of blocks B_1, B_2 of Q, either $B_1 \subseteq E^{-1}(B_2)$ or $B_1 \cap E^{-1}(B_2) = \varnothing$. Here $E^{-1}(B_2)$ is the preimage set, $E^{-1}(B_2) = \{x \mid \exists y \in B_2 \text{ such that } xEy\}$. Kanellakis and Smolka [9] studied this problem in connection with equivalence testing of CCS expressions [14]. They gave an algorithm running in $O(mn)$ time and $O(m+n)$ space, where m is the size of E (number of related pairs) and n is the size of U. We propose an algorithm running in $O(m \log n)$ time and $O(m+n)$ space.

* Received by the editors May 19, 1986; accepted for publication (in revised form) March 25, 1987.

† Computer Science Department, Rutgers University, New Brunswick, New Jersey 08903 and Computer Science Department, New York University/Courant Institute of Mathematical Science, New York, New York 10012. Part of this work was done while this author was a summer visitor at IBM Research, Yorktown Heights, New York 10598. The research of this author was partially supported by National Science Foundation grant MCS-82-12936 and Office of Naval Research contract N00014-84-K-0444.

‡ Computer Science Department, Princeton University, Princeton, New Jersey 08544 and AT&T Bell Laboratories, Murray Hill, New Jersey 07974. The research of this author was partially supported by National Science Foundation grants MCS-82-12936 and DCR-86-05962 and Office of Naval Research contract N00014-84-K-0444.

(iii) In § 4, we consider the *double lexical ordering problem*: given an $n \times k$ nonnegative matrix M containing m nonzero entries, independently permute the rows and columns of M so that both the rows and the columns are in nonincreasing lexicographic order. Lubiw [11] defined this problem and gave an $O(m(\log (n+k))^2+n+k)$ time and $O(m+n+k)$ space algorithm. We develop an implementation of her algorithm running in $O(m \log (n+k)+n+k)$ time and $O(m+n+k)$ space. Our method uses a new, efficient algorithm for unmerging two sorted sets.

We conclude in § 5 with some miscellaneous remarks. Although our algorithms use different refinement strategies tuned for efficiency, the similarities among them are compelling and suggest further investigation of the underlying principles.

2. Lexicographic sorting. Let Σ be an alphabet of $k \geq 1$ symbols and let Σ^* be the set of finite length strings over Σ. The empty string is denoted by λ. For $x \in \Sigma^*$ we denote the ith symbol of x by $x(i)$ and the length of x (number of symbols) by $|x|$. Symbol $x(i)$ is said to be in *position* i of string x. For any $x, y \in \Sigma^*$, the *concatenation* of x and y, denoted by xy, is a new string z of length $|x|+|y|$, in which $z(i)=x(i)$ for $1 \leq i \leq |x|$ and $z(|x|+i)=y(i)$ for $1 \leq i \leq |y|$. For any $x, y \in \Sigma^*$, x is a *prefix* of y iff $y=xz$ for some string $z \in \Sigma^*$. Note that every string is a prefix of itself. If x is a prefix of y but $x \neq y$, then x is a *proper prefix* of y. Let $<$ be a total ordering of Σ. We extend $<$ to a total ordering of Σ^*, called *lexicographic order*, by defining $x \leq y$ iff either (i) there exists j such that $x(j)<y(j)$ and $x(i)=y(i)$ for $1 \leq i<j$, or (ii) x is a prefix of y.

Let $\Sigma=\{1, 2, \cdots, k\}$, let $<$ be numeric order on $\Sigma \cup \{0\}$, and let Σ^*0 denote the set of strings formed by appending 0 to the end of each string in Σ^*. Our formulation of the *lexicographic sorting problem* is that of arranging a multiset of strings $U \subseteq \Sigma^*0$ in increasing lexicographic order. Note that a lexicographic arrangement of strings is the same whether all or none have 0 end markers.

In considering this problem we shall denote the strings in U by $x_1, x_2, \cdots, x_n$ and their total length $\sum_{i=1}^{n} |x_i|$ by m. For each string $x_i \in U$, the *distinguishing prefix* of x_i in U, denoted by x_i', is either

(i) the shortest prefix of x_i that is not a prefix of any other string in U if x_i occurs uniquely in U; or

(ii) x_i if x_i has multiple occurrences in U.

Because the end marker 0 can only occur in the last position of a string, no string in U can be a proper prefix of any other string in U. Thus, if $x, y \in U$ and $x \neq y$, we know that $x' \neq y'$. Consequently, if $x, y \in U$, then $x \leq y$ iff $x' \leq y'$, so that sorting the distinguishing prefixes is both necessary and sufficient to sort U. If we denote the total length $\sum_{i=1}^{n} |x_i'|$ of the distinguishing prefixes by m', then lexicographic sorting requires $\Omega(m')$ time. Note that $|x_i'| \geq 1$ by the definition of x_i', which implies $m' \geq n$.

Aho, Hopcroft and Ullman [1] presented a lexicographic sorting algorithm running in $O(m+k)$ time and space. Their algorithm uses a multipass radix sort, processing the strings from last position to first and using appropriate preprocessing to determine the set of buckets occupied during each pass of the radix sort. Mehlhorn [13] studied the special case of lexicographic sorting in which all strings have the same length. He concluded that a straightforward algorithm scanning strings from first position to last must use $\Omega(km)$ time (see [13, exercise 18, p. 100]).

We exhibit a lexicographic sorting algorithm that scans strings in the direction from first to last position and runs in $O(m'+k)$ time, thereby improving the Aho, Hopcroft and Ullman algorithm. The auxiliary space required by our algorithm (not counting space for the input strings) is $O(n+k)$, also an improvement. Our algorithm

requires fewer passes than the Aho, Hopcroft and Ullman algorithm and may be superior in practice as well as in theory. For an appropriately defined probability model, m' is on the order of $n \log_k n$; thus in situations where m is much larger than $n \log_k n$, an improvement from $O(m+k)$ to $O(m'+k)$ is significant.

The first and most important step in obtaining an efficient sorting algorithm is to separate the concern of finding the distinguishing prefixes from that of sorting the prefixes. Thus our algorithm consists of two steps:

(1) Determine the distinguishing prefixes of all the strings.

(2) Lexicographically sort these prefixes.

To describe the first step, we use the following definitions. We partition U into labeled blocks B_α. Block B_α is the multiset of all strings in U with prefix α; e.g., $B_\lambda = U$. We call α the *associated prefix* of B_α. If α and β are strings and α is a prefix of β, then $B_\alpha \supseteq B_\beta$. Block B_α is *finished* iff $\alpha = x'$ for some string $x \in U$. Thus we can determine the distinguishing prefixes for the strings in U by finding all of the finished blocks. A block that is not finished is said to be *unfinished.*

Let P be a set of labeled blocks that partition U. We say that P is *finished* if it consists of all the finished blocks; otherwise, it is *unfinished.* If P contains an unfinished block B_α, then $split(B_\alpha, P)$ denotes the refinement of P that results from replacing block B_α by the blocks in the set $\{B_{\alpha u} \mid u \in \Sigma \cup \{0\}$ *such that* $\exists x \in B_\alpha$ *for which* $x(|\alpha|+1) = u\}$.

To determine all the finished blocks, we start with an initial partition $P = \{B_\lambda\}$ and successively refine P by executing the following step until P is finished:

Refine. Find an unfinished block $B_\alpha \in P$; replace P by $split(B_\alpha, P)$.

LEMMA 1. *The refinement algorithm maintains the invariant that the partition F of finished blocks is a refinement of P, and for all $B_\beta \in F$ there exists $B_\alpha \in P$ such that α is a prefix of β.*

Proof. The proof is by induction on the number of refinement steps. The lemma is true for B_λ. Assume it is true after $k \geqq 0$ steps. Let B_β be a finished block. By hypothesis, just after the kth step there is a block $B_\alpha \in P$ such that α is a prefix of β, and so $B_\beta \subseteq B_\alpha$. If B_α is finished, so that $\alpha = \beta$, then the invariant holds after the next step. If B_α is unfinished, then α is a proper prefix of β. Hence, $split(B_\alpha, P)$ is defined and contains a block $B_{\alpha u}$ such that αu is a prefix of β, which implies $B_\beta \subseteq B_{\alpha u}$. □

THEOREM 1. *The algorithm terminates and is correct.*

Proof. Lemma 1 implies that the algorithm terminates and is correct. Also by Lemma 1, the sum of the lengths of all the associated prefixes of blocks in P can be no greater than m'. Since each refinement step increases this sum, the algorithm terminates after at most m' refinement steps. □

We make several observations that facilitate an efficient implementation of the preceding algorithm. By Lemma 1 and the definition of distinguishing prefixes, the algorithm maintains the invariant that a block $B_\alpha \in P$ is unfinished iff $|B_\alpha| \neq 1$ and $\alpha(|\alpha|) \neq 0$. Thus we can test whether a block is finished or not in constant time, and we can also maintain the set S of unfinished blocks in P efficiently. Although we could implement each block B_α by an index $I = |\alpha|$ and a list of pointers to strings, by implementing S as a queue and using a breadth-first strategy to refine the unfinished blocks of S, we can make I a global variable.

That is, we use a repeated partition refinement strategy that scans the strings starting from the first position and partitions the strings into blocks on the basis of their scanned prefixes. The algorithm maintains three objects: a partition P of the strings, a set S containing the unfinished blocks of P, and an integer I indicating the

current position being examined in the strings. Initially, $P=\{U\}$, $S=\{U\}$, and $I=0$. The algorithm consists of repeating the following step until S is empty:

Scan Blocks. Replace I by $I+1$. Let $S'=S$ and replace S by ϕ. Repeat the following step for each block B_α in S'.

Refine. Remove B_α from S'. Replace P by $split(B_\alpha, P)$. Add each new unfinished block to S. The distinguishing prefix of any string belonging to each new finished block $B_{\alpha u}$ is αu, which is of length I.

The preceding algorithm maintains the invariant $|\alpha|=I-1$ on entry to the *refine* step. The effect of one iteration of the step *scan blocks* is to refine the partition P based on the Ith symbol in each string. The correctness of the algorithm is obvious. We can implement the algorithm to run in $O(m')$ time and $O(n+k)$ auxiliary space as follows. Let B be a block selected for refinement. To partition B, we use an auxiliary array of buckets numbered 0 through k. We insert the strings in B into the buckets corresponding to their Ith symbols, simultaneously making a list of all nonempty buckets. (When a string is added to an empty bucket, we add that bucket to the list of nonempty buckets.) Once the strings are distributed among the buckets, we examine each nonempty bucket, forming a new block from the contents and emptying the bucket.

When moving strings from place to place, we do not move the strings themselves but rather pointers to them; thus each string movement takes $O(1)$ time. We can avoid initializing each bucket in the array of buckets to empty by using the solution to exercise 2.12 of Aho, Hopcroft and Ullman's book [1]. The total time for the partitioning algorithm is then $O(m')$, since each symbol of each distinguishing prefix is scanned once. If we explicitly initialize the array of buckets before the first refinement step, the time for partitioning, including initialization, is $O(m'+k)$. The auxiliary space needed by the partitioning algorithm is $O(1)$ words per string plus $k+1$ words for the array of buckets, for a total of $O(n+k)$.

To carry out step two, the sorting of the prefixes, one option is to use the lexicographic sorting algorithm of Aho, Hopcroft and Ullman. This method, however, uses $O(m'+k)$ auxiliary space. If we compute some extra information during step one, we can reduce the space needed and considerably simplify the sorting task.

We distinguish between two kinds of refinements. Let us call a refinement a *proper refinement* if the refined block B is actually split into two or more smaller blocks and an *improper refinement* if B is not split but is returned to P intact. We modify our basic algorithm for step one in order to construct a *refinement tree* T whose nodes correspond to the distinct blocks produced by the algorithm. Initially T contains only one node B_λ. As T is constructed, its leaves are the blocks of P. After a proper refinement in which a block B_α is split, we make B_α the parent of each new leaf block $B_{\alpha u}$ and say that u is the *sorting symbol* for $B_{\alpha u}$. An improper refinement to a leaf block B does not change T, since only the associated prefix and not the value of B is modified. The sorting symbol, if any, remains unchanged. Note that only the root block lacks a sorting symbol.

The procedure for constructing a refinement tree T preserves the following two invariants: (i) siblings (i.e., nodes with the same parent) in the tree have distinct sorting symbols; and (ii) if B_α and B_β are siblings with sorting symbols u and v, respectively, and $u<v$, then all strings $x\in B_\alpha$ are lexicographically less than all strings $y\in B_\beta$. Thus, when T is fully constructed, if we arrange each set of siblings within T in ascending order by sorting symbol, then the blocks forming the frontier of T contain the strings of U in lexicographic order.

We represent each block created during step one by a record. For each block B we store with the block its parent $p(B)$ in the refinement tree. If block B belongs to

P (i.e., is either finished or is unfinished and belongs to S), we also store with the block a list of pointers to the strings it contains. For each symbol $i = 0, \cdots, k$, we construct a set L_i of blocks in the refinement tree that have sorting symbol i. When a new block $B_{\alpha u}$ is created during a proper refinement of block B_α, we add $B_{\alpha u}$ to L_u and remove from B_α the list of pointers to the strings it contains. By implementing $L_0, \cdots, L_k$ as an array of lists of blocks, these extra computations increase the time and auxiliary space used in step one by only a constant factor.

In step two, the process of ordering the tree consists of constructing for each block an ordered list of its children. We begin by initializing these lists of children to be empty. Then, for $i = 0, \cdots, k$ and for each block $B \in L_i$, we remove B from L_i and add it to the end of the list of children of block $p(B)$. Thus, when this computation is completed, the children of each block will be in increasing order by sorting symbol.

Since each node of the refinement tree T has either no children or at least two children, T contains at most $2n-1$ nodes. Ordering the tree takes $O(n+k)$ time and auxiliary space. The preorder tree traversal that actually produces the sorted list of strings takes $O(n)$ time. The total resources needed for lexicographic sorting are thus $O(m'+k)$ time and $O(n+k)$ auxiliary space. The space consists of $O(1)$ words per string and two auxiliary arrays of $k+1$ words each. The two auxiliary arrays of buckets can be combined into one array by storing strings belonging to a leaf block of the refinement tree at the front of each bucket and storing blocks internal to the refinement tree at the back. This saves $k+1$ words of space.

We conclude this section with two observations. First, the algorithm can be greatly simplified if the alphabet is of constant size, i.e., $k = O(1)$. In this case we can combine step two with step one, because we can afford to scan empty buckets of the array when forming new blocks during a refinement step. When partitioning a block B during step one, after distributing the strings of B into buckets, we scan all the buckets in increasing order, forming a new block for each nonempty bucket. This allows us to modify the basic algorithm for step one so that both the partition P and the queue of unfinished blocks S are kept in sorted order by the associated prefixes of their blocks. Each block has a pointer to its successor in both S and P. After P is finished, we can output all the strings in lexicographic order by a single linear pass through its blocks.

Alternatively, we can perform a single left-to-right scan of the ordered blocks of P as they are refined. In this method, we need to store the position index locally in each block. Initially, P consists of the single block B_λ with index $I = 0$, which is the first block processed. To process a block B_α we increment its index and test whether it is finished. If it is, then we process the next block to the right or stop if P is finished. If B_α is unfinished, we perform a refinement and replace B_α with the new blocks in sorted order. The index of each new block is I. The leftmost of the new blocks added to P is processed next. This algorithm has the advantage that the strings in a block B can be printed as soon as we determine that B is finished.

Our second observation is that if we can process several symbols of a string at a time then we can obtain a time-space trade-off for the algorithm: if we process c symbols at a time, the running time is $O(m'/c + k^c)$, and the space used is $O(n + k^c)$. In the case of a binary alphabet, this means that we can sort in $O(m'/\log n)$ time and $O(n)$ auxiliary space if we are allowed to operate on $\log n$ bits at a time. An interesting open problem is whether the ideas of Kirkpatrick and Reisch [10] can somehow be applied to the variable-length lexicographic sorting problem we have considered here.

3. Relational coarsest partition. Let E be a binary relation over a finite set U, i.e., a subset of $U \times U$. We abbreviate $(x, y) \in E$ by xEy. For any subset $S \subseteq U$, $E(S) = \{y \mid \exists x \in S \text{ such that } xEy\}$ and $E^{-1}(S) = \{x \mid \exists y \in S \text{ such that } xEy\}$. If $B \subseteq U$ and $S \subseteq U$,

B is *stable* with respect to S if either $B \subseteq E^{-1}(S)$ or $B \cap E^{-1}(S) = \emptyset$. If P is a partition of U, P is *stable* with respect to S if all of the blocks belonging to P are stable with respect to S. P is *stable* if it is stable with respect to each of its own blocks.

The *relational coarsest partition problem* is that of finding, for a given relation E and initial partition P over a set U, the coarsest stable refinement of P, i.e., the partition such that every other stable partition is a refinement of it, so that it has the fewest blocks. (In Theorem 2 we prove that the coarsest stable refinement is unique.) In discussing time bounds for this problem, we let n denote the size of U and m the size of E.

This problem was studied by Kanellakis and Smolka [9] in connection with testing congruence (a kind of equivalence) of finite state processes in the calculus of communicating systems (CCS) [14]. They gave an algorithm requiring $O(mn)$ time and $O(m+n)$ space. For the special case in which every element $x \in U$ has an image set $E(\{x\})$ of size at most a constant c, they gave an algorithm running in $O(c^2 n \log n)$ time. They conjectured the existence of an $O(m \log n)$ time algorithm for the general problem.

We develop an $O(m \log n)$ time algorithm, thereby verifying their conjecture. Our algorithm combines Hopcroft's "process the smaller half" strategy [1], [7] with a new variant of partition refinement. Hopcroft used the "process the smaller half" idea to solve a deceptively similar problem, that of computing the coarsest partition stable with respect to one or more given functions. A solution to this problem can be used to minimize the number of states of a deterministic finite automaton [1], [5], [7]. As Kanellakis and Smolka noted, the relational coarsest partition problem is sufficiently different from the functional problem that a nontrivial generalization of Hopcroft's algorithm is needed to solve it.

Our algorithm uses a primitive refinement operation that generalizes the one used in Hopcroft's algorithm. For any partition Q and subset $S \subseteq U$, let $split(S, Q)$ be the refinement of Q obtained by replacing each block $B \in Q$ such that $B \cap E^{-1}(S) \neq \emptyset$ and $B - E^{-1}(S) \neq \emptyset$ by the two blocks $B' = B \cap E^{-1}(S)$ and $B'' = B - E^{-1}(S)$. We call S a *splitter* of Q if $split(S, Q) \neq Q$. Note that Q is unstable with respect to S if and only if S is a splitter of Q.

Our algorithm exploits several properties of *split* and consequences of stability. Let S and Q be two subsets of U, and let P and R be two partitions of U. The following elementary properties are stated without proof.

(1) Stability is *inherited* under refinement; that is, if R is a refinement of P and P is stable with respect to a set S, then so is R.

(2) Stability is *inherited* under union; that is, a partition that is stable with respect to two sets is also stable with respect to their union.

(3) Function *split* is *monotone* in its second argument; that is, if $S \subseteq U$ and P is a refinement of Q, then $split(S, P)$ is a refinement of $split(S, Q)$.

(4) Function *split* is *commutative*. The coarsest partition of P stable with respect to both S and Q is $split(S, split(Q, P)) = split(Q, split(S, P))$.

We begin by describing a naïve algorithm for the problem. The algorithm maintains a partition Q that is initially P and is refined until it is the coarsest stable refinement. The algorithm consists of repeating the following step until Q is stable with respect to each of its blocks:

Refine. Find a set S that is a union of some of the blocks of Q and is a splitter of Q; replace Q by $split(S, Q)$.

Before proving the correctness of this algorithm, let us make a few observations. The effect of a refinement step is to replace a partition unstable with respect to a set S by a refinement stable with respect to S. Since stability is inherited under refinement, a given set S can be used as a splitter in the algorithm only once. Since stability is inherited under the union of splitters, after sets are used as splitters their unions cannot be used as splitters. In particular, a stable partition is stable with respect to the union of any subset of its blocks.

LEMMA 2. *The algorithm maintains the invariant that any coarsest stable refinement of the initial partition P is also a refinement of the current partition Q.*

Proof. By induction on the number of refinement steps. The lemma is true initially by definition. Suppose it is true before a refinement step that refines partition Q using a splitter S. Let R be any coarsest stable refinement of P. Since S is a union of blocks of Q and R is a refinement of Q by the induction hypothesis, S is a union of blocks of R. Hence, R is stable with respect to S. Since *split* is monotone, $R = split(S, R)$ is a refinement of $split(S, Q)$. □

THEOREM 2. *The refinement algorithm is correct and terminates after at most $n-1$ refinement steps, having computed the unique coarsest stable partition.*

Proof. Since the number of blocks in Q must be between one and n, and since this number increases after each refinement step, the algorithm terminates after at most $n-1$ refinement steps. Once no more refinement steps are possible, Q is stable, and by Lemma 2 any stable refinement is a refinement of Q. It follows that Q is the unique coarsest stable refinement. □

The refinement algorithm is more general than is necessary to solve the problem: there is no need to use unions of blocks of the current partition Q as splitters; restricting the splitters to blocks of Q will do. However, the freedom to split using unions of blocks is one of the crucial ideas needed in developing a fast version of algorithm. We shall see how this freedom is used later in the section.

In an efficient implementation of the algorithm, it is useful to reduce the problem instance to one in which $|E(\{x\})| \geq 1$ for all $x \in U$. To do this we preprocess the partition P by splitting each block $B \in P$ into $B' = B \cap E^{-1}(U)$ and $B'' = B - E^{-1}(U)$. The blocks B'' will never be split by the refinement algorithm; thus we can run the refinement algorithm on the partition P' consisting of the set of blocks B'. P' is a partition of the set $U' = E^{-1}(U)$, of size at most m. The coarsest stable refinement of P' together with the blocks B'' is the coarsest stable refinement of P. The preprocessing and postprocessing take $O(m+n)$ time if we have available for each element $x \in U$ its preimage set $E^{-1}(\{x\})$. Henceforth we shall assume $|E(\{x\})| \geq 1$ for all $x \in U$. This implies $m \geq n$.

We can implement the refinement algorithm to run in $O(nm)$ time by storing for each element $x \in U$ its preimage set $E^{-1}(\{x\})$. Finding a block of Q that is a splitter and performing the appropriate splitting takes $O(m)$ time. (Obtaining this bound is an easy exercise in list processing.) An $O(mn)$ time bound for the entire algorithm follows from the bound on the number of splitting steps.

To obtain a faster version of the algorithm, we need a good way to find splitters. In addition to the current partition Q, we maintain another partition X such that Q is a refinement of X and Q is stable with respect to every block of X. Initially $Q = P$ and X is the partition containing U as its single block. The improved algorithm consists of repeating the following step until $Q = X$:

> *Refine.* Find a block $S \in X$ that is not a block of Q. Find a block $B \in Q$ such that $B \subseteq S$ and $|B| \leq |S|/2$. Replace S within X by the two sets B and $S - B$; replace Q by $split(S - B, split(B, Q))$.

The correctness of this improved algorithm follows from the correctness of the original algorithm and from the two ways given previously in which a partition can inherit stability with respect to a set.

Before discussing this algorithm in general, let us consider the special case in which E is a function, i.e., $|E(\{x\})| = 1$ for all $x \in U$. In this case, if Q is a partition stable with respect to a set S that is a union of some of the blocks of Q, and $B \subseteq S$ is a block of Q, then *split*(B, Q) is stable with respect to $S - B$, since if B_1 is a block of *split*(B, Q), $B_1 \subseteq E^{-1}(B)$ implies $B_1 \cap E^{-1}(S-B) = \emptyset$, and $B_1 \subseteq E^{-1}(S) - E^{-1}(B)$ implies $B_1 \subseteq E^{-1}(S-B)$. This means that in each refinement step it suffices to replace Q by *split*(B, Q), since *split*(B, Q) = *split*$(S-B,$ *split*$(B, Q))$. This is the idea underlying Hopcroft's "process the smaller half" algorithm for the functional coarsest partition problem; the refining set B is at most half the size of the stable set S containing it.

In the more general relational coarsest partition problem, stability with respect to both S and B does *not* imply stability with respect to $S - B$, and Hopcroft's algorithm cannot be used. Nevertheless, we are still able to exploit Hopcroft's "process the smaller half" idea, by refining with respect to both B and $S - B$ using a method that explicitly scans only B.

Consider a general step of the improved refinement algorithm.

LEMMA 3. *Suppose that partition Q is stable with respect to a set S that is a union of some of the blocks of Q. Suppose also that partition Q is refined first with respect to a block $B \subseteq S$ and then with respect to $S - B$. Then the following conditions hold*:

(1) *Refining Q with respect to B splits a block $D \in Q$ into two blocks $D_1 = D \cap E^{-1}(B)$ and $D_2 = D - D_1$ iff $D \cap E^{-1}(B) \neq \emptyset$ and $D - E^{-1}(B) \neq \emptyset$.*

(2) *Refining* split(B, Q) *with respect to $S - B$ splits D_1 into two blocks $D_{11} = D_1 \cap E^{-1}(S-B)$ and $D_{12} = D_1 - D_{11}$ iff $D_1 \cap E^{-1}(S-B) \neq \emptyset$ and $D_1 - E^{-1}(S-B) \neq \emptyset$.*

(3) *Refining* split(B, Q) *with respect to $S - B$ does not split D_2.*

(4) $D_{12} = D_1 \cap (E^{-1}(B) - E^{-1}(S-B))$.

Proof. (1, 2) These follow from the definition of *split*.

(3) By (1), if D is split, then $D \cap E^{-1}(B) \neq \emptyset$. Since D is stable with respect to S, and since $B \subseteq S$, then $D_2 \subseteq D \subseteq E^{-1}(S)$. Since by (1) $D_2 \cap E^{-1}(B) = \emptyset$, it follows that $D_2 \subseteq E^{-1}(S-B)$.

(4) This follows from the fact that $D_1 \subseteq E^{-1}(B)$ and $D_{12} = D_1 - E^{-1}(S-B)$. □

Performing the three-way splitting of a block D into D_{11}, D_{12}, and D_2 as described in Lemma 3 is the hard part of the algorithm. Identity (4) of Lemma 3 is the crucial observation that we shall use in our implementation.

A given element $x \in U$ is in at most $\log_2 n + 1$ different blocks B used as refining sets, since each successive such set is at most half the size of the previous one. We shall describe an implementation of the algorithm in which a refinement step with respect to a block B takes $O(|B| + \Sigma_{y \in B} |E^{-1}(\{y\})|)$ time. From this an $O(m \log n)$ overall time bound on the algorithm follows by summing over all blocks B used for refinement and over all elements in such blocks.

Efficient implementation of the algorithm requires several data structures. We represent each element x by a record, which we shall not distinguish from the element itself. We represent each pair x, y such that xEy by a record that we shall call an *edge* and denote by xEy. We represent each block in partition Q and each block in partition X by a record that we shall not distinguish from the block itself. A block S of X is *simple* if it contains only a single block of Q (equal to S but indicated by its own record) and *compound* if it contains two or more blocks of Q. The various records are linked together in the following ways. Each edge xEy points to element x. Each element y points to a list of the edges xEy. This allows scanning of the set $E^{-1}(\{y\})$ in time

proportional to its size. Each block of Q has an associated integer giving its size and points to a doubly linked list of the elements in it. (The double linking allows deletion in $O(1)$ time.) Each element points to the block of Q containing it. Each block of X points to a doubly linked list of the blocks of Q contained in it. Each block of Q points to the block of X containing it. We also maintain the set C of compound blocks of X. Initially C contains the single block U, which is the union of the blocks of P. (If P contains only one block, P itself is the coarsest stable refinement, and no computation is necessary.)

To facilitate three-way splitting we need one more collection of records. For each block S of X and each element $x \in E^{-1}(S)$, we maintain a record containing the integer $count(x, S) = |S \cap E(\{x\})|$. We shall not distinguish this record from the count itself. Each edge xEy such that $y \in S$ contains a pointer to $count(x, S)$. Initially there is one count per vertex, $count(x, U) = |E(\{x\})|$; each edge xEy contains a pointer to $count(x, U)$.

The space needed for all the data structures is $O(m)$, as is the initialization time. (Recall that $m \geqq n$.) The refinement algorithm consists of repeating refinement steps until $C = \varnothing$. A refinement step is performed as follows:

Step 1 (select a refining block). Remove some block S from C. (Block S is a compound block of X.) Examine the first two blocks in the list of blocks of Q contained in S. Let B be the smaller. (Break a tie arbitrarily.)

Step 2 (update X). Remove B from S and create a new (simple) block S' of X containing B as its only block of Q. If S is still compound, put S back into C.

Step 3 (compute $E^{-1}(B)$). Copy the elements of B into a temporary set B'. (This facilitates splitting B with respect to itself during the refinement.) Compute $E^{-1}(B)$ by scanning the edges xEy such that $y \in B$ and adding each element x in such an edge to $E^{-1}(B)$, if it has not already been added. Duplicates are suppressed by marking elements as they are encountered and linking them together for later unmarking. During the same scan, compute $count(x, B) = |\{y \in B \mid xEy\}|$, store this count in a new *count* record, and make x point to it.

Step 4 (refine Q with respect to B). For each block D of Q containing some element of $E^{-1}(B)$, split D into $D_1 = D \cap E^{-1}(B)$ and $D_2 = D - D_1$. Do this by scanning the elements of $E^{-1}(B)$. To process an element $x \in E^{-1}(B)$, determine the block D of Q containing it, and create an associated block D' if one does not already exist. Move x from D to D'.

During the scanning, construct a list of those blocks D that are split. After the scanning, process the list of split blocks. For each such block D with associated block D', mark D' as no longer being associated with D (so that it will be correctly processed in subsequent iterations of Step 4); eliminate the record for D if D is now empty; and, if D is nonempty and the block of X containing D and D' has been made compound by the split, add this block to C.

Step 5 (compute $E^{-1}(B) - E^{-1}(S - B)$). Scan the edges xEy such that $y \in B'$. To process an edge xEy, determine $count(x, B)$ (to which x points) and $count(x, S)$ (to which xEy points). If $count(x, B) = count(x, S)$, add x to $E^{-1}(B) - E^{-1}(S - B)$ if it has not been added already.

Step 6 (refine Q with respect to $S - B$). Proceed exactly as in Step 4 but scan $E^{-1}(B) - E^{-1}(S - B)$ (computed in Step 5) instead of $E^{-1}(B)$.

Step 7 (update counts). Scan the edges xEy such that $y \in B'$. To process an edge xEy, decrement $count(x, S)$ (to which xEy points). If this count becomes zero, delete the *count* record, and make xEy point to $count(x, B)$ (to which x points). After scanning all the appropriate edges, discard B'.

The correctness of this implementation follows in a straightforward way from our discussion above of three-way splitting. The time spent in a refinement step is $O(1)$ per edge scanned plus $O(1)$ per vertex of B, for a total of $O(|B|+\Sigma_{y\in B}|E^{-1}(\{y\})|)$ time. An $O(m \log n)$ time bound for the entire algorithm follows as discussed above. It is possible to improve the efficiency of the algorithm by a constant factor by combining various steps; we have kept the steps separate for clarity.

We close this section with some remarks about another partition refinement problem. Let U be a finite set, E a binary relation over U, and P a partition over U. Suppose $S \subseteq U$. Partition P is *size-stable* (abbreviated *s-stable*) with respect to S if for all blocks $B \in P$, $|E(\{x\}) \cap S| = |E(\{y\}) \cap S|$ for all elements $x, y \in B$. Partition P is *s-stable* if it is s-stable with respect to each of its blocks. The problem is to find the coarsest s-stable refinement of P. This problem is central to certain heuristics for graph isomorphism [16]. Observe that s-stability implies stability but not vice versa. An example of a stable partition that is not s-stable is a single block $B = \{1, 2\}$ such that $E(\{1\}) = \{1, 2\}$ and $E(\{2\}) = \{1\}$.

A partition refinement algorithm like the one we have described can be used to find the coarsest s-stable refinement in $O(m \log n)$ time. The problem is both simpler and more complicated than the stable coarsest refinement problem: simpler in that Hopcroft's approach applies directly, but more complicated in that when a block splits, it splits into an arbitrarily large number of new blocks. We shall sketch the algorithm. An algorithm with the same running time was proposed earlier by Cardon and Crochemore [3].

For a partition Q and a set S, let *s-split*(S, Q) be the partition formed from Q by splitting each block $B \in Q$ into blocks $B_0, B_1, \cdots, B_k$, where $B_i = \{x \in B: |E(\{x\}) \cap S| = i\}$. (Any of the B_i's can be empty; for a proper split to take place, at least two must be nonempty.) The algorithm maintains a queue L of possible splitters, initially containing every block of P. The algorithm consists of initializing $Q = P$ and applying the following step until L is empty, at which time Q is the coarsest s-stable refinement:

> *S-Refine.* Remove from L its first set S. Replace Q by *s-split*(S, Q). Whenever a block $B \in Q$ splits into two or more nonempty blocks, add all but the largest to the back of L.

The correctness of this algorithm follows from the fact that if Q is s-stable with respect to a set S and S is split into smaller blocks $S_1, S_2, \cdots, S_k$, refining with respect to all but one of the sets S_i guarantees s-stability with respect to the last one. The implementation of the algorithm is analogous to that of the stable coarsest refinement algorithm. The data structures are much simplified because when refining with respect to a block $B \subseteq S$ we do not need to refine with respect to $S - B$. This removes the need for the auxiliary partition X and the count records. On the other hand, the splitting of a block becomes more complicated, because it can split into many pieces. Implementing the *s-refine* step to take $O(|S|+\Sigma_{y\in S}|E^{-1}(\{y\})|)$ time, where S is the set removed from L, is an interesting exercise in list processing that we leave to the reader. An $O(m \log n)$ time bound for the entire algorithm follows from the observation that a given element y can be in only $\log_2 n + 1$ sets removed from L.

Although our s-refinement algorithm resembles Cardon and Crochemore's algorithm, Hopcroft's algorithm for the functional coarsest partition problem, and Kanellakis and Smolka's algorithm for the bounded fanout case of the relational coarsest partition problem, our method is somewhat simpler because, by refining with respect to old blocks rather than current ones, we are able to avoid some updating.

(The other algorithms must update L each time a block in L is split.) Our approach thus gives a slightly simplified algorithm for the functional problem. It is also easily generalized to handle several relations, where we require stability or s-stability with respect to each one. See, e.g., Cardon and Crochemore's paper.

4. Double lexical ordering. The third and last problem we consider is one of matrix reordering. Let M be an n by k nonnegative matrix containing m nonzeros, called its *entries.* The *double lexical ordering problem* is that of independently permuting the rows and columns of M so that both the row vectors and the column vectors are in nonincreasing lexicographic order. Lexicographic order is defined as in § 2, reading each row from left to right and each column from top to bottom.

Lubiw [11] defined the double lexical ordering problem and found a number of applications, including the efficient recognition of totally balanced matrices and strongly chordal graphs. In her version of the problem, rows are read from right to left and columns from bottom to top (i.e., vectors are read from last to first component), and the required rows and columns are in nondecreasing lexicographic order. To convert our results to her setting, it suffices to exchange "left" with "right," "top" with "bottom," and "increasing" with "decreasing" throughout this section.

Lubiw developed a double lexical ordering algorithm taking $O(m \log (n+k)^2 + n+k)$ time and $O(m+n+k)$ space. We develop an implementation of her algorithm running in $O(m \log (n+k)+k)$ time and $O(m+n+k)$ space. Since all Lubiw's applications are for the special case of zero-one matrices, we give a simplified algorithm with the same resource bounds for this case.

As input for the ordering problem, we assume a matrix representation consisting of a list of triples, each composed of an entry, its row, and its column. From this input we can construct lists of the entries in each row using a radix sort by row. We can construct similar lists for the columns. This takes $O(m+n)$ time. We assume that $m \geqq n \geqq k$. We can validate the assumption $n \geqq k$ by transposing the matrix if necessary, which takes $O(m)$ time. We can validate the assumption $m \geqq n$ by finding all the rows with no entries and ordering them last, which takes $O(m+n)$ time. Ordering the submatrix consisting of the remaining rows and all the columns suffices to order the entire matrix.

We begin by describing a variant of Lubiw's ordering algorithm. We need some terminology. An *ordered partition* of a finite set S is a partition of S whose blocks are totally ordered. We denote an ordered partition by a list of the blocks in order. A total ordering $<$ of S is *consistent* with an ordered partition P of S if whenever two blocks $S_1 \in P$ and $S_2 \in P$ satisfy $S_1 < S_2$, then every $e_1 \in S_1$ and $e_2 \in S_2$ satisfy $e_1 < e_2$. If P and Q are ordered partitions of S, Q is a *refinement* of P if every block of Q is contained in a block of P and every total ordering of S consistent with Q is consistent with P.

Let M be a nonnegative matrix with n rows and k columns. $M(i, j)$ denotes the entry in the ith row and jth column. An ordered partition $R_1, R_2, \cdots, R_p$ of the rows of M and an ordered partition $C_1, C_2, \cdots, C_q$ of the columns of M divide M into $p \times q$ *matrix blocks* $M(R_i, C_j)$. Block $M(R_i, C_j)$ is the submatrix defined by the rows in row block R_i and the columns in column block C_j. Let $B = M(R_i, C_j)$ be a matrix block. If $M(r, c) = M(r', c')$ for all $r, r' \in R_i$, $c, c' \in C_j$, then block B is *constant.* We denote by $\max(B)$ the maximum value in matrix block B. A *row slice* of B is the intersection of a row $r \in R$ with B, i.e., the submatrix $M(r, C_j)$. A *splitting row* of B is a row $r \in R_i$ such that the row slice $M(r, C_j)$ is nonconstant and contains at least one entry equal to $\max(B)$. If $B' = M(R_{i'}, C_{j'})$ is another matrix block, then B' is *above* B if $i' < i$ and $j' = j$; B' is *left* of B if $i' = i$ and $j' < j$.

The ordering algorithm maintains an ordered partition of the rows, initially containing one block of all the rows, and an ordered partition of the columns, initially containing one block of all the columns. The algorithm refines the row and column partitions until every matrix block defined by the partitions is constant, at which time any total ordering of the rows and columns consistent with the final partitions is a double lexical ordering. The algorithm consists of repeating the following step until every matrix block is constant:

Refine. Let $B = M(R, C)$ be any nonconstant matrix block such that all blocks left of B and all blocks above B are constant. If B has a splitting row r, replace C in the ordered column partition by the ordered pair $C' = \{c \in C \mid M(r, c) = \max(B)\}$, $C'' = \{c \in C \mid M(r, c) < \max(B)\}$ (so that C' comes before C'' in the column ordered partition). If B has no splitting row, replace R in the ordered row partition by the ordered pair $R' = \{r \in R \mid M(r, c) = \max(B) \forall c \in C\}$, $R'' = \{r \in R \mid \exists c \in C$ such that $M(r, c) < \max(B)\}$ (so that R' comes before R'' in the ordered row partition).

Lubiw's proof of correctness for her algorithm is valid without change for our algorithm. However, our algorithm differs in two ways from Lubiw's algorithm as presented in [11]. First, in each refinement step, her implementation selects a matrix block $B = M(R, C)$ that has C minimum among all nonconstant blocks, and among blocks within C has R minimum. Selected matrix blocks are totally ordered from top to bottom within each column block and from left to right among column blocks. In contrast, our algorithm selects blocks according to the partial order defined by "left" and "above" (i.e., if block B is left of or above block B', then B precedes B'). This additional flexibility in our algorithm contributes to the improved time bound. Second, her algorithm arbitrarily chooses either a splitting row or column (defined analogously to splitting row) when both exist, whereas our algorithm chooses a splitting column only when there are no splitting rows. However, this is not an essential difference, because whenever there is no splitting row (respectively splitting column), the refinement made by her algorithm is the same as ours. For example, if there is no splitting row, her algorithm chooses any splitting column c and replaces the row block R by the ordered pair $R' = \{r \in R \mid M(r, c) = \max(B)\}$, $R'' = \{r \in R \mid M(r, c) < \max(B)\}$. If there is no splitting row, every row slice of $M(R, C)$ containing at least one value equal to $\max(B)$ must have all values equal to $\max(B)$. Thus, in this case the refinement made by her algorithm is independent of the choice of splitting column and is the same as the refinement made by our algorithm.

Both Lubiw's efficient implementation and ours are based on Hopcroft's "process the smaller half" idea, which we have already used in § 3. Since the special case of zero-one matrices is much simpler than the general case and since it is what occurs in all of Lubiw's applications, we discuss it first. In the zero-one case, the refinement step becomes the following:

Refine. Let $B = M(R, C)$ be any nonconstant matrix block such that all blocks left of B and all blocks above B are constant. If there is a row r such that $M(r, C)$ is nonconstant, replace column block C by the ordered pair $C' = \{c \in C \mid M(r, c) = 1\}$, $C'' = \{c \in C \mid M(r, c) = 0\}$. Otherwise, replace row block R by the ordered pair $R' = \{r \in R \mid M(r, c) = 1 \forall c \in C\}$, $R'' = \{r \in R \mid M(r, c) = 0 \forall c \in C\}$. (In the latter case both $M(R', C)$ and $M(R'', C)$ are constant.)

To implement this algorithm, we need to use a rather elaborate data structure, since we must keep track of a row partition, a column partition, and the matrix blocks

they define. We also keep track of row slices. We need only keep track of row slices and matrix blocks containing at least one entry; thus in what follows the terms "row slice" and "matrix block" refer to those with at least one entry. Since we are in the zero-one case, every entry is a one. The data structure contains a record for each entry, row, column, row block, column block, row slice, and matrix block. In our discussion we do not distinguish between an object and the corresponding record. Each entry has pointers to the row, column, and row slice containing it. Each row points to the row block containing it and to a list of its entries. Each column points to the column block containing it and to a list of its entries. Each row block has its size (number of rows) and points to a doubly linked list of its rows. Each column block has its size (number of columns) and points to a doubly linked list of its columns. Each row slice has its size (number of entries) and points to a doubly linked list of its entries and to the matrix block containing it. Each matrix block points to a list of its row slices. In addition, we maintain three lists: one of the row blocks in order, one of the column blocks in order, and one of the matrix blocks, in an order consistent with the partial orders "left" and "above". (If block B is left of or above B', B occurs before B' in the list. See Fig. 1.) Initializing the data structure takes $O(m)$ time.

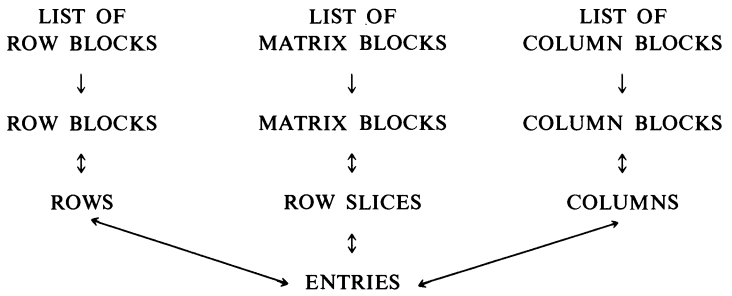


FIG. 1. *Schematic representation of the data structure for the double lexical ordering algorithm.*

To perform a refinement step, we remove the first block $B = M(R, C)$ from the list of matrix blocks. We determine the row block R and column block C (by finding a row slice of B, finding an entry in the row slice, finding the row and column of the entry, and finding the row block of the row and the column block of the column). We scan the row slices of B one at a time until finding one, say s, with fewer than $|C|$ entries. This takes $O(1)$ time per slice. If we find such an s, we scan its entries and determine the set of columns C' containing these entries. We split C into C' and $C'' = C - C'$ and update the data structures accordingly. Then we add $M(R, C')$ and $M(R, C'')$ to the front of the list of matrix blocks (so that $M(R, C')$ is the first block in the list). If there is no such s, we find the set of rows R' containing the row slices of B, split R into R' and $R'' = R - R'$, and update the data structure accordingly.

In updating the data structure we use the "relabel the smaller" half idea. When updating to reflect a split of a column block C into C' and C'', we process whichever of C' and C'' contains fewer columns, spending time proportional to its number of entries. Similarly, when updating to reflect a split of a row block R into R' and R'', we process whichever of R' and R'' contains fewer rows, spending time proportional to its number of entries. Thus, the cost of splitting a row or column can be regarded as the sum of unit time charges to each entry e in the smaller half of the split block. Each two consecutive times that an entry e is charged because of a split in column

(respectively row) blocks, the size of the new block containing e is reduced by at least a factor of two. Thus, any entry e can be charged no more than $O(\log n)$ times, and the total running time of the algorithm is $O(m \log n)$. The space needed is $O(m)$.

Let us discuss what happens when a column block is split. Such a split causes matrix blocks and their row slices to be split also. Finding a row slice s in B with fewer than $|C|$ entries takes $O(1)$ time plus $O(1)$ time for each row slice in B with $|C|$ entries (in the worst case). A row slice in B with $|C|$ entries has at least one entry in both of the new column blocks; thus the cost of finding s is $O(1)$ per entry in the new block with fewer columns. Let s contain $|s|$ entries. Once s is found, we can traverse the list of entries in s to find the columns in the new block C' in $O(|s|)$ time. Once C is split into C' and C'', all the entries in s are in a constant row slice. It follows that a given entry is in only one row slice s used for splitting; hence the total number of columns of new blocks C' (which bounds the total cost of traversing row slices s) formed during the running of the algorithm is $O(m)$. Removing the columns of C' from C takes $O(|C'|)$ time. Suppose that C' has fewer columns than C''. (The other case is similar.) We scan the entries in C', removing them from their current row slices to form new row slices. When the first row slice in a matrix block is split, we form a new matrix block as well, to hold the new row slices split from the block. The time for constructing new row slices and matrix blocks is proportional to the number of entries in C'. When an old row slice or matrix block becomes empty (because all its entries have been deleted), we delete it. We insert a new matrix block into the list of matrix blocks just in front of the block from which it is split. The remaining details of the updating are straightforward.

The updating when a row block is split is similar except that row slices are not split but only moved from old matrix blocks to new ones. The time to split R into R' and R'' is proportional to the number of row slices in B plus the number of entries in whichever of the blocks R' and R'' is processed. It follows from the discussion above that the total running time of the double lexical ordering algorithm is $O(m \log n)$.

We turn now to the general case, in which the matrix entries are arbitrary positive real numbers. Our method is the same as in the zero-one case, with two exceptions. First, we change the data structure slightly, to allow for entries having different values. Second, and more fundamental, we need a new algorithmic tool, to solve a problem in data structures we call the *unmerging problem*: given a list L containing p elements and a set of pointers to q elements in L, partition L into two lists: L', containing the elements indicated by the pointers, and L'', containing the remaining elements. The list order must be preserved, i.e., if element x precedes element y in either L' or L'', then x precedes y in L. This ordering requirement is what makes the problem hard; without it, unmerging takes $O(q)$ time using doubly linked lists, a fact we have used repeatedly in this paper. In the Appendix, we describe how to solve the unmerging problem in $O(q(1+\log(p/q)))$ time by representing the lists as balanced search trees. This efficient solution to the unmerging problem allows us to beat Lubiw's double lexical ordering algorithm by a logarithmic factor even in the general case.

One new concept is needed in order for the zero-one algorithm to handle the general case. A *row slice overlay* is a maximal set of entries in a row slice all having the same value. We change the data structure as follows. Instead of representing row slices, we represent row slice overlays. We represent each overlay by a record, which in our discussion will not be distinguished from the overlay itself. Each overlay has a pointer to the matrix block containing it, a pointer to a doubly linked list of its entries, and an integer giving the number of its entries. Each entry points to the overlay containing it. Each matrix block points to a list of the overlays it contains, in nonincreas-

ing order by the value of their entries. Each list of overlays is represented by a balanced search tree of a kind that supports efficient unmerging and also allows an insertion next to a specified overlay or a deletion to be performed in $O(1)$ amortized time.[1] Red-black trees [6], [17] or weak B-trees [8], [12] will do. Initializing the entire data structure takes $O(m \log n)$ time since we need to sort the entries in each row to determine the overlays and to order the lists of overlays.

We make the following changes in the implementation of a refinement step. Let $B = M(R, C)$ be the first matrix block on the list of matrix blocks. We examine overlays on the list of B's overlays until finding one containing fewer than $|C|$ entries or until the next candidate overlay has entries smaller than $\max(B)$. In the former case, we split C into C', containing the entries in the selected overlay, and $C'' = C - C'$; then we add $M(R, C')$ and $M(R, C'')$ to the front of the list of matrix blocks (with $M(R, C')$ first). In the latter case, we split R into R', containing the entries of B equal to $\max(B)$, and $R'' = R - R'$; then we add $M(R'', C)$ to the front of the list of matrix blocks ($M(R', C)$ is a constant block).

The updating of the data structure proceeds as in the zero-one case except for the updating of the overlay lists. Consider a split of a column block C into C' and C''. Without loss of generality, suppose the entries in C' are processed to perform the updating. Let B be a block containing entries in C'. As entries in C' are examined, they are removed from their old overlays and inserted into new overlays, each of which is inserted in B's list of overlays next to the overlay from which it split. An unmerging operation is then performed to split the overlay list into two lists, one for each of the blocks into which B splits. The updating when a row block splits is similar, except that overlays do not split but are merely moved from one list to another by unmerging operations.

Excluding the unmerging operations, the total time for updating the data structure over the entire algorithm is $O(m \log n)$. We shall derive the same bound for the total time of the unmerging operations. Consider an unmerging operation that splits an overlay list L for a matrix block B into two lists, L' and L'', respectively, for the blocks B' and B'' into which B splits. Let B' be the block containing the entries processed to perform the splitting. Let p be the number of entries in B and q the number in B'. The time for the unmerging operation is $O(|L'|(1+\log((|L'|+|L''|)/|L'|))) = O(q(1+\log(p/q)))$, since $p \geqq |L'|+|L''|$, $q \geqq |L'|$, and $x(1+\log(c/x))$ is an increasing function of x.

We charge $O(q)$ time of the unmerging operation to the processing of the entries in B'. The total of all such charges over the entire algorithm is $O(m \log n)$, since each entry is charged $O(\log n)$ times. It remains to account for $O(q \log(p/q))$ time. We call this the *excess time* associated with the unmerge operation. Let $T(x)$ be the maximum total excess time spent doing unmerging operations associated with a block B containing at most x entries and all smaller blocks formed later contained in B. Then $T(x)$ obeys the recurrence $T(1) = 0$, $T(x) \leqq \max_{y<x}\{T(y) + T(x-y) + O(y \log(x/y))\}$ for $x \geqq 2$. The bound $T(x) = O(x \log x)$ follows easily by induction. Hence the total excess unmerging time over the entire algorithm is $O(m \log m) = O(m \log n)$. We conclude that the algorithm runs in $O(m \log n)$ time total. An $O(m)$ space bound is obvious.

By way of conclusion, we note that our algorithm not only is asymptotically faster than Lubiw's algorithm but it uses less space by a constant factor, since we have

[1] By *amortized time* we mean the time per operation averaged over a worst-case sequence of operations. See Tarjan's survey paper [18] for a full discussion of this concept.

eliminated several unnecessary parts of her data structure: row slices (which she uses in the general case), column slices, column slice overlays, and matrix block overlays. Whether the data structure can be further simplified is an open question. Our algorithm improves the speed of Lubiw's algorithms for recognizing totally balanced matrices and strongly chordal graphs by a logarithmic factor.

5. Remarks. We have presented efficient partition refinement algorithms for three rather different problems. These three algorithms together with our earlier work [15] are part of a larger investigation of why some partition refinement algorithms run in linear time while others do not. We hope that our ideas on partition refinement and our solution to the unmerging problem (see the Appendix) will contribute to the more efficient solution of other problems, such as perhaps the congruence closure problem [4].

Appendix. An efficient unmerging algorithm. Let L be a list containing p elements and let S be a set of pointers to q elements in L. The unmerging problem is to divide L into two lists: L', containing the elements of L indicated by the pointers in S; and L'', containing the remaining elements. The order of elements in L' and L'' must be the same as in L, i.e., if x precedes y in either L' or L'', then x precedes y in L.

To solve the unmerging problem, we represent the input list L by a balanced search tree T. For definiteness, we shall assume that T is a red-black tree [6], [17] or a weak B-tree [8], [12], although many other kinds of balanced trees will do as well. We assume some familiarity with balanced search trees. There are three steps to the algorithm.

Step 1. Color all nodes indicated by pointers yellow and all their ancestors green. To do this, process the nodes indicated by the pointers, one at a time. To process a node x, color x yellow, then the parent of x green, then the grandparent of x green, and so on, until reaching an already-colored node.

Once Step 1 is completed, the colored nodes define a subtree T' of T containing $O(q(1+\log(p/q)))$ nodes [2], [8]. The time required by Step 1 is proportional to the number of nodes in T', i.e., $O(q(1+\log(p/q)))$ time.

Step 2. Perform a symmetric-order traversal of T', uncoloring each colored node as it is reached and adding it to a list if it is yellow.

The list constructed during Step 2 is L'. A red-black tree or weak B-tree representing L' can be constructed in $O(q)$ time (by successive insertions of the items in order [2], [8], [12], [17]).

Step 3. Delete from T the elements indicated by the pointers in S one at a time.

The amortized time per deletion in Step 3 is $O(1)$ [8], [12], [17]. The total worst-case deletion time is $O(q(1+\log(p/q)))$ [2], [8]. Hence the total time for unmerging is $O(q(1+\log(p/q)))$, a bound that is valid both in the worst case and in the amortized case.

Acknowledgment. We thank Anna Lubiw for her perceptive comments concerning the double lexical ordering algorithm.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] M. R. BROWN AND R. E. TARJAN, *Design and analysis of a data structure for representing sorted lists*, this Journal, 9 (1980), pp. 594–614.

[3] A. CARDON AND M. CROCHEMORE, *Partitioning a graph in* $O(|A|\log_2|V|)$, Theoret. Comput. Sci., 19 (1982), pp. 85–98.

[4] P. J. DOWNEY, R. SETHI AND R. E. TARJAN, *Variations on the common subexpression problem*, J. Assoc. Comput. Mach., 27 (1980), pp. 758-771.
[5] D. GRIES, *Describing an algorithm by Hopcroft*, Acta Inform., 2 (1973), pp. 97-109.
[6] L. J. GUIBAS AND R. SEDGEWICK, *A dichromatic framework for balanced trees*, Proc. 19th Annual IEEE Symposium on the Foundations of Computer Science, 1978, pp. 18-21.
[7] J. HOPCROFT, *An n* log *n algorithm for minimizing states in a finite automation*, in Theory of Machines and Computations, Z. Kohavi and A. Paz, eds., Academic Press, New York, 1971, pp. 189-196.
[8] S. HUDDLESTON AND K. MEHLHORN, *A new data structure for representing sorted lists*, Acta Inform., 17 (1982), pp. 157-184.
[9] P. C. KANELLAKIS AND S. A. SMOLKA, *CCS expressions, finite state processes, and three problems of equivalence*, Proc. ACM Symposium on Principles of Distributed Computing, 1983, pp. 228-240.
[10] D. KIRKPATRICK AND S. REISCH, *Upper bounds for sorting integers on random access machines*, Theoret. Comput. Sci., 28 (1984), pp. 263-276.
[11] A. LUBIW, *Doubly lexical orderings of matrices*, Proc. 17th Annual ACM Symposium on the Theory of Computing, 1985, pp. 396-404; this Journal, 16 (1987), pp. 854-879.
[12] D. MAIER AND S. C. SALVETER, *Hysterical B-trees*, Inform. Process. Lett., 12 (1981), pp. 199-202.
[13] K. MEHLHORN, *Data Structures and Algorithms* 1: *Sorting and Searching*, Springer-Verlag, Berlin, 1984.
[14] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer-Verlag, Berlin, 1980.
[15] R. PAIGE, R. E. TARJAN AND R. BONIC, *A linear time solution to the single function coarsest partition problem*, Theoret. Comput. Sci., 40 (1985), pp. 67-84.
[16] R. C. READ AND D. G. CORNEIL, *The graph isomorphism disease*, J. Graph Theory, 1 (1977), pp. 339-363.
[17] R. E. TARJAN, *Data Structures and Network Algorithms*, CBMS-USF Regional Conference Series in Applied Mathematics 44, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.
[18] ———, *Amortized computational complexity*, SIAM J. Algebraic Discrete Methods, 6 (1985), pp. 306-318.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics

OPTIMAL PARALLEL SORTING SCHEME BY ORDER STATISTICS*

MARK C. K. YANG†, JUN S. HUANG‡ AND YUAN-CHIEH CHOW§

Abstract. This paper presents a detailed analysis of a sampling approach used in the partitioning of a data file for the parallel balanced tree sort in a local area network or a multiprocessor environment. The average overall time complexity for sorting N data on a k processor system is derived. The performance of the parallel sorting rests upon how evenly the file can be partitioned into k ordered subfiles. A data partition scheme by sampling is proposed and analyzed. Formulas for computing the optimal sampling size are obtained. The results also show the computational improvement of the sorting as a function of k and the sampling overhead. The performance of the sampling method is studied and found to be approaching the absolute optimal in some cases.

Key words. parallel sorting, algorithms, complexity, samples, order statistics, optimization

AMS(MOS) subject classifications. 62G30, 68E05

1. Introduction. Sorting is an essential operation in data processing as well as in many areas of scientific research. Recent advances in circuit technology and computer architecture have prompted several efforts on developing parallel sorting algorithms and parallel sorting architectures. In general, the design and complexity of parallel sorting algorithms depend heavily on the architecture of the sorting machines. Other factors that have significant impacts on the performance of parallel sorting include file distribution and data access mechanism. Parallel sorting architecture can be classified into two major categories: Sorting that requires special hardware or processors and sorting that can be implemented on general purpose computer networks or multiprocessor systems.

There is a tremendous wealth of proposed parallel sorting schemes in the first category. These sorting schemes are typically expressed as sorting N data with N or more interconnected processors (or comparators and switching elements) with shared common memory. The following examples show some common characteristics of this class of sorting architectures. Batcher [1] proposed a bitonic sorting network which requires $O(N \log^2 N)$ 2×2 switching elements (comparators) and achieved an $O(\log^2 N)$ time complexity. The bitonic sorter was later modified and realized with only $N/2$ comparators interconnected as a perfect shuffle (see Stone [20]). Muller and Preparata [15] showed that by use of N^2 comparators and an enumeration scheme one can accomplish the theoretical lower bound of $O(\log N)$. Hirschberg [11] improved the scheme and obtained the same $O(\log N)$ time complexity with only N processors, an optimal speedup of a factor N. This method, however, necessitates a larger storage requirement to compensate for some minor side effects. In these cases, the sorting networks assume that N data to be sorted are readily available to be accessed simultaneously by the network. That is, data access time is not included in the computation of the time complexity. Several different approaches that embed the sorting time into the data input and output also exist in the literature. These include the tree sorter proposed by Bentley and Kung [2] and the rebound sorter by Chen et al. [4]. Both showed a combined data accessing and sorting time of $O(n)$. Sorting machines in this category require special hardware, a large number of processing

* Received by the editors April 22, 1985; accepted for publication January 7, 1987.
† Department of Statistics, University of Florida, Gainesville, Florida 32611.
‡ Institute of Information Science, Academia Sinica, Taipei 114, Taiwan.
§ Department of Computer and Information Sciences, University of Florida, Gainesville, Florida 32611.

elements and, for a given architecture, increasing the number of processing elements is necessary when the size of the data file increases. A complete taxonomy of these architectures was surveyed by Bitton et al. [3].

In the second category, some of the previous assumptions are relaxed. Parallel sorting is performed in a general purpose local area network or multi-processor system. The number of processors, k, is smaller than the size of the data file, N, and in general $k \ll N$. A commonly adapted approach is to divide the data file into k equal size subfiles and to sort each subfile independently and concurrently for later merging. The overall time complexity consists of accessing, partitioning, sorting, and merging of the data file. The system architecture is independent of the size of the data file. The number of processors involved in the cooperating tasks has direct bearing on the sorting speed. A drawback of this approach is the large communication overhead due to the need of data transfers in the merge phase of the sorting process. Winslow and Chow [22] suggested an alternative to alleviate this problem by using the parallel balanced tree sort and showed that it is optimal in both communication and computation cost. The algorithm consists of three stages: the distribution (or partition) of the data set into ordered subsets, independent parallel sorting in each subset, and the concatenation of the sorted subsets. The concatenation of the sorted and ordered data subsets eliminates the need for merging and thus reduces significantly the amount of data transfer among processors. The key factor in the performance of the sorting depends on how well the data set can be partitioned equally.

The main objective of this paper is to find an optimal strategy for data partitioning in this parallel sorting scheme. The partitioning of the data file is performed by comparing each data against a balanced binary tree constructed by a set of $k-1$ fences (or bounds). This operation forms k buckets of data with order among them. The $k-1$ fences are selected from $nk-1$ random samples of the original data file. The following sections analyze the performance of this partitioning method. The major results of this paper are summarized as follows:

(1) An expression of the overall average time complexity is derived.

(2) The optimal sample size for any given data size is determined.

(3) The rate of computational time reduction as the number of processor increases is analyzed.

(4) The efficiency of this partition method as measured by the best theoretical partition scheme is obtained.

The reason for doing (1) and (2) is quite obvious. Result (3) will help the user to effectively decide on the number of processors for sorting and other tasks. Result (4) checks the optimality of the present method versus any other methods.

2. Definitions, assumptions and criteria. Let the number of processors available for parallel sorting be k and the data to be sorted have size N. Since there are quite a few efficient sorting algorithms such as quicksort, samplesort, shellsort, and balance tree sort, no priority of these algorithms will be assumed. However, we assume that, within a given computer, the same sorting algorithm will be used in all the processors. Because all efficient algorithms take $O(N \log N)$ comparisons to do sorting, we assume that the sorting time for N pieces of data is $cN \log (N)$ where c is a fixed constant independent of N and log is the natural logarithm. Let the $k-1$ fences for data partition be $F_1, F_2, \cdots, F_{k-1}$ and for the notational convenience let $F_0 = -\infty$ and $F_k = \infty$. Then the data between F_{i-1} and F_i are assigned to processor i for sorting. The method of data assignment can be a binary comparison scheme such as the tree shown in Fig. 1. Let the data size in the ith partition be Q_i. Then the time to finish the parallel

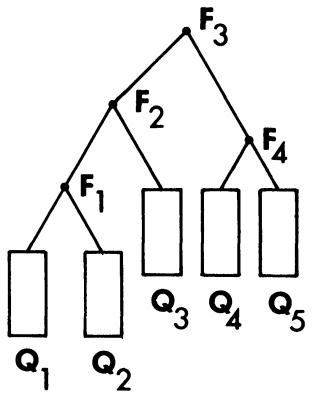


FIG. 1. *Example of binary tree with $k=5$ processors. F_i: Fences, Q_i: Partitions.*

sorting in the k processors is the time to finish the largest data partition

$$Q = \max_{1 \leq i \leq k} (Q_i), \tag{2.1}$$

and the total processing time T for sorting is

$$\begin{aligned} T = {} & \text{sampling time} + \text{fence construction time} + \text{partition time} \\ & + \text{parallel sorting time} + \text{concatenation time.} \end{aligned} \tag{2.2}$$

Other ways for computing the total processing time obviously exist and they will be discussed in § 6.

A natural way to construct the fence is to use the order statistics of the sample. Let the sample size be m, all data distinct, and its order statistic be

$$Y_1 < Y_2 < \cdots < Y_m. \tag{2.3}$$

In order to minimize (2.1), it is necessary to make the Q_i's as even as possible. Thus, one should choose a sample so that the numbers of the sampled data in the gaps between consecutive fences are the same. Consequently, one should always let $m = nk-1$ for some positive n and let the fences be

$$Y_n, Y_{2n}, \cdots, Y_{(k-1)n}. \tag{2.4}$$

A theoretical justification for this choice is given later in Theorem 1. Since (2.4) depends on the sorted sample (2.3), we let the fence construction time be $c(nk-1)\log(nk-1)$. The sampling time should be proportional to the sample size and we let it be $c_1(nk-1)$, where c_1 is the time required to take one random sample. The partitioning time for N pieces of data requires approximately $N \log_2 k$ number of comparisons; let it be $c_2 N \log_2 k$. The concatenation time for ordered data set already sorted is very small and can be ignored. Hence, (2.2) becomes

$$T = c_1(nk-1) + c(nk-1)\log(nk-1) + c_2 N \log_2 k + cQ \log Q. \tag{2.5}$$

The optimal sorting is obviously to find the right n so that (2.5) is minimized. Since T is a random variable, we choose to minimize its expected value which will still be denoted as T for convenience. It can easily be seen that the main task of minimizing T is to find the expected value of $Q \log Q$. This will be derived in §§ 3 and 4.

3. Small sample properties. Without loss of generality, let the N pieces of data be $1, 2, 3, \cdots, N$. Then the Q's defined in (2.1) based on a sample of size $m = nk-1$ are

$$Q_i = Y_{in} - Y_{(i-1)n} - 1, \qquad i = 1, 2, \cdots, k, \tag{3.1}$$

where $1 \equiv Y_0 < Y_1 < Y_2 < \cdots < Y_n < \cdots < Y_{2n} < \cdots < Y_{kn-1} < Y_{kn} \equiv N$ are the order statistic of a sample of size $kn-1$ from $\{1, 2, \cdots, N\}$. The distribution of Q_i for the special case when $n = 1$ has been treated in Frazer and McKellar [10]. The following theorem defines the distribution of the Q's in (3.1).

THEOREM 1. *The Q_i's defined in* (3.1) *are identically distributed with*

$$P(Q_i = j) = \frac{\binom{N-j-1}{(k-1)n-1}\binom{j}{n-1}}{\binom{N}{nk-1}}, \qquad j = n-1, n, \cdots, N-n(k-1). \tag{3.2}$$

COROLLARY 1. $EQ_i = (N-k+1)/k$ *and* $\operatorname{Var}(Q_i) = (N-nk+1)(k-1)/\{k(nk+1)\}$.

All the proofs of theorems in this paper are given in § 7. This theorem confirms the fact that the Q_i's defined this way are as even as possible from a statistical point of view and consequently produce the smallest possible Q.

Unfortunately, the Q_i's are not independent. It seems that there is no simple expression for the distribution of $Q = \max(Q_i)$. However, for small k and N, its distribution can be computed by the following recursive formula.

THEOREM 2. *Let*

$$p(N, n, k, j) = \Pr\{Q = j \mid \textit{data size } N \textit{ with sample size } nk-1\}$$

$$P(N, n, k, j) = \Pr\{Q \leqq j \mid \textit{data size } N \textit{ with sample size } nk-1\}$$

$$= \sum_{i=j}^{N-n(k-1)} p(N, n, k, i).$$

Then

$$p(N, n, k, j) = \sum_{m=1}^{a} \binom{k}{m} \frac{\binom{j}{n-1}^m \binom{N-mj-m}{n(k-m)-1}}{\binom{N}{nk-1}} P(N-mj-j, n, k-m, j-1), \tag{3.3}$$

with

$$a = \left[\frac{N-nk+1}{j+1-n}\right], \qquad j = 1 + \left[\frac{N-k+.5}{k}\right], \cdots, N-n(k-1).$$

Hence $[x]$ denotes the integer part of x.

Since (3.3) has no recursion in n, its computation only involves three indices: N, k and j. Hence, it is no problem to compute the p's for moderate N and small k in mainframe computers. Once the distribution of Q is computed, the value $E(Q \log Q)$ can be computed easily. Table 2 gives some of the values computed by (3.3). The details of Table 2 will be discussed in § 5.

4. Asymptotic formula for $E(Q \log Q)$. In this section, we let the values of the data be

$$\left\{\frac{1}{N}, \frac{2}{N}, \cdots, \frac{j}{N}, \cdots, \frac{N}{N}\right\}.$$

This expression again does not lose the generality. Let the order statistics be

$$X_1(N) < X_2(N) < \cdots < X_{nk-1}(N)$$

and define $X_0(N) = 1/N$, $X_{nk}(N) = 1$, and

$$\begin{aligned} &L_i(N) = X_{in}(N) - X_{(i-1)n}(N) - 1/N, \qquad i = 1, 2, \cdots, k, \\ &L(N) = \max_{1 \le i \le k} (L_i(N)). \end{aligned} \tag{4.1}$$

Then

$$Q \log Q = NL \log (NL) = (N \log N)L + N(L \log L). \tag{4.2}$$

In this section, our main purpose is to derive asymptotic formulae for $E(L)$ and $E(L \log L)$.

THEOREM 3. *Let $m = nk - 1$, $X_1 < X_2 < \cdots < X_m$ be the order statistics of m random observations from the uniform $(0, 1)$ distribution, and*

$$\begin{aligned} &D_0 = X_1, \ D_m = 1 - X_m, \\ &D_i = X_{in} - X_{(i-1)n} \quad \textit{for } i = 2, 3, \cdots, k-1, \\ &D = \max_{1 \le i \le k} (D_i). \end{aligned} \tag{4.3}$$

Then

$$\lim_{N \to \infty} E(L) = E(D) \quad \textit{and} \quad \lim_{N \to \infty} EL \log L = ED \log D.$$

Although this result seems intuitive, we could not find any proof of it. Its proof and related literature are given in § 7.

It is well known that the joint distribution of $(D_1, D_2, \cdots, D_k)$ is Dirichlet with parameters $(n, n, \cdots, n)$ (see Johnson and Kotz [13, p. 231]). The computation of the distribution of $D = \max (D_i)$, in general, involves either k-fold integration or k-fold summation which makes the value $ED \log D$ difficult to find for $k > 3$ (see Tiao and Guttman [21]). However, for large N and small k, n tends to be large also. The following theorem gives a simple asymptotic formula for ED and for $ED \log D$.

THEOREM 4. *Let $(Z_1, Z_2, \cdots, Z_k)$ be a k-variate independent standard normal random vector, $Z_{(k)} = \max_{1 \le i \le k} (Z_i)$, and $\mu_{(k)} = EZ_{(k)}$. Then for large n*

$$ED \doteq \frac{1}{k} + \frac{1}{k\sqrt{n}} \mu_{(k)} \quad \textit{and} \quad ED \log D \doteq -\frac{\log k}{k} + \frac{|1 - \log k|}{k\sqrt{n}} \mu_{(k)}. \tag{4.4}$$

Tables for $\mu_{(k)}$ can be found in CRC Table [7, p. 115] and David [8]. Combining (4.2) and (4.4), we have

$$\begin{aligned} EQ \log Q &= N \log N(1 + \mu_{(k)}/\sqrt{n})/k + N(-\log k + |1 - \log k| \mu_{(k)}/\sqrt{n})/k \\ &= (N/k) \log (N/k) + N\mu_{(k)}(\log N + |1 - \log k|)/k\sqrt{n}). \end{aligned} \tag{4.5}$$

This expression for $EQ \log Q$ is quite simple, but the question now is how good this approximation is for moderate N and n. We found that for moderate N and n, it is

better to add a finite population correction factor in (4.5), i.e., replace N by $N^* = N-k+1$, and multiply $[1-(nk-1)/N]^{1/2}$ to the last term of (4.5). The reason for the first correction is due to the fact that only N^* pieces of data are actually partitioned, for the fences are retained during the partition process. The second multiplier on $\mu_{(k)}$ is the commonly used finite population correction factor in sampling theory (see, e.g., Cochran [6, p. 24]). Without these two small modification factors, the relative errors would be slightly larger in Table 2. The final form of $EQ \log Q$ becomes

$$EQ \log Q \doteq \left(\frac{N^*}{k}\right) \log \left(\frac{N^*}{k}\right) + N^* \mu_{(k)} \left(1 - \frac{|nk-1|}{N}\right)^{1/2} (\log N + |1 - \log k|)/(k\sqrt{n}). \tag{4.6}$$

5. Optimal choice of sample size. With the formulae developed in the previous sections, finding the optimal sample size $nk-1$ for given k and N is quite straightforward. To have some idea of the relationship between the optimal n for given N and k, (2.5) is minimized for a large choice of c_1, k and N. Here we present mainly the results for $c_1 = 0$, which is close to the situation when a systematic sample is chosen. Since $N \log k$ is independent of n, it can be ignored in the optimalization process. Thus, to find the optimal n is equivalent to finding n so that

$$T' = (nk-1) \log (nk-1) + EQ \log Q \tag{5.1}$$

is minimized. For fixed k, the relation between the optimal n, denoted by $\hat{n}$, and N can be best represented on log scales. An illustration is given in Fig. 2. The coefficients for the linear regression equations

$$\log_{10} \hat{n} = a + b \log_{10} N + \varepsilon \tag{5.2}$$

for various k's are given in Table 1. For other values of c_1/c, the linear relationship still holds. All the R^2 values for the regression equations are in the neighborhood of 0.98. An illustration is given in Fig. 3 where $c_1/c = 4$, $c \equiv 1$ and $k = 2$, 10, 50.

Table 2 illustrates some details of the relationship between $\hat{n}$, and N for $k = 2$. The exact T' defined by (5.1) were computed by (3.3), and approximate T's were

TABLE 1

Regression coefficients in (5.2) *for the curves in Fig.* 2.

k	a	b
2	−0.627	0.663
3	−0.374	0.660
4	−0.789	0.652
5	−0.836	0.646
6	−0.877	0.641
7	−0.912	0.636
8	−0.945	0.632
9	−0.973	0.628
10	−1.267	0.665
20	−1.430	0.640
30	−1.451	0.623
40	−1.599	0.616
50	−1.593	0.599
100	−2.057	0.616
1000	−2.610	0.620

TABLE 2

Some detailed tabulation of T' in (5.1) by the exact formula and the asymptotic formula for $k = 2$.

N	n	Exact	Approx.	Rel. Error (vs. $N \log N$)
50	1	134.3149	135.8220	0.0077
50	2	122.2827	121.4521	0.0042
50	3	119.3034 (√)	118.2033 (√)	0.0056
50	4	119.9825	118.8999	0.0053
50	5	126.5955	121.6460	0.0253
100	1	322.5684	329.3622	0.0148
100	2	291.5940	291.7846	0.0003
100	3	279.2073	278.2338	0.0021
100	4	274.0343	272.7807	0.0027
100	5	272.7252 (√)	271.3268 (√)	0.0028
100	6	273.5175	272.2509	0.0028
100	7	275.9712	274.7556	0.0026
500	4	1845.82	1846.37	0.0002
500	5	1804.55	1803.59	0.0003
500	6	1775.56	1773.99	0.0005
500	7	1754.55	1752.78	0.0006
500	12	1709.13	1707.98	0.0004
500	13	1706.61	1705.66	0.0003
500	14	1705.40	1704.65 (√)	0.0002
500	15	1705.28 (√)	1704.74	0.0002
500	16	1706.10	1705.76	0.0001
1000	15	3720.03	3718.58	0.0002
1000	16	3711.66	3710.44	0.0002
1000	21	3690.47	3690.38	0.00001
1000	22	3689.30	3689.42	0.00002
1000	23	3688.90 (√)	3689.24 (√)	0.00005
1000	24	3689.20	3689.75	0.00008
1000	25	3690.14	3690.89	0.0001

(√) Indicates where minimums occur.

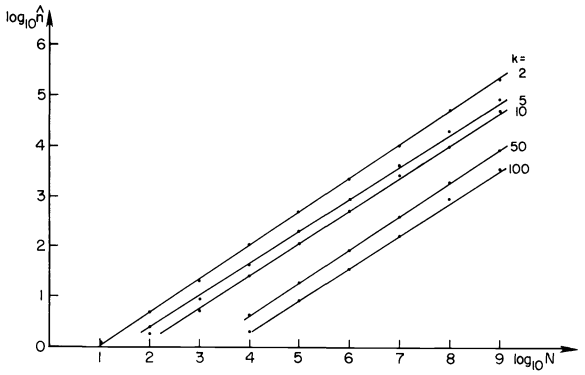


FIG. 2. *The relation between $\hat{n}$ and N for fixed k when $c_1 = 0$.*

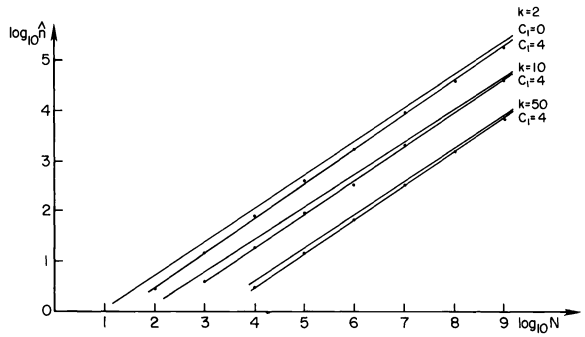


FIG. 3. *The relation between $\hat{n}$ and N for $c_1 = 0$ and $c_1 = 4$. (The top lines are for $c_1 = 0$ and the bottom ones are for $c_1 = 4$.)*

computed by (4.6). The place with a check mark ($\surd$) indicates the place where minimum T' occurs. For example, when $N = 50$, the optimal choice of n is $\hat{n} = 3$ with $T = 119.3034$ by the exact formula and 118.2033 by the approximation formula. From a large amount of comparisons of the exact and approximation values, no two corresponding $\hat{n}$'s were found which have a difference of more than 1. Actually, the method of finding the optimal n by (5.1) is to set its derivative with respect to n to zero. Thus, the $\hat{n}$ found this way is not generally an integer and, consequently, a difference of one to the exact optimal value should not be considered as a discrepancy. Moreover, it can be seen that the optimal n always appears at a flat plateau. Thus, a slight discrepancy produced by the regression formula in Table 1 has little effect in practice.

As one can see from Table 2, the approximation becomes better as n becomes larger. The errors are measured relative to $N \log N$ because only the terms L and $L \log L$ in (4.2) are approximated. Furthermore, the errors of practical concern should also be relative errors. From Table 2 one should feel safe to use the approximation formula for practically all N, k and n.

An important question for distributed computation, such as parallel sorting in a local area network environment, is how to allocate the processors for various tasks efficiently. Increasing the number of processors adds more parallelism to the system. However, it also increases the overhead and, in the case of parallel sorting, the performance of the partition algorithms will degrade. Thus, there is optimal choice of the maximum number of processors that should be used for sorting. It can be shown that:

THEOREM 5. *If the number of processors is allowed to increase without limit, then a necessary condition for T' to be minimum is $n = 1$.*

Note that the condition $n = 1$ is only necessary. Given $n = 1$, T' will not always decrease as k increases in the present sampling scheme. It will reach a valley. Moreover, the factor c_2 should be considered when T in (2.5) is used. We will not pursue this matter any further since for $n = 1$, k has to be very large and some other efficient sorting schemes such as those mentioned in the Introduction should be considered. However, it is useful to know the amount of time saved when additional processors are added for sorting even when the number of processors are limited. Computational results have shown that the marginal profit decreases rapidly as k increases. Let $\hat{T}'$ denote the computing time (5.1) at optimal n. Figure 4 gives the ratio

$$\hat{T}'/(N \log N) \tag{5.3}$$

as a function of k for fixed N. For example, when $N = 1000$, the first 9 processors

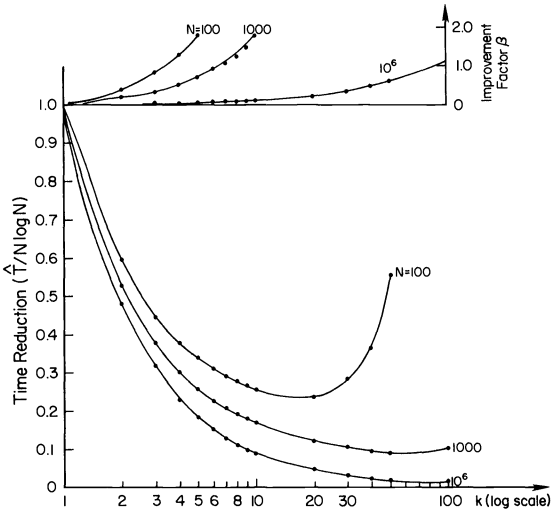


FIG. 4. *Time reduction and improvement factor as functions of N and k.*

reduce the sorting time by 82.5% while an additional 10 processors can reduce the total time only by an additional 4.5%.

Now we consider the question of the optimality of the present sampling method. There may exist other better algorithms that can be used to allocate the data. If we maintain that the sorting method in the microprocessors is fixed, then the absolute minimum sorting time for a data of size N is $c(N/k)\log(N/k)$, which is almost the first term of (4.6). The remaining terms of (4.6) can be considered as the overhead of the present method. We define the improvement factor as

$$\beta = \frac{\hat{T}' - (N/k)\log(N/k)}{(N/k)\log(N/k)}, \tag{5.4}$$

and obviously there is not much one can improve when β is close to 0. Values of β's are given in Fig. 4. For large N, small c_1, c_2 and k, the present method is approaching the absolute optimal. The assumptions of c_1 and c_2 being small are reasonable when a systematic sample is taken and when the partitioning time can be embedded into the sorting time when partitioning and sorting are performed simultaneously. More discussions will be presented in the next section.

6. Conclusion and concluding remarks. We have shown a complete analysis of the partitioning process by order statistics in the parallel balanced tree sort. To minimize the total processing time T for sorting N data on k processors using order statistics, the key question is to find the expected value of $Q \log Q$ where Q is the maximum Q_i, the size of an individual partitioned subfile. The optimal sample size for constructing the Q_i's can be computed numerically or approximated by the asymptotic formula derived in the paper. Another important result of the paper is the relation of sorting time as a function of k for given N. This is extremely useful in a system where there is a need to allocate the processors effectively and efficiently. Finally, the performance of the proposed sampling method is shown to approach the absolute optimal for large N.

One parameter which is also crucial for the performance of parallel sorting in local area network environment is the communication overhead among the cooperating processors. The proposed sorting scheme only requires data transfer in the partitioning phase of the sorting process. Thus, the communication overhead can be overlapped

with the parallel sorting on each local processor. The overall sorting time can then be expressed as the sum of sampling time, fence construction time, max (partitioning time, parallel sorting time), and concatenation time. If the time for each data transfer is relatively larger than the internal comparison time in the processor, and the local sorting can be started with partial data, then the parallel sorting time on each local processor can almost be completely embedded into the partitioning time. However, for small k and large N our computation shows that the parallel sorting time dominates the partitioning time. Thus, the assumption of $c_2 = 0$ at the end of the last section is quite reasonable.

It is noted that the analysis presented in this paper is still valid for the various c's used in the expression of the time complexity. The scheme does not assume any particular sorting algorithm used in each local processor. The sampling approach and the results of the analysis can be applied to any general local area network or multiprocessor system with distributed computation capability.

7. Proofs.

Proof of Theorem 1. Let $q_i(j)$ be the probability that $Y_i = j$. Then, it is easy to see that

$$q_i(j) = \binom{j-1}{i-1}\binom{N-j}{kn-i-1}\Big/\binom{N}{nk-1}.$$

Thus, by $P(Q_1 = j) = q_n(j+1)$ and $P(Q_k = j) = q_{(k-1)n}(N-j)$, Theorem 1 is proved for $i = 1$ and k by simple substitutions.

For $2 \leqq i \leqq k-1$,

$$P(Q_i = j) = \sum_{t=(i-1)n}^{N-(k-i)n-j} q_{(i-1)n}(t)\, q_{in}(t+j+1 \mid Y_{(i-1)n} = t)$$

$$= \sum_{t=(i-1)n}^{N-(k-i)n-j} \frac{\binom{t-1}{(i-1)n-1}\binom{N-t}{(k-i+1)n-1}}{\binom{N}{kn-1}} \cdot \frac{\binom{j}{n-1}\binom{N-t-j-1}{(k-i+1)n-n-1}}{\binom{N-t}{(k-i+1)n-1}}$$

$$= \frac{\binom{N-j-1}{(k-1)n-1}\binom{j}{n-1}}{\binom{N}{kn-1}}$$

where $q_{in}(t+j+1 \mid Y_{(i-1)n} = t)$ denotes the probability $q_n(j+1)$ for a sample of size $(k-i+1)n-1$ from a set of size $N-t$. The computations of the mean and variance of this distribution are quite straightforward and hence omitted here.

Proof of Theorem 2. It can be easily seen that

$$p(N, n, k, j) = \sum_{m=1}^{a} \Pr\{m\ Q_i\text{'s are } j \text{ and the rest are less than } j\} \tag{7.1}$$

and there are $\binom{k}{m}$ possible mutually exclusive ways of m Q_i's being j. It can also be seen that

$$\Pr\{Q_1 = Q_2 = \cdots = Q_m = j \text{ and } \max_{i>m}(Q_i) < j\}$$

$$= \sum_{A} \binom{j}{n-1}^m \binom{j_1}{n-1} \cdots \binom{j_k}{n-1}\Big/\binom{N}{nk-1} \tag{7.2}$$

$$= \frac{\binom{j}{n-1}^m}{\binom{N}{nk-1}} \sum_A \binom{j_1}{n-1}\binom{j_2}{n-1}\cdots\binom{j_k}{n-1},$$

where A is the set $\{j_i < j, i = m+1, \cdots, k, \Sigma j_i = N - mj - (k-1)\}$. By $N - mj - (k-1) = N - m(j+1) - \{(k-m)-1\}$, (7.2) becomes

$$\frac{\binom{j}{n-1}^m \binom{N-mj-m}{n(k-m)-1}}{\binom{N}{nk-1}} P(N-mj-m, n, k-m, j-1).$$

Formula (3.3) is proved when the above formula is substituted into (7.1).

Proof of Theorem 3. Let $X_1, X_2, \cdots, X_m$ be a sample (unordered) from $(1/N, 2/N, \cdots, N/N)$ and $Y_1, Y_2, \cdots, Y_m$ be a sample (unordered) from $(0, 1)$. Then it can be shown that the following three mappings, g_1, g_2 and g_3 are continuous:

$$(X_1, X_2, \cdots, X_m)(\text{unordered}) \xrightarrow[g_1]{} X_1 < X_2 < \cdots < X_m (\text{ordered})$$

$$\downarrow g_2$$

$$(L_1, L_2, \cdots, L_m)$$

$$\downarrow g_3$$

$$L = \max(L_i).$$

If we let $g = g_3 \circ g_2 \circ g_1$, then g is a continuous mapping. By a well-known theorem in probability (see e.g., Serfling [19, p. 24]), we know that if

$$(X_1, X_2, \cdots, X_m) \to (Y_1, Y_2, \cdots, Y_m) \tag{7.3}$$

in distribution, then $g(X_1, \cdots, X_m) \to g(Y_1, Y_2, \cdots, Y_m)$ in distribution, or $L \to D$ in distribution. To show (7.3), first let $X_1^*, X_2^*, \cdots, X_m^*$ be a random sample from $(1/N, 2/N, \cdots, N/N)$ *with* replacement. Then obviously for any m real numbers $(x_1, x_2, \cdots, x_m)$

$$\Pr\{X_1^* \leqq x_1, X_2^* \leqq x_2, \cdots, X_m^* \leqq x_m\} = \frac{[x_1 N]}{N} \cdot \frac{[x_2 N]}{N} \cdots \frac{[x_m N]}{N} \xrightarrow[N\to\infty]{} x_1 \cdot x_2 \cdots x_m$$

$$= \Pr\{Y_1 \leqq x_1, Y_2 \leqq x_2, \cdots, Y_m \leqq x_m\}$$

which established $(X_1^*, X_2^*, \cdots, X_m^*) \to (Y_1, Y_2, \cdots, Y_m)$ in distribution. To show that $\Pr\{X_1 \leqq x_1, X_2 \leqq x_2, \cdots, X_m \leqq x_m\}$ converges to $\Pr\{X_1^* \leqq x_1, \cdots, X_m^* \leqq x_m\}$ as $N \to \infty$ is simple because

$$\Pr\{X_1 \leqq x_1, \cdots, X_m \leqq x_m\}$$

$$= \Pr\{X_1 \leqq x_1, \cdots, X_m \leqq x_m \mid \text{there is no replicates}\} \cdot \Pr\{\text{no replicates}\}$$

$$+ \Pr\{X_1 \leqq x_1, \cdots, X_m \leqq x_m \mid \text{with replicates}\} \cdot \Pr\{\text{with replicates}\},$$

and

$$\Pr\{X_1 \leqq x_1, \cdots, X_m \leqq x_m \mid \text{no replicates}\}$$

$$= \Pr\{X_1^* \leqq x_1, \cdots, X_m^* \leqq x_m\}$$

and Pr {with replicates} is the classic occupancy problem (see, e.g., Johnson and Kotz [12]) and it converges to 0 as $N \to \infty$. The convergence of EL to ED and $EL \log L$ to

$ED \log D$ requires the uniform integrability of L and $L \log L$ with respect to N (for definition of uniform integrability, see Serfling [19, pp. 13-15]). Since L and $L \log L$ are both uniformly bounded for $0 \leqq L \leqq 1$, the uniform integrability holds and consequently Theorem 3 is proved.

Though no proof of this theorem is found in the literature, some related works are available. Among them are Sarndel [18] and Del Pino [9].

Proof of Theorem 4. By theorems of Karlin [14, Chap. 9], it can be seen that D_1, $D_2, \cdots, D_k$ defined in Theorem 3 can be written as $D_i = T_i/S$, where

$$T_1 = \sum_{i=1}^{n} E_i, \quad T_2 = \sum_{i=n-1}^{2n} E_i, \cdots, T_j = \sum_{i=(j-1)n+1}^{jn} E_i, \cdots, T_k = \sum_{i=(k-1)n+1}^{kn} E_i, \quad S = \sum_{j=1}^{k} T_j,$$

and $\{E_i\}$ are independent and identically distributed exponential random variables with expectation 1. Thus, the vector $(T_1, T_2, \cdots, T_k, S)'$ has mean vector $(1, 1, \cdots, 1, k)'$ and variance-covariance matrix

$$\Sigma = \begin{bmatrix} & & & 1 \\ & I & & \vdots \\ & & & 1 \\ 1 & \cdots & 1 & k \end{bmatrix},$$

where I is a $k \times k$ identity matrix. By the multivariate central limit theorem (see Serfling [19, Thm. B, 2nd formula, p. 28])

$$\frac{1}{n}\begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_k \\ S \end{bmatrix} \sim \mathrm{AN}\left(\begin{bmatrix} 1 \\ \vdots \\ 1 \\ k \end{bmatrix}, \frac{1}{n}\Sigma\right),$$

where AN states for asymptotic normal distribution. Using the theorem of Serfling [19, p. 122], we have

$$\begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_k \end{bmatrix} = \begin{bmatrix} T_1/S \\ T_2/S \\ \vdots \\ T_k/S \end{bmatrix} \sim \mathrm{AN}\left(\begin{bmatrix} 1/k \\ 1/k \\ \vdots \\ 1/k \end{bmatrix}, \frac{1}{n} J\Sigma J'\right)$$

$$\equiv AN\left(\begin{bmatrix} 1/k \\ 1/k \\ \vdots \\ 1/k \end{bmatrix}, \frac{1}{n}\Sigma^*\right),$$

where

$$J = \frac{1}{k}\left[I, -\frac{1}{k} l_k\right], \qquad l_k = [1, 1, \cdots, 1]',$$

$$\Sigma^* = \frac{k-1}{k^3}\begin{bmatrix} 1 & \rho & \rho & \cdots & \rho \\ \rho & 1 & \rho & \cdots & \rho \\ \vdots & & & & \vdots \\ \rho & & \cdots & \rho & 1 \end{bmatrix}, \qquad \rho = -\frac{1}{(k-1)}.$$

In other words,

$$\sqrt{n}\begin{bmatrix} D_1 - 1/k \\ D_2 - 1/k \\ \vdots \\ D_k - 1/k \end{bmatrix} \sim \text{AN}(0, \Sigma^*), \quad \text{and} \tag{7.4}$$

$$\sqrt{n}\begin{bmatrix} D_1 \log D_1 + \log k/k \\ D_2 \log D_2 + \log k/k \\ \vdots \\ D_k \log D_k + \log k/k \end{bmatrix} \sim \text{AN}(0, (1-\log k)^2\Sigma^*). \tag{7.5}$$

The second asymptotic distribution follows again from the theorem of Serfling [19, p. 122] with $g(x_i) = x_i \log x_i$. To show convergence in mean, we need the following results.

LEMMA 1. *Let $\zeta_i = (\zeta_{i1}, \zeta_{i2}, \cdots, \zeta_{ik})'$, $i = 1, 2 \cdots$ be a sequence of random vectors, and ζ_i converge to $\eta = (\eta_1, \cdots, \eta_k)$ in distribution. If $E|\zeta_{ij}|^2$ is uniformly bounded, then $E \max(\zeta_{i1}, \cdots, \zeta_{ik}) \to E \max(\eta_1, \cdots, \eta_k)$.*

The proof of this lemma follows from $|\max(\zeta_{i1}, \cdots, \zeta_{ik})|^2 \leqq \sum_{j=1}^{k} |\zeta_{ij}|^2$ and Theorem 4.5.2 of Chung [5, p. 85].

With the result of Lemma 1, Theorem 4 is proved if we can show that

$$\sup_n E(\sqrt{n}|D_1 - 1/k|)^2 < \infty, \quad \text{and} \tag{7.6}$$

$$\sup_n E(\sqrt{n}|D_1 \log D_1 + \log k/k|)^2 < \infty. \tag{7.7}$$

The methods of proving the above two facts are similar. Since the proof of the first is much easier than the second, only the second is presented. Since $(D_1, D_2, \cdots, D_k)$ has a Dirichlet distribution with parameters $(n, n, \cdots, n)$ (see reference in the text), D_1 has a beta distribution with parameters $(n, n(k-1))$, and finding its moments is routine work. Using Taylor's expansion for $D \log D$, we have

$$x \log x = -\frac{\log k}{k} - \left(x - \frac{1}{k}\right)[\log(\theta x) + 1] \quad \text{for } 0 < \theta < 1.$$

It can be shown, by considering all possible values of θ and x, that

$$|\log(\theta x) + 1|^2 \leqq |1 + \log k|^2 + \left(\frac{1}{2x}\right)^2.$$

Thus,

$$\begin{aligned} E(D_1 \log D_1 + \log k/k)^2 &\leqq E(1+\log k)^2 E\left(D_1 - \frac{1}{k}\right)^2 + E\left\{\left(D_1 - \frac{1}{k}\right)\frac{1}{2D_1}\right\}^2 \\ &= \frac{(1+\log k)^2(k-1)}{k^2(nk-1)} + \frac{nk(2k-1)+(2k+1)^2}{8k^2(n-1)(n-2)}, \end{aligned}$$

and consequently (7.7) is proved. Inequality (7.6) can be proved in a similar manner. Using (7.4) and (7.5), and the continuity of maximum function, we have

$$\frac{k\sqrt{k}}{\sqrt{k-1}} E(\sqrt{n}(D_1 - 1/k)) \to \mu^*_{(k)}, \quad \text{and} \tag{7.8}$$

$$\frac{k\sqrt{k}}{\sqrt{k-1}} \frac{1}{|1-\log k|} E(\sqrt{n}(D_1 \log D_1 + \log k/k)) \to \mu^*_{(k)} \tag{7.9}$$

where $\mu^*_{(k)}$ is the mean of the maximum of a k-variate unit normal random variables with equal correlations $\rho = -1/(k-1)$. The values of $\mu^*_{(k)}$ can be found in Owen and Steck [17], i.e.,

$$\mu^*_{(k)} = (1+1/(k-1))^{1/2}\mu_{(k)},$$

where $\mu_{(k)}$ is the mean of the largest order statistic of k independent unit normal random variables. Simple arithmetic leads (7.8) and (7.9) to (4.4). The theorem is proved.

Proof of Theorem 5. This theorem can easily be seen by contradiction. Suppose with unlimited number of processors, the smallest T' is obtained by using k processors with a sample $nk-1$ for some $n>1$. Then it is obvious that if $nk-1$ processors are used, the sorting time for the sample is the same but the number of elements in each processor is smaller at least by $n-1$. Thus, $n>1$ cannot be optimal.

Acknowledgments. The authors thank Professors D. E. Knuth, W. C. Parr and Malay Ghosh for their helpful discussions.

REFERENCES

[1] K. E. BATCHER, *Sorting networks and their applications*, Proc. 1968 Spring Joint Computer Conference, Vol. 32, AF/PS Press, Reston, VA, 1968, pp. 307-314.

[2] J. L. BENTLEY AND H. T. KUNG, *A tree machine for searching problems*, Proc. 1979 Internat. Conference on Parallel Processing.

[3] D. BITTON, D. J. DEWITT, D. K. HSIAO AND J. MENON, *A taxonomy of parallel sorting*, ACM Computing Survey, 16 (1984), pp. 287-318.

[4] T. C. CHEN, V. Y. LUM AND C. TUNG, *The rebound sorter: An efficient sort engine for large files*, IEEE Proc. 4th Interaction Conference on Very Large Data Bases, 1978, pp. 312-318.

[5] K. L. CHUNG, *A Course in Probability Theory*, 2nd ed., Academic Press, New York, 1974.

[6] W. G. COCHRAN, *Sampling Techniques*, 3rd ed., John Wiley, New York, 1977.

[7] CRC Handbook of Tables for Probability and Statistics, The Chemical Rubber Co., 1966.

[8] H. A. DAVID, *Order Statistics*, 2nd ed., John Wiley, New York, 1981.

[9] G. E. DEL PINO, *On the asymptotic distribution of k-spacings with applications to goodness-of-fit tests*, Ann. Statist., 7 (1979), pp. 1058-1065.

[10] W. D. FRAZER AND A. C. MCKELLAR, *Sample sort: A sampling approach to minimal storage free sorting*, J. Assoc. Comput. Mach., 17 (1970), pp. 496-507.

[11] D. S. HIRSCHBERG, *Fast parallel sorting algorithms*, Comm. ACM, 21 (1978), pp. 657-661.

[12] N. L. JOHNSON AND S. KOTZ, *Urn Models and their Applications*, John Wiley, New York, 1977.

[13] ———, *Distributions in Statistics: Continuous Multivariate Distributions*, John Wiley, New York, 1972.

[14] S. KARLIN, *A First Course in Stochastic Processes*, Academic Press, New York, 1968.

[15] D. E. MULLER AND F. P. PREPARATA, *Bounds and complexities of networks for sorting and for switching*, J. Assoc. Comput. Mach., 22 (1975), pp. 185-201.

[16] D. NASSIMI AND S. SAHMI, *Parallel permutation and sorting algorithms and a new generalized connection network*, J. Assoc. Comput. Mach., 29 (1982), pp. 642-667.

[17] D. B. OWEN AND G. P. STECK, *Moments of order statistics from the equicorrelation multivariate normal distribution*, Ann. Math. Statist., 33 (1962), pp. 1286-1291.

[18] C. E. SARNDAL, *Some properties of the negative hypergeometric distribution and its limit distributions*, Metrika, 13 (1968), pp. 171-189.

[19] R. J. SERFLING, *Approximation Theorems of Mathematical Statistics*, John Wiley, New York, 1980.

[20] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-22, 2 (1971), pp. 153-161.

[21] G. G. TIAO AND I. GUTTMAN, *The inverted Dirichlet distribution with applications*, J. Amer. Statist. Assoc., 60 (1965), pp. 793-805.

[22] L. E. WINSLOW AND Y. C. CHOW, *The analysis and design of some new sorting machines*, IEEE Trans. Comput., 32 (1983), pp. 677-683.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
003

FAST ALGORITHMS FOR SHORTEST PATHS IN PLANAR GRAPHS, WITH APPLICATIONS*

GREG N. FREDERICKSON†

Abstract. Graph decomposition and data structures techniques are presented that exploit the structure of planar graphs to yield faster algorithms for a number of shortest path and related problems. Improved algorithms are presented for the single source problem, the all pairs problem, the problem of finding a minimum cut in an undirected graph, and testing the feasibility of a multicommodity flow when all sources and sinks are on the same face. The algorithm for the single source takes $O(n\sqrt{\log n})$ time in an n-vertex graph, an improvement from $O(n \log n)$. The algorithm for all pairs takes $O(n^2)$ time, an improvement from $O(n^2 \log n)$. The algorithm for minimum cut takes $O(n \log n)$ time, an improvement from $O(n(\log n)^2)$. As a consequence, an algorithm for maximum flow is similarly improved.

Key words. all pairs shortest paths, decision trees, heaps, maximum flow, minimum cut, multicommodity flow, planar graph, planar separator, single source shortest paths

AMS(MOS) subject classification. 68Q

1. Introduction. Algorithms that are efficient on sparse graphs will do well on planar graphs, since the number of edges in an n-vertex planar graph is $O(n)$. But in what ways can the planarity of a graph be exploited, beyond taking advantage merely of the sparsity? We examine this question by considering the problem of finding shortest paths in planar graphs. We present algorithms faster than those previously known for solving the single source shortest paths, the all pairs shortest paths, and several related problems. The work highlights the effect that planarity has on issues of not only planar graph decomposition but also data structures.

Finding shortest paths is a fundamental and well-studied problem in applied graph theory [AHU], [DP]. For the single source problem on a directed or undirected graph with nonnegative edge costs, Dijkstra's algorithm takes $O(n^2)$ time [D]. An implementation of Dijkstra's algorithm that uses a heap improves this to $O(n \log n)$ time for any graph with $O(n)$ edges [J]. Our algorithm for the single source problem in a planar graph uses $O(n\sqrt{\log n})$ time. For the all pairs problem on an undirected graph with nonnegative edge costs, the best previous algorithm runs the heap version of Dijkstra's algorithm n times, each time with a different vertex as the source, and thus uses $O(n^2 \log n)$ time altogether on a planar graph. Using results of [LRT] and [EK], [J], [Ne], [Ni], this strategy can be extended to handle directed graphs with negative cost edges but no negative cycles in the same time bound. We present an algorithm for both variants of the all pairs problem that uses $O(n^2)$ time. This result is optimal, if the output is required to be in the straightforward form.

The best previous algorithms for finding an s-t minimum cut [R] and an s-t maximum flow [HJ] in an undirected planar graph use a number of shortest path computations. We show how to adapt our shortest path techniques to improve these algorithms from $O(n(\log n)^2)$ to $O(n \log n)$ time. Algorithms for determining feasibility in a multicommodity flow problem in which all terminals are on the same face of a planar graph have been presented in [Hs2], [MNS]. We show how to adapt our techniques to yield a faster feasibility test.

* Received by the editors July 23, 1984; accepted for publication (in revised form) January 5, 1987. This research was supported in part by the National Science Foundation under grants MCS-8201083 and DCR-8320124. A preliminary version of this article appeared in the Proceedings of the 24th IEEE Symposium on the Foundations of Computer Science, held in Tucson, Arizona, November 1983.

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

Our algorithms are based on Dijkstra's single source algorithm, which we now describe briefly. Dijkstra's algorithm performs a search of the graph that proceeds in iterations. For each vertex v whose shortest distance $d(v)$ from the source s is not known, the vertex is termed *open*, and the currently known shortest distance $\rho(v)$ from the source to v is maintained. Initially, all vertices are open, $\rho(s)=0$, and $\rho(v)=\infty$ for all other vertices v. On each iteration, the open vertex v with minimum $\rho(v)$ is closed, and the shortest distances $\rho(w)$ are updated for all w such that there is an edge (v, w). The distances $\rho(v)$ can be maintained in a heap, giving $O(\log n)$ time per update. Since there are $O(n)$ updates in handling a planar graph, the total time is $O(n \log n)$.

We improve on Dijkstra's algorithm by conducting the iterative search on a carefully selected subset of the vertices. This means that a *preprocessing phase* is necessary to identify this subset of vertices. This phase must also determine shortest paths between all pairs of vertices in this subset, where intermediate vertices on these paths are not in the subset. The *search phase* consists of two parts. During the *main thrust* of the search phase, when a vertex v in the special subset is closed, $\rho(w)$ must be updated for all vertices w in the subset such that a path of the above type exists from v to w. At the end of the main thrust, the shortest distances are known from the source to each vertex in the subset. The *mop-up* portion of the search phase then determines the shortest distance to every remaining vertex.

For the main thrust of the search phase to be efficient, the vertices in the subset must separate the graph into a number of regions of convenient size and favorable adjacency properties. A planar separator algorithm [LT1] can be used in separating a planar graph into regions. However, a straightforward use of this algorithm is not adequate to generate regions with appropriate characteristics. We thus contribute interesting results for planar graph decomposition.

A second idea that makes the main thrust of the search phase efficient involves the design of appropriate data structures. When a vertex v is closed, many distances $\rho(w)$ may need to be updated in the heap. It turns out that the total number of updates performed during the search phase is in worst cast at least proportional to the number performed in Dijkstra's algorithm. Fortunately however, the updates involve the same region. We present a heap whose organization is based on the adjacency of regions and thus allows a batch of related updates to be handled more efficiently.

Our algorithms for all pairs shortest paths and minimum cut in a planar graph rely in addition on performing extensive preprocessing of the graph, that allows for a number of very fast searches. The preprocessing constructs decision trees to identify the portions of a shortest path tree within each of many very small regions. The decision trees implement a divide-and-conquer approach that splits a region based on shortest paths to a set of separator vertices.

The paper is organized as follows. In §§ 2 and 3 we show how to divide the graph into regions and identify the special subset of vertices on which to perform the main thrust. In § 4 we present an efficient data structure for performing the main thrust. Section 5 gives the single source algorithm. Sections 6 and 7 describe one-time preprocessing and the decision trees that allow for very efficient searches. The improvements to finding minimum cuts, maximum flows, and multicommodity flows are discussed in §§ 8 and 9.

2. Regions and boundary vertices. Our shortest path algorithms make use of a division of the planar graph into regions. A *region* will contain two types of vertices, boundary vertices and interior vertices. An *interior vertex* is contained in exactly one

region, and is adjacent only to vertices contained in that region, while a *boundary vertex* is shared among at least two regions. An example of a graph which has been divided into regions is shown in Fig. 1. The boundary vertices are shown in bold, and the regions are circled. To generate appropriate regions, we make use of a linear-time algorithm of Lipton and Tarjan that finds a planar separator [LT1]. Let the vertices have nonnegative vertex costs summing to no more than one. The separator algorithm partitions the vertices of G into three sets A, B, and C such that no edge joins a vertex in A with a vertex in B, neither A nor B has total cost exceeding $\frac{2}{3}$, and C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.

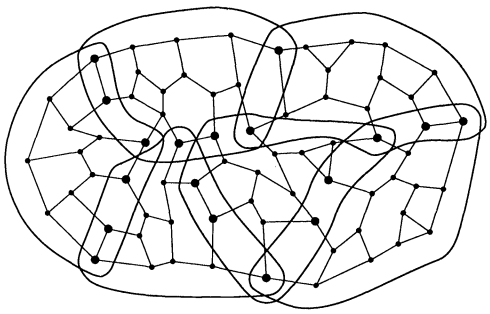


FIG. 1. *Division of a planar graph into regions.*

Given a parameter r, we first discuss how to generate $\Theta(n/r)$ regions with $O(r)$ vertices each, and $O(n/\sqrt{r})$ boundary vertices in total. Initially, G consists of one region with all vertices interior. Apply the separator algorithm to the graph with all vertex weights equal to $1/n$, yielding sets A, B and C. Infer two regions with vertex sets $A_1 \subseteq A \cup C$ and $A_2 \subseteq B \cup C$, of sizes $\alpha n + O(\sqrt{n})$ and $(1-\alpha)n + O(\sqrt{n})$, where $\frac{1}{3} \leqq \alpha \leqq \frac{2}{3}$. Recursively apply the procedure on the subgraph induced by any region with more than r vertices. The total time required will be $O(n \log (n/r))$. This approach is similar to that described in [LT2].

LEMMA 1. *An n-vertex planar graph can be divided into $\Theta(n/r)$ regions with no more than r vertices each, and $O(n/\sqrt{r})$ boundary vertices in total.*

Proof. Let the above method be applied to a graph. For any boundary vertex v, let $b(v)$ be one less than the number of regions that contain v in the division. Let $B(n, r)$ be the total of $b(v)$ over all boundary vertices v. Thus $B(n, r)$ is the sum of the number of boundary vertices v weighted by the count $b(v)$. From the above discussion, we have the following recurrence:

$$B(n, r) \leqq c\sqrt{n} + B(\alpha n + O(\sqrt{n}), r) + B((1-\alpha)n + O(\sqrt{n}), r) \quad \text{for } n > r,$$

$$B(n, r) = 0 \quad \text{for } n \leqq r$$

where $c = 2\sqrt{2}$ and $\frac{1}{3} \leqq \alpha \leqq \frac{2}{3}$. Then we claim that

$$B(n, r) \leqq cn/\sqrt{r} - d\sqrt{n}$$

for some constant d. The claim can be shown by induction.

There are $\Theta(n/r)$ regions, since the sum of the number of vertices in each region is $n + B(n, r) = n + O(n/\sqrt{r})$, and each region is of size $\Theta(r)$. □

An *r-division* is a division into $\Theta(n/r)$ regions of $O(r)$ vertices each and $O(\sqrt{r})$ boundary vertices each. An r-division may be obtained from the preceding division

in the following way. While there is a region that has more than $c\sqrt{r}$ boundary vertices, for some constant c, apply the planar separator algorithm to the region, with the n' boundary vertices each having weight $1/n'$, and interior vertices having weight zero. Infer the two resulting regions as before.

LEMMA 2. *A planar graph of n vertices can be divided into an r-division in* $O(n \log n)$ *time.*

Proof. We claim that the above method generates the desired division. Consider a division before regions are split to enforce the requirement of $O(\sqrt{r})$ boundary vertices per region. Let t_i be the number of regions with i boundary vertices. From the proof of the previous lemma, note that $\sum_i it_i = \sum_{v \in V_B} (b(v)+1)$, where V_B is the set of boundary vertices. It follows that $\sum_{v \in V_B} (b(v)+1) < 2B(n, r)$, which is $O(n/\sqrt{r})$. For a region with i boundary vertices, where $i > c\sqrt{r}$, at most $di/(c\sqrt{r})$ splits need be done, for some constants c and d. This will result in at most $1 + di/(c\sqrt{r})$ regions, and at most $c\sqrt{r}$ new boundary vertices per split. Thus the number of new boundary vertices is at most

$$\sum_i (c\sqrt{r})(di/(c\sqrt{r}))t_i \leqq d \sum_i it_i = O(n/\sqrt{r}).$$

The number of new regions will be at most

$$\sum_i (di/(c\sqrt{r}))t_i = O(n/r). \qquad \square$$

It is convenient to have no boundary vertex be in more than some constant number of regions. Thus before any other preprocessing, we transform the initial planar graph G_0 into planar graph G with no vertex having degree greater than 3. A well-known transformation in graph theory [Hr, p. 132] may be used to generate G from G_0. Consider a planar embedding of G_0. For each vertex v of degree $d > 3$, where $w_0, \cdots, w_{d-1}$ is a cyclic ordering of the vertices adjacent to v in the planar embedding, replace v with new vertices $v_0, \cdots, v_{d-1}$. Add edges $\{(v_i, v_{(i+1) \bmod d}) | i = 0, \cdots, d-1\}$, each of distance 0, and replace the edges $\{(w_i, v) | i = 0, \cdots, d-1\}$ with $\{(w_i, v_i) | i = 0, \cdots, d-1\}$, of corresponding distances. From a corollary of Euler's formula [Hr], the number of vertices in the resulting graph will be $n \leqq 6n_0 - 12$, where n_0 is the number of vertices in G_0.

3. Suitable graph divisions. To facilitate efficient execution of heap updates in the search phase of our algorithm, it is helpful to have each region share boundary vertices with relatively few other regions. Two problems arise with the division as generated in the previous section. First, a boundary vertex may be in a large number of regions, even though its degree is limited to three. Second, the regions generated by the previous method are not necessarily connected, so that the effect of locality provided by the planarity of the graph may be lost. A *suitable r-division* of a planar graph is an r-division such that

(1) each boundary vertex is contained in at most three regions, and

(2) any region that is not connected consists of connected components, all of which share boundary vertices with exactly the same set of either one or two connected regions.

The following strategy ensures that no boundary vertex is in more than three regions. Consider an application of the planar separator algorithm, which yields sets A, B and C. Let C' be the set of vertices in C not adjacent to any vertex in $A \cup B$,

and let $C'' = C - C'$. Identify the connected components $A_1, A_2, \cdots, A_q$ in $A \cup B \cup C'$. Any vertex v in C'' adjacent to a vertex in A_i and not adjacent to a vertex in A_j for $j \neq i$ can be removed from C'' and inserted into A_i. We term the resulting subgraphs *connected subgraphs.* Connected subgraph i will have interior vertices A_i and boundary vertices in C'' that are adjacent to some vertex in A_i. The time to find and augment the connected subgraphs will be linear. A boundary vertex will be in at most three connected subgraphs, since the degree is at most three, and a vertex is included as a boundary vertex of a connected subgraph only if it is adjacent to an interior vertex in the connected subgraph.

More than $\Theta(n/r)$ connected subgraphs can result when the above strategy is applied recursively to each connected subgraph with more than r vertices. To generate $\Theta(n/r)$ regions from the set of connected subgraphs, do the following in a greedy fashion. Initialize each region to be a connected subgraph. While there are two regions that share a common boundary vertex and each has no more than $r/2$ vertices, union the regions together. It is still possible that there may be more than $\Theta(n/r)$ regions. While there are two regions that each have no more than $r/2$ vertices, and are adjacent to the same set of either one or two regions, combine them. This procedure can be performed in linear time.

The resulting division will have each region being either connected or the union of connected subgraphs which share boundary vertices with the same set of regions. Examples of regions that are unions of connected subgraphs are shown in Fig. 2. Each component of region C is adjacent only to region A, and each component of region D is adjacent to both region A and region B.

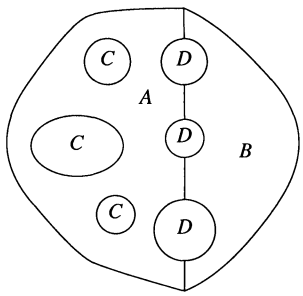


FIG. 2. *Examples of regions that are unions of connected components.*

Of course, it is also required that no region have more than $c\sqrt{r}$ boundary vertices, for some constant c. However this constraint is not a problem. Before initializing the regions to be the connected subgraphs, apply the separator strategy to any connected subgraph with more than $c\sqrt{r}$ boundary vertices. The regions are then initialized to be the resulting subgraphs, and combined in a fashion similar to that described above. But now a region can be combined with another if it has at most $r/2$ vertices and at most $c\sqrt{r}/2$ boundary vertices.

THEOREM 1. *A planar graph of n vertices can be divided into a suitable r-division in $O(n \log n)$ time.*

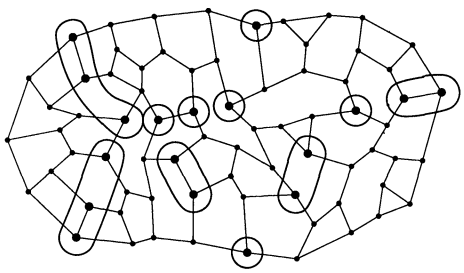
Proof. It is clear that the above strategy generates regions with at most r vertices and $c\sqrt{r}$ boundary vertices. We must establish that there will be $\Theta(n/r)$ regions in the division. Consider a graph, called the *region graph,* in which there is a node for each

region in the division, and an edge between two nodes if the corresponding regions share a boundary. The region graph is not explicitly constructed, but rather is a device for counting regions in the division. Call a node *small* if it represents a region that has at most $r/2$ vertices and at most $c\sqrt{r}/2$ boundary vertices, and *normal* otherwise. Since there are $O(n)$ vertices and $O(n/\sqrt{r})$ boundary vertices in total, there can be no more than $O(n/r)$ normal nodes.

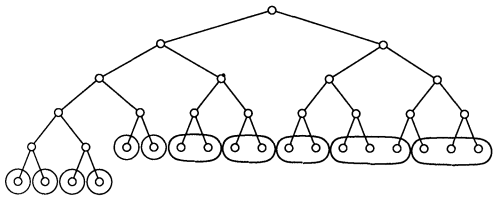
From the foregoing procedure, the following properties hold. Each small node is adjacent only to normal nodes. There can be no more than one small node of degree 1 adjacent to each normal node. There can be no more than one small node of degree 2 adjacent to any pair of normal nodes. Thus the number of small nodes of degree 2 is no greater than the number of edges in the reduced graph with each small node and its incident edges replaced by one edge. Hence the number of small nodes of degree 2 is in worst case proportional to the number of normal nodes. Consider the reduced graph with small nodes of degree one and two removed, and consider a planar embedding. Add as many edges as possible between normal nodes, and then delete the edges incident on small nodes. There will be at most one of these small nodes per planar face, of which there can be at most $O(n/r)$. Thus there are at most $O(n/r)$ small nodes of all degrees in the reduced graph.

The time bound follows, since the time to apply the separator strategy recursively is $O(n \log n)$, and the time to perform the combining is $O(n)$. □

4. A topology-based heap and the batched update operation. We now show how to organize a heap based on a suitable r-division. Partition the boundary vertices into *boundary sets*, which are maximal subsets such that every member of a set is shared by exactly the same set of regions. The boundary sets corresponding to the boundary vertices in Fig. 1 are circled in Fig. 3(a). Every boundary set contained in exactly two regions will correspond to an edge in the region graph discussed in the proof of Theorem 1. As for boundary sets contained in three regions, no region can contain



(a)



(b)

FIG. 3. (a) *Boundary sets for the regions of Fig.* 1, *and* (b) *the corresponding topology-based heap.*

more such boundary sets than there are edges incident on its node in the region graph. Since the region graph is planar, and there are $\Theta(n/r)$ nodes in it, there are $\Theta(n/r)$ edges. Thus there are $\Theta(n/r)$ boundary sets.

The boundary sets can be identified as follows. Assume that the regions are indexed from 1 up to the number of regions. For each boundary vertex, generate an ordered list of the either two or three regions in which the vertex is located. Now sort the set of lists, using a lexicographic sort. All boundary vertices in the same boundary set will have their lists appear in consecutive order as a result of the sort. The time to identify the sets will be proportional to the number of boundary vertices.

A *topology-based heap on boundary vertices* is a heap represented by a balanced tree in which the values associated with the boundary vertices appear in the leaves, with values from any one boundary set being in consecutive leaves. A topology-based heap corresponding to the boundary sets of Fig. 3(a) is shown in Fig. 3(b). The sets of leaves that are circled correspond to the boundary sets. Note that the boundary vertices can be associated with the leaves in precisely the order generated by the previous lexicographic sort. Thus a topology-based heap can be set up in $O(n/\sqrt{r})$ time.

A *batched update* is an operation that updates in the topology-based heap the values associated with all vertices of a boundary set. In the search phase of our algorithm, whenever some vertex is closed, each region containing the vertex will have a batched update performed for each of its boundary sets. To perform a batched update, do the following. Modify the values at all leaves corresponding to vertices in the boundary set. Then proceed to modify the ancestors of these leaves level by level moving upward: first parents of the leaves, then grandparents, etc., until the value at the root is modified.

The number of nodes in the heap that are modified by a batched update is less than $2 \log n$ plus twice the size of the boundary set. This is established by the following reasoning. The number of interior nodes, all of whose leaf descendants correspond to vertices in the boundary set, is less than the size of the boundary set. The only other interior nodes that are examined have some leaf descendants that correspond to vertices in the boundary set, and some that do not. There are at most two such interior nodes per level.

THEOREM 2. *Let an n-vertex planar graph be divided into a suitable r-division. Using the associated topology-based heap, the main thrust of our algorithm will perform a set of batched update operations which cost* $O(n+(n/\sqrt{r})\log n)$.

Proof. Since there are at most $O(\sqrt{r})$ boundary vertices per region, and each boundary set is in at most three regions, each boundary set can have a batched update performed on it at most $O(\sqrt{r})$ times. Thus the total work involved in all batched updates for any one boundary set is proportional to $\sqrt{r}\log n$ plus $\sqrt{r}$ times the cardinality of the boundary set. Since there are $\Theta(n/r)$ boundary sets, the total cost of the first term over all boundary sets is $O((n/\sqrt{r})\log n)$. Since there are $O(n/\sqrt{r})$ boundary vertices, the total cost of the second term over all boundary sets is $O(n)$. □

We now describe how our strategy would work, if we are already given a suitable r-division of a graph, for a parameter r which we shall specify later. To perform the main thrust, we need, for each region, the shortest paths between every pair of boundary vertices. These paths can be found by performing, for each boundary vertex of a region, Dijkstra's algorithm within the region. Since there are $O(n/\sqrt{r})$ boundary vertices, and each run of Dijkstra's algorithm will use $O(r \log r)$ time, the time required to find these paths will be $O(n\sqrt{r}\log r)$.

Given this preprocessing, the main thrust of the algorithm follows our previous description, using the topology-based heap, and performing batched update operations

in it. At termination of the main thrust, shortest paths from the source to each boundary vertex will be known. Shortest distances to other vertices may be found by performing mop-up in each region separately. We can use Dijkstra's algorithm in each region, starting the search from the set of boundary vertices, labeled with the shortest distances to them. We choose $r = (\log n)/(\log\log n)$, so as to balance the times for the main thrust and the preprocessing.

LEMMA 3. *Let an n-vertex planar graph be divided into a suitable r-division, where* $r = (\log n)/(\log\log n)$. *A single source shortest path tree can be found in* $O(n\sqrt{\log n}\sqrt{\log\log n})$ *time.*

Proof. As claimed in Theorem 2, the time to perform the main thrust will be $O(n + (n/\sqrt{r})\log n)$. The mop-up will take $O(r \log r)$ time for each of $\Theta(n/r)$ regions, yielding $O(n \log r)$ time altogether. The result follows since $r = (\log n)/(\log\log n)$. □

5. A fast algorithm for the single source problem. In this section we give an algorithm for finding a single source shortest path tree in a planar graph in $O(n\sqrt{\log n})$ time. To achieve this bound, we must show how to find a suitable r-division in $o(n \log n)$ time. We must also improve on the performance of the algorithm in the last section, as described in Lemma 3. We first discuss the latter problem.

A time-critical part of the computation is in the preprocessing, in which for each region the shortest paths between every pair of boundary vertices are found. Instead of using Dijkstra's algorithm to perform these searches within regions, we shall use our search strategy. This will of course require that suitable divisions of each region be found, so that the search can be performed.

We now describe our algorithm. First find a suitable r_1-division of the graph, where $r_1 = \log n$. (We shall describe later how to do this in $o(n \log n)$ time.) Call each region in this division a *level* 1 *region.* Then for each level 1 region find a suitable r_2-division, where $r_2 = (\log\log n)^2$. Call each region in these divisions a *level* 2 *region.* When finding this division, start with each level 1 boundary vertex automatically being a level 2 boundary vertex. This will not cause more than $\Theta(n/\sqrt{r_2})$ boundary vertices of level 2 regions to be created. For each level 2 region, find the shortest paths between every pair of its boundary vertices. Dijkstra's algorithm should be used for this task, with the source being each boundary vertex of the region in turn. Then for each level 1 region, find the shortest paths between every pair of its boundary vertices. The main thrust of our search phase, described earlier, should be used. This concludes the preprocessing. Having found, for each level 1 region, the shortest distances between its boundary vertices, we then perform the search phase on the graph. The main thrust will yield a shortest path tree encoding the shortest paths to each level 1 boundary vertex in the graph. The mop-up phase can then be performed by labeling boundary vertices with the shortest distances to them, and then using Dijkstra's algorithm within each region.

We analyze the time for this algorithm, exclusive of the time to find the r_1-division. The time required to find an r_2-division of a region of size r_1 is $O(r_1 \log r_1)$. Summing over the $\Theta(n/r_1)$ regions gives a total time of $O(n \log\log n)$. Using Dijkstra's algorithm, the time to find a shortest path tree in a region of size at most r_2 is $O(r_2 \log r_2)$. Finding these shortest path trees for each of $O(n/\sqrt{r_2})$ level 2 boundary vertices will use $O(n \log\log n \log\log\log n)$ time. Using our main thrust, the time to find a shortest path tree in a region of size at most r_1 is $O(r_1 + (r_1/\sqrt{r_2})\log r_1)$, or $O(r_1)$. Note that no mop-up is necessary, since each level 1 boundary vertex was automatically designated a level 2 boundary vertex. Thus the total time for finding these shortest path trees for each of the $O(n/\sqrt{r_1})$ level 1 boundary vertices is $O(n\sqrt{r_1})$, or $O(n\sqrt{\log n})$.

By Theorem 2, the main thrust within the graph will take time $O(n\sqrt{\log n})$. The mop-up for each of $\Theta(n/r_1)$ level 1 regions will take $O(r_1 \log r_1)$ time per region, or $O(n \log\log n)$ time total. Thus the total time for everything except finding the r_1-division will be $O(n\sqrt{\log n})$.

We now show how to generate a suitable r-division quickly. Find a spanning tree of the graph using depth-first search. Generate connected sets of $\Theta(\sqrt{r})$ vertices in a bottom-up fashion, using a procedure FINDCLUSTERS from [F1]. Now shrink the graph on these sets, yielding a planar graph G_s with $\Theta(n/\sqrt{r})$ vertices. Apply the planar separator strategy from § 3 to graph G_s to yield $\Theta(n/r^{3/2})$ regions of cardinality $O(r)$. Expand G_s back to G. In G there will be $O(n/r)$ regions of size $O(\sqrt{r})$ resulting from boundary vertices in G_s, plus $\Theta(n/r^{3/2})$ regions of cardinality $O(r^{3/2})$ resulting from the interior vertices of G_s. Infer boundary vertices and slightly expanded regions that share these vertices. Apply our procedures from § 3 on the regions to generate a suitable r-division.

LEMMA 4. *A planar graph of n vertices may be divided into a suitable r-division in $O(n \log r + (n/\sqrt{r}) \log n)$ time.*

Proof. The connected sets and the graph G_s can be identified in $O(n)$ time. The time to generate a division of G_s will be $O(n(\log n)/\sqrt{r})$. The time to expand G_s back to G will be $O(n)$. The time to split regions of size $O(r^{3/2})$ will be $O(n \log r)$. □

THEOREM 3. *A single source shortest paths problem on an n-vertex planar graph with nonnegative edge costs can be solved in $O(n\sqrt{\log n})$ time.* □

Our single source shortest path algorithm can be used to speed up an algorithm [Hs1], [IS] for finding a maximum flow in an s-t planar network. The dominating term in the running time of this algorithm is the time to solve one single source problem. Substituting our single source shortest path algorithm yields the following.

COROLLARY 1. *A maximum flow in an s-t planar network can be found in $O(n\sqrt{\log n})$ time.* □

6. Decision trees for regions. As indicated in the Introduction, there are several problems that can be solved by solving a number of single source problems. For such a problem it is not necessary to balance the preprocessing time against the search time of one shortest path computation, since the preprocessing need be done only once, while the search will be done many times. In the next section, using the results of this section, we show how to realize $O(n)$ search time at a one-time expense of $O(n \log n)$ preprocessing time.

A part of the search will use decision trees, one for each of many small regions. For each of these regions, its decision tree will take as input the shortest distances from some source vertex in the graph to each of the boundary vertices of the region (plus some other information to be described subsequently). A leaf in the decision tree will identify all edges inside the region that are contained in a shortest path tree of the graph rooted at the source vertex. The height of the decision tree for any region, and thus the time to search it, will be proportional to the size of the region. The time to build the decision tree will be quite large in comparison to the size of the region. However, this technique will be applied in the next section to regions of very small size as a function of n.

We assume initially that the region is *nice*, i.e., that there is a planar embedding of the region such that all boundary vertices of the region are on the exterior face, and that a shortest path from the source to any boundary vertex is exterior to the region. In general, a region will not be nice. In the case when the region is not nice we show later in this section how to set up a decision tree that will identify a subset

of the boundary vertices that are on an exterior face in some embedding, so that shortest paths can still be computed.

The decision tree that we construct for nice regions will work for nice regions that are in a *prepared form.* Each prepared region will consist of a source vertex, a set of vertices V^b, and a set of other vertices V^u. For each vertex v_j in V^b there is an edge from the source to v_j, and for each vertex v_k in V^u there is no edge from the source to v_k. Every edge except those incident on the source has an edge cost supplied when the decision tree is built. For each vertex v_j in V^b, the cost of the edge from the source to v_j will be supplied as input when the decision tree algorithm is run. The input will satisfy the constraint that for each v_j in V^b, the edge from the source to v_j will represent a shortest path from the source to v_j that is exterior to the region.

We first discuss how to transform a nice region H into its prepared form. The boundary vertices of H will comprise the set V^b, and the interior vertices will comprise the set V^u. Introduce a source vertex and an edge from the source to each vertex v_j in V^b. The region from the lower right in the graph of Fig. 1 is shown in Fig. 4(a). The corresponding region in prepared form, along with costs on its edges, is shown in Fig. 4(b).

We next discuss how shortest distances and shortest paths from the source to every vertex in H can be found efficiently. Let $m = |V^u|$ and $b = |V^b|$. If $m = 0$, then shortest paths are already known from the source to each vertex in H, and no comparisons are needed in its decision tree. If $b = 1$, let $V^b = \{v_1\}$. A shortest distance

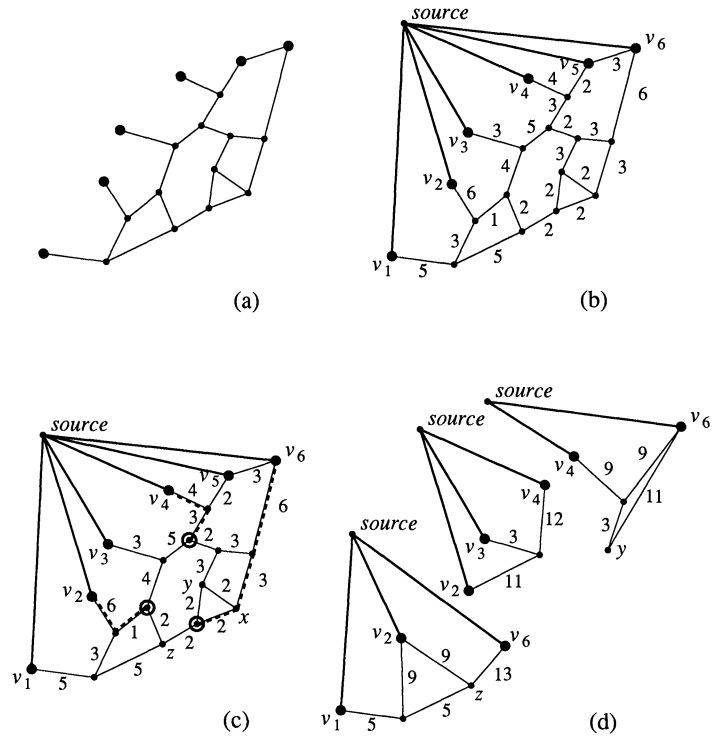


FIG. 4. (a) *A region from the division in Fig.* 1; (b) *Its corresponding prepared form, with edge costs shown*; (c) *Separator vertices circled, and shortest paths to them shown*; (d) *Prepared subregions induced by shortest paths to separator vertices.*

to each vertex x will be $d(source, x) = d(source, v_1) + d(v_1, x)$. A shortest path from the source to x must go from the source to v_1, and then follow a shortest path from v_1 to x. Since a shortest path from v_1 to x does not depend on the input to the decision tree, again no comparisons are needed in the decision tree for H.

If $b = 2$, let $V^b = \{v_1, v_2\}$. The shortest distance from the source to each vertex x will be the smaller of $d(source, v_1) + d(v_1, x)$ and $d(source, v_2) + d(v_2, x)$. Note that distances $d(v_1, x)$ and $d(v_2, x)$, and the shortest paths realizing these distances, do not depend on the inputs to the decision tree, and can be computed when the decision tree is built. Thus for each vertex x of V^u, only one comparison is needed in the decision tree to determine which of v_1 and v_2 is on the shortest path from the source to x. This means that m comparisons involving input values are sufficient to determine the edges in H contained in a shortest path tree rooted at the source.

If $m > 0$ and $b > 2$, region H can be handled as follows. By the planar separator theorem of [LT1], there is a separator V^s of size at most $2\sqrt{2}\sqrt{m}$ for the subgraph induced by V^u. Let $s = |V^s|$. For each vertex v_k in V^s, a shortest distance from the source to v_k can be found by finding the smallest value $d(source, v_j) + d(v_j, v_k)$ over all vertices v_j in V^b. We assume that a tie between two vertices v_j and $v_{j'}$ in V^b is broken by choosing v_j if $j < j'$. The shortest path from the source to v_k will consist of the edge from the source to the vertex v_j realizing the minimum, followed by the shortest path from v_j to v_k. Since the latter shortest paths do not depend on the input to the decision tree, only $(b-1)s$ comparisons involving input values are necessary to find shortest paths from the source to all vertices in V^s. These $(b-1)s$ comparisons would be performed in the top $(b-1)s$ levels of the decision tree for the region, and would determine the set of shortest paths to all vertices in V^s. The remaining levels can be determined as follows.

Any set of paths realizing shortest distances to vertices in V^s divides H into some number of subregions, say t. Note that any vertex x in V^u which is on a shortest path from the source to some v_k in V^s has a shortest path from the source that is a prefix of the shortest path to v_k. Let C be the set of vertices in V^u that are on a shortest path from the source to a vertex in V^s. Every maximal connected subset in $V^u - C$ will be taken to be the interior vertices V_i^u of a subregion H_i. Any edge between vertices in V_i^u will remain in subregion H_i.

The set V_i^b is determined as follows. Vertex v_j in V^b is included in V_i^b if either v_j is adjacent to some vertex y in V_i^u, or the shortest path to some vertex v_k in V^s contains both v_j and a vertex x which is adjacent to some vertex y in V_i^u. In the former case, there will be an edge included in H_i from v_j to y. In the latter case, for any such pair v_j and y, there will be an edge (v_j, y) in H_i, of cost $d(v_j, x) + c(x, y)$ which is minimum over all such x.

With respect to the region in Fig. 4(b), a separator set V^s for the subgraph induced by V^u is shown in Fig. 4(c), along with shortest paths to vertices in V^s. These paths partition the region into three nice subregions, which are shown in prepared form in Fig. 4(d). For instance, the edge (v_6, y) in Fig. 4(d) results from y being adjacent to vertex x, which is on the shortest path from v_6 to one of the separator vertices in Fig. 4(c). The cost 11 follows since $c(x, y) = 2$ and $d(v_6, x) = 6 + 3 = 9$. Similarly, edges (v_2, z) and (v_6, z) in Fig. 4(d) result from z being adjacent to separator vertices that are claimed by v_2 and v_6 in Fig. 4(c). Their costs of 9 and 13, resp., represent the costs $6 + 1 + 2$ and $6 + 3 + 2 + 2$ of the corresponding paths.

LEMMA 5. *Consider the dividing of a region H into subregions $\{H_i\}$ based on paths as above. Let $b_i = |V_i^b|$. Let $I = \{i \mid b_i > 2\}$. Then $\sum_{i \in I} (b_i - 2) \leqq b - 2$.*

Proof. By induction on $|I|$. The basis with $|I| = 1$ is immediate. For $|I| > 1$, we

assume that the lemma is true for any region H' with $|I'|<|I|$. We show how to split H into H' and H'', so that the induction hypothesis can be applied to each, and the bound established. Index the boundary vertices in H in order around the exterior face from 1 to b. Consider any i' in I.

Let v_j and $v_{j'}$ be in $V^b_{i'}$ such that $j<j'$ and there is no $v_{j''}$ in $V^b_{i'}$ with $j<j''<j'$. Then for any $V^b_{i''}$, either all vertices v_k in $V^b_{i''}$ have $j\leqq k\leqq j'$, or no vertices v_k in $V^b_{i''}$ have $j<k<j'$. Suppose $V^b_{i''}$ contained v_k and $v_{k'}$, where $j<k<j'$ and either $k'<j$ or $j'<k'$. There would be a path from v_j to $v_{j'}$ in $H_{i'}$, and a path from v_k to $v_{k'}$ in $H_{i''}$. Since all vertices in V^b are on the exterior face in H, the paths must cross, an impossibility.

Let i', $i''\in I$, and without loss of generality choose j and j' as above so that all vertices v_k in $V^b_{i''}$ have $j\leqq k\leqq j'$. Now partition the H_i so that $i\in I'$ if and only if H_i contains no vertex v_k in $V^b_{i'}$ with $j<k<j'$, and $i\in I''$ otherwise. This induces a dividing of H into H' and H'', with $H_{i'}$ within H', and $H_{i''}$ within H''. Thus I is partitioned into I' and I'', with each of smaller cardinality than I. The only boundary vertices shared by H' and H'' are v_j and $v_{j'}$, so that $b'+b''\leqq b+2$. Applying the induction hypothesis, $\sum_{i\in I'}(b_i-2)\leqq b'-2$, and $\sum_{i\in I''}(b_i-2)\leqq b''-2$. Thus $\sum_{i\in I}(b_i-2)\leqq (b'-2)+(b''-2)\leqq b-2$. □

We summarize the construction of the decision tree for a region H which has an embedding with all boundary vertices on an exterior face. If $m=0$ or $b=1$, then the decision tree consists of a leaf labelled with the shortest path tree in H. If $b=2$, then the decision tree is of height m, with a test at every level determining for each interior vertex x in turn whether the shortest path to x goes through v_1 or v_2. If $m>0$ and $b>2$, then first build a decision tree of height $(b-1)s$, which determines for each vertex v_k in V^s through which vertex in V^b a shortest path from the source passes. Each leaf in this tree represents a certain division of H into t subregions based on the shortest paths from the source to vertices on these paths. Each subregion H_i has a corresponding prepared form, and the decision tree for H_i can be found recursively. Such a decision tree for H_1 will replace the leaf for that particular partition. The decision tree for H_2 will replace each of the leaves of H_1, and so on for the rest of the H_i's resulting from that partition. Each leaf will be labelled with the shortest path tree that is the result of the comparisons on the path down to that leaf. When all partitions of H have been handled, the decision tree for H is complete.

From the above discussion, the height of the resulting decision tree, and thus the time to search it, can be described by the recurrence

$$T(0, b) = T(m, 1) = 0,$$

$$T(m, 2) = m,$$

$$T(m, b)\leqq \max_{s,t,m_i}\left\{s(b-1)+\sum_{i=1}^{t} T(m_i, b_i)\right\}$$

where $m_i\leqq 2m/3$ (by the planar separator algorithm) and $\sum_{i=1}^{t} m_i\leqq m$.

LEMMA 6. *For $b>2$, the solution to the above recurrence satisfies*

$$T(m, b)\leqq c\sqrt{m}\, b+m$$

where $c=6\sqrt{2}+4\sqrt{3}$.

Proof. By induction on m. The basis is for $m=0$. $T(0, b)=0$ clearly satisfies the above bound. The induction step is for $m>0$. If $b=1$ or $b=2$, then the bound is

clearly satisfied. If $b > 2$, then we have

$$T(m, b) \leqq \max_{s,t,m_i} \left\{ s(b-1) + \sum_{i=1}^{t} T(m_i, b_i) \right\}$$

$$= \max_{s,t,m_i} \left\{ s(b-1) + \sum_{i \in I} T(m_i, b_i) + \sum_{i \notin I} T(m_i, b_i) \right\}$$

$$\leqq \max_{s,t,m_i} \left\{ s(b-1) + \sum_{i \in I} (c\sqrt{m_i}\, b_i + m_i) + \sum_{i \notin I} m_i \right\}.$$

Use of the planar separator ensures that $m_i \leqq 2m/3$. From the manner in which the subregions are generated, $m \geqq \sum_{i=1}^{t} m_i$, and $\sum_{i \in I} (b_i - 2) \leqq b - 2$. Thus the sums are maximized if $|I| = 1$, with $m_i = 2m/3$ and $b_i = b$ for the one i in I. The term involving s is maximized when s is as large as possible, which can be at most $2\sqrt{2}\sqrt{m}$ by the separator theorem [LT1]. Thus

$$T(m, b) \leqq 2\sqrt{2}\sqrt{m}\,(b-1) + c\sqrt{2m/3}\,b + m$$

$$< (2\sqrt{2} + c\sqrt{2/3})\sqrt{m}\,b + m.$$

Taking c as above we have the claimed result. □

We now discuss the handling of a region for which there is no planar embedding with all boundary vertices on the exterior face. An example of such a region is region A in Fig. 2. We show how to build a decision tree which finds a subset of boundary vertices with the desired property, while still allowing for the correct computation of shortest paths. We assume that a shortest path tree from the source to each of the boundary vertices in the graph will be known, and that the boundary vertices in this tree will have been given preorder and postorder numbers.

As input to the tree we have shortest distances from the source vertex to each boundary vertex of the region, along with the preorder and postorder numbers of each boundary vertex of the region with respect to the preorder and postorder numbering of all boundary vertices in a shortest path tree in the graph rooted at the source. It is well known that a vertex v_j is an ancestor of v_k in a tree if and only if *preorder* $(v_j) \leqq$ *preorder* (v_k) and *postorder* $(v_j) \geqq$ *postorder* (v_k). Thus it is possible to determine the structure of the shortest path tree with respect to the b boundary vertices of H by comparing each preorder number with every other preorder number, and similarly with postorder numbers. This will take $b(b-1)$ comparisons. Precisely those boundary vertices in H with no ancestor that is a boundary vertex in H will be in the subset. All others will be viewed as interior.

To preserve shortest paths, certain pseudoedges must be introduced. If v_k is a boundary vertex not included in the subset, with nearest boundary vertex ancestor v_j, then pseudoedge (v_j, v_k) is introduced if and only if $d(source, v_k) - d(source, v_j)$ is less than the distance within H from v_j to v_k. Such an edge is of cost $d(source, v_k) - d(source, v_j)$, and represents the shortest path from v_j to v_k. At most $b-2$ comparisons are needed to determine which pseudoedges must be introduced.

Thus we build a decision tree of height at most $b(b-1)+b-2$. On each pair of the first $b(b-1)$ levels a different pair v_j, v_k is tested to determine whether v_j is an ancestors of v_k, or vice versa. On the following at most $b-2$ levels it is determined which new edges are introduced. Associated with each leaf of the decision tree is an induced nice region, having as boundary vertices a subset of boundary vertices from the original region, and including the additional edges as described above.

Given this decision tree, we can extend it to a decision tree that finds the portion of shortest paths from a source going through any region. In place of each leaf in the above tree, put a decision tree for the corresponding induced nice region. The resulting decision tree has the top at most $b(b-1)+b-2$ levels determining the induced region, and the remaining levels finding the shortest paths in the induced region. Leaves of the resulting decision tree will not contain pseudoedges.

THEOREM 4. *Let H be a region in a suitable r-division of a planar graph G. The portion in H of a shortest path tree for the source in G can be found using a decision tree of height $O(r)$, which takes as input the shortest distances from the source to each of the boundary vertices in H, plus preorder and postorder numbers of each boundary vertex. The decision tree can be built in $O(r^2(\log r)2^{cr})$ time, and will use $O(r2^{cr})$ space, for some constant c.*

Proof. The region will have $O(\sqrt{r})$ boundary vertices, and at most r interior vertices. Thus the prepared form for region H will have b be $O(\sqrt{r})$ and $m \leqq r$. By Lemma 6 and the preceding discussion, the height of the decision tree for region H will be $O(r)$. Determining shortest paths within the region, encoded as r shortest path trees, will use $O(r^2 \log r)$ time. The amount of work to determine the information generated by a set of prescribed tests will be $O(r^2)$. If $b=2$, the result for each set of outcomes of the m comparisons can be translated into a shortest path tree in $O(r)$ time per vertex, or $O(r^2)$ time total. If $m>0$ and $b>2$, the result for each set of outcomes of the $(b-1)s$ comparisons can be translated into a dividing of a region into subregions in $O(b+r)$ time per separator plus $O(r)$ time per vertex, or $O(r^2)$ time in total. Since the decision tree is a binary tree, and of height at most cr for some constant c, the number of nodes in the tree is $O(2^{cr})$. Each of $O(2^{cr})$ leaves will use $O(r)$ space. □

7. A very fast search strategy. We use the decision tree technique from the previous section, plus an additional technique, to generate a search strategy that takes $O(n)$ time. In § 5, by choosing $r_1=(\log n)^2$, we could have realized $O(n \log\log n)$ search time, at the expense of $O(n \log n)$ preprocessing time. Furthermore, we could have reduced the search time to $O(n \log\log\log n)$ if we had done the following. Instead of using Dijkstra's algorithm on level 1 regions for the mop-up, we could have run our main thrust on level 1 regions and then used Dijkstra's algorithm for the mop-up within level 2 regions. Assuming that $r_1=(\log n)^2$ and $r_2=(\log\log n)^2$, this would have yielded a search time of $O(n \log\log\log n)$. This idea can be carried to its limit by using $\log^* n$ levels. At $O(n)$ main thrust time per level, this would give $O(n \log^* n)$ time total.

We show how to do better using the decision trees. The preprocessing is as follows. First find a suitable r_1-division of the graph, where $r_1=(\log n)^2$. For each level 1 region find a suitable r_2-division where $r_2=(\log\log n)^2$. When finding this division, start with each level 1 boundary vertex automatically being a level 2 boundary vertex. For each level 2 region find a suitable r_3-division where $r_3=(\log\log\log n)^2$. When finding these divisions, start with each level 2 boundary vertex automatically being a level 3 boundary vertex. For each level 3 region, build a decision tree. Then for each level 3 region, find the shortest paths between every pair of level 3 boundary vertices. This should be done by using the decision tree to solve a single source problem for each boundary vertex. For each level 2 region, find the shortest paths between every pair of level 2 boundary vertices. The main thrust is used for this, using each level 2 boundary vertex in turn as the source. Finally for each level 1 region, find the shortest paths between every pair of level 1 boundary vertices, using the main thrust from each level 1 boundary vertex.

The search of the graph is then the following. To perform a single source computation, first find the shortest paths from the source to the boundary vertices of the level 1 region containing the source. This can be done using Dijkstra's algorithm. Given the shortest distances to boundary vertices in the level 1 region containing the source, perform the main thrust on level 1 boundary vertices. Given these results, perform the main thrust within each level 1 region on level 2 boundary vertices. Then perform the main thrust in each level 2 region on level 3 boundary vertices. Finally, complete the problem in each level 3 region by using the decision tree.

THEOREM 5. *An n-vertex planar graph can be preprocessed in $O(n \log n)$ time so that each subsequent shortest path computation can be performed in $O(n)$ time.*

Proof. The r_1-division can be found in $O(n \log n)$ time. The r_2- and r_3-divisions can be seen to require no more time than this. Building decision trees for all of the $\Theta(n/r_3)$ level 3 regions, will take time $O(nr_3(\log r_3)2^{cr_3})$. With $r_3 = (\log \log \log n)^2$, the time will be $O(n \log n)$. Finding the shortest paths between boundary vertices at levels 3, 2 and 1 can be seen to take $O(n \log n)$ time by arguments similar to previous ones. Thus the preprocessing time is $O(n \log n)$. For the time to solve the next single source problem, we have the following. The initial shortest path computation within the level 1 region containing the source will use $O(r_1 \log r_1)$ time, which is $O((\log n)^2 \log \log n)$. The searches on boundary vertices at levels 1, 2 and 3 can be seen to take $O(n)$ time. The search within a level 3 region, using the decision tree for the region, will be $O(r_3)$, which yields $O(n)$ total for all level 3 regions. □

The best previous algorithm for all pairs shortest paths in a planar graph is presented in Johnson [J]: For the case of nonnegative edge costs, Dijkstra's algorithm with a heap implementation may be run from each vertex, yielding $O(n^2 \log n)$ time. If there are edges of negative cost, but no negative cycles, and the edges are directed, then the best single source algorithm for planar graphs [LRT] uses $O(n^{3/2})$ time. However, such an algorithm need be used only once in an all pairs computation. Dijkstra's algorithm will perform correctly if vertices are closed in order of $d(v) + h(v)$, where $h(v)$ is the shortest distance from v to a specific vertex u [EK], [J], [Ne], [Ni]. Using our algorithm in place of Dijkstra's, we achieve the following.

THEOREM 6. *The all pairs shortest paths problem on an n-vertex planar graph with either undirected edges and no negative costs or directed edges and negative costs but no negative cycles can be solved in $O(n^2)$ time.* □

8. Minimum cut and maximum flow. Our shortest path procedure may be used to speed up Reif's algorithm for finding a minimum s-t cut in an undirected planar network, which runs in $O(n(\log n)^2)$ time [R]. The idea behind the algorithm is that a minimum cut corresponds in the planar dual to a shortest cycle that separates s and t, and that this cycle can be found efficiently by a divide-and-conquer strategy. Reif's algorithm proceeds in the following manner. The graph G is embedded in the plane, and the dual graph $D(G)$ is determined. Each edge in $D(G)$ has cost equal to the capacity of its corresponding edge in G. Let P_{st} be a minimum cost path in $D(G)$ from a vertex representing a face in G that borders s to a vertex representing a face in G that borders t.

If P_{st} contains one vertex, v, then s and t are on the same face, f_v, in G. The minimum s-t cut in G can be found by using a shortest path algorithm in an augmented dual $D'(G)$ generated as follows. Vertex v is replaced by new vertices v' and v''. Each edge (v, w) incident on v is replaced by (v', w) if the corresponding edge in G is in that portion of the boundary of face f_v that starts at s and proceeds clockwise to t, and by (v'', w) otherwise. The shortest path P_v between v' and v'' corresponds to the

minimum cost cycle in $D(G)$ containing v that separates face s from face t. (This portion of Reif's algorithm corresponds to the $O(n \log n)$ time algorithms of Itai and Shiloach (IS) and Hassin [Hs1] for finding a maximum s-t flow in a planar graph when both s and t are on the same face.)

If P_{st} contains more than one vertex, Reif's algorithm does the following. The augmented dual $D'(G)$ is generated by splitting apart the path P_{st} into two paths P'_{st} and P''_{st}, duplicating the vertices and edges of P_{st}, and replacing the edges incident on P_{st} as follows. Any edge (v, w) that is not in P_{st} but is incident on vertex v in P_{st} is replaced by (v', w) if (v, w) is to the right of P_{st} in $D(G)$, and by (v'', w) if (v, w) is to the left of P_{st} in $D(G)$. Once $D'(G)$ is generated, it is searched as follows. Let v be a midpoint of the original path P_{st}. Let v' and v'' be the corresponding vertices in $D'(G)$. A shortest path computation in $D'(G)$ is performed using v' as the source and v'' as the destination. The shortest path P_v between v' and v'' corresponds to the minimum cost cycle in $D(G)$ containing v that separates face s from face t. The network $D'(G)$ is split into two subnetworks along P_v, including vertices and edges from P_v in both subnetworks. Any maximal path of at least two bridges is replaced by a new edge, of cost the sum of costs in the maximal path. Then Reif's algorithm recurses on each half. Among the cycles so identified, the cycle of minimum cost will correspond to a minimum s-t cut. The work performed across each level of recursion is dominated by the shortest path computations, which are $O(n \log n)$ time if Dijkstra's algorithm is used. The time to handle all levels of recursion is $O(n(\log n)^2)$, since there are $O(\log n)$ levels of recursion.

In place of Dijkstra's shortest path algorithm, we will use our own algorithm. We perform preprocessing similar to that discussed in § 7 on the augmented dual $D'(G)$. First find a suitable r_1-division of the graph, where $r_1 = (\log n)^2$, and then a suitable r_2-division of each level 1 region, where $r_2 = (\log \log n)^2$. When finding this division, start with each level 1 boundary vertex automatically being a level 2 boundary vertex. For each level 2 region, find the shortest paths between every pair of level 2 boundary vertices, using Dijkstra's algorithm. Finally for each level 1 region, find the shortest paths between every pair of level 1 boundary vertices, using the main thrust from each level 1 boundary vertex.

Now perform Reif's minimum cut algorithm, but whenever a source-destination shortest path computation is required, use the following. Find the shortest paths from the source to the boundary vertices of the level 2 region containing the source. Dijkstra's algorithm can be used for this purpose. Then find the shortest paths to the boundary vertices of the level 1 region containing the source. Our main thrust can be used within this level 1 region. Now perform our main thrust on level 1 boundary vertices. This will in particular give the shortest distances to the boundary vertices of the level 1 region containing the destination. Perform the main thrust on level 2 boundary vertices within this region. This will give the shortest distances to the boundary vertices of the level 2 region containing the destination. Finally perform a shortest path computation within this region, using Dijkstra's algorithm. This will then give the shortest path from the source to the destination.

Whenever a shortest path corresponding to a cycle is identified, a number of regions in general will need to be split. However it is not hard to do this if the boundaries are maintained in a convenient form. The regions can be split so that an efficient main thrust can still be carried out. For each region, maintain a list of its boundary vertices, ordered by boundary set, and within boundary set ordered by the order along the boundary. For each boundary vertex in the region, shortest distance information to all other boundary vertices is kept in this same order. When a region is split, the

vertices are partitioned into two sets, each of which can be described by a sequence of index pointers into the original list. If a subnetwork contains a sequence of split regions with just two boundary vertices, then this sequence should be replaced by a pseudo split region with just two boundary vertices. This operation is analogous to replacing a sequence of bridges in Reif's algorithm.

During a search, a split region will be handled in the following fashion. When a boundary vertex in one set of the partition is closed, only vertices in the same set of the partition are updated. This means that for all searches at any level of the recursion in the modification of Reif's algorithm, no more work is done in updating the heap than at the top level of the recursion. This follows since the same number of boundary vertices are closed as in the top level of recursion, and for each boundary vertex that is closed, the updating is at worst no more expensive than at the top level.

THEOREM 7. *A minimum s-t cut in an undirected planar network can be identified in* $O(n \log n)$ *time.*

Proof. The time required is the following. The preprocessing time is $O(n \log n)$. The time to search from a source to the boundary vertices of the level 2 region containing it is $O(r_2 \log r_2)$, which is $O((\log \log n)^2 \log \log \log n)$. The time to search from the boundary vertices of the level 2 region containing the destination to the destination itself will be the same. Since there are $O(n)$ source-destination computations, these activities will use in total $O(n(\log \log n)^2 \log \log \log n)$ time. The remaining activities in the source-destination computations are accounted for by considering all such computations on one level of recursion. The main thrusts on any one such level of recursion will take $O(n)$ time. Note that the time to split regions corresponding to the source-destination paths found will total no more than proportional to the number of edges in the graph, which is $O(n)$. Since there are $O(\log n)$ levels, the total time for all source-destination computations is $O(n \log n)$. □

Hassin and Johnson [HJ] have shown how to use Reif's construction to find a maximum flow in a planar undirected network in $O(n(\log n)^2)$ time. The output that they need from Reif's algorithm is the minimum cost *s-t* cut cycle in $D(G)$ that contains v, for each vertex v in P_{st}. Aside from the time to find a minimum *s-t* cut, the Hassin and Johnson algorithm uses $O(n \log n)$ time. Since our minimum cut algorithm produces the same information about cut cycles as Reif's, ours may be used in its place.

COROLLARY 2. *A maximum s-t flow in an undirected planar network can be found in* $O(n \log n)$ *time.* □

9. Multicommodity flows. In this section we discuss the multicommodity flow problem in a planar graph in which all sources and sinks are on the same face. Algorithms for determining feasibility and finding feasible flows in such graphs have been presented in [Hs2], [MNS]. The feasibility test for multicommodity flows given in [MNS] uses $O(\min\{n^2 \log^* n, kn\sqrt{\log n}\})$ time, where k is the number of source-sink pairs. A flow construction algorithm is also presented in [MNS], which runs in $O(kn + n^2\sqrt{\log n})$ time. We show how to adapt our techniques to yield a faster feasibility test.

Let B be the set of edges bounding the face f that contains all the sources and sinks as vertices. Let b be the number of vertices of this face that serve as sinks and/or sources. We assume that $b \leqq 2k$, since if $b > 2k$, then additional edges of capacity 0 can be added to the graph to yield $b \leqq 2k$. As discussed in [Hs2], [MNS], these edges will connect consecutive source and/or sink vertices on the boundary of face f. Thus the only vertices on the boundary of face f in the modified graph will be sources

and/or sinks, implying that the number of edges on the boundary of this face will be at most $2k$. We assume that the input graph has already been so modified, and quote our results in these terms. Also, $k \leqq b(b-1)/2$, since there are at most this many distinct pairs from among the vertices bounding f.

The feasibility can be tested as follows. Let e and e' be two edges in B. Let $C(e, e')$ be the set of cuts that contain as the only edges from B the edges e and e'. For a cut X, let $c(X)$ be the total capacity of X, and $d(X)$ the total demand of X. Let $m(X) = c(X) - d(X)$ be the *margin* of cut X. Define $m(e, e')$ as the minimum $m(X)$ such that X is in $C(e, e')$. From results in [OS], it is shown in [Hs2], [MNS] that there is a feasible k-commodity flow if and only if $m(e, e') > 0$ for every pair e, e' in B.

In [Hs2], [MNS], the values $m(e, e')$ are determined in the following way. For a fixed e in B and all e' in B, $d(e, e')$ can be determined in $O(k+b)$ time. By traversing the edges of B in order starting at e, the values of $d(e, e')$ for all pairs (e, e') can be found in $O(k+b^2)$ time. To compute the values of $c(e, e')$, a dual graph $D(G)$ is constructed, with costs on the dual edges equal to capacities of the corresponding original edges. Let $c(e, e')$ be the minimum value of $c(X)$ for X in $C(e, e')$. Then $c(e, e')$ is equal to the sum of the capacities of e and e' plus the length of a shortest nontrivial path joining e and e' in $D(G)$. Using Dijkstra's algorithm, all values $c(e, e')$ can be found in $O(k + bn \log n)$ time. In [MNS], it is noted that our single source and all pairs algorithms (presented in [F2]) can be used to reduce the time to $O(\min\{n^2 \log^* n, bn\sqrt{\log n}\})$.

However we can speed up the computation of the $c(e, e')$ values more by using the shortest path methods described in §§ 6 and 7. We use our approach of one-time preprocessing and many-time search, but tailor it to the value of b. If $b \leqq \log n$, choose $r_1 = b \log n$. If $b > \log n$, choose $r_1 = (\log n)^2$, as before.

THEOREM 8. *The testing for feasibility of a k-commodity flow in an n-vertex planar graph in which all sources and sinks are on one face, bounded by b edges, $b \leqq 2k$, can be performed in $O(bn + n\sqrt{b \log n})$ time.*

Proof. First consider the case when $b \leqq \log n$, and $r_1 = b \log n$. The preprocessing time will be dominated by the time to find shortest path trees for each level 1 boundary vertex, which in total will be $O(n\sqrt{r_1})$, or $O(n\sqrt{b \log n})$. The b iterations of search will each be dominated by the main thrust at level 1, which is $O(b(n + (n/\sqrt{r_1}) \log n))$ by Theorem 2, or $O(n\sqrt{b \log n})$.

Next consider the case when $b > \log n$, and $r_1 = (\log n)^2$. By Theorem 5, the preprocessing will be $O(n \log n)$, which is $O(bn)$. The b iterations of search and post-processing will use $O(bn)$ time.

Finally, the $O(k+b^2)$ time to determine all $d(e, e')$ values is $O(b^2)$ by previous discussion, and the latter is $O(bn)$. □

Acknowledgments. The author thanks Takao Nishizeki, Knut Moeller, Carsten Vogt, Peter Klosterberg and an anonymous referee for their careful reading of the manuscript and many helpful suggestions.

REFERENCES

[AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[DP] N. DEO AND C. PANG, *Shortest-path algorithms: taxonomy and annotation*, Networks, 14 (1984), pp. 275-323.

[D] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269-271.

[EK] J. EDMONDS AND R. M. KARP, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. Assoc. Comput. Mach., 19 (1972), pp. 248–264.

[F1] G. N. FREDERICKSON, *Data structures for on-line updating of minimum spanning trees, with applications*, this Journal, 14 (1985), pp. 781–798.

[F2] ———, *Shortest path problems in planar graphs*, Proc. 24th IEEE Symp. on Foundations of Computer Science, Tucson, AZ, November 1983, pp. 242–247.

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

[Hs1] R. HASSIN, *Maximum flow in (s, t) planar networks*, Inform. Process. Lett., 13 (1981), p. 107.

[Hs2] ———, *On multicommodity flows in planar graphs*, Networks, 14 (1984), pp. 225–235.

[HJ] R. HASSIN AND D. B. JOHNSON, *An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks*, this Journal, 14 (1985), pp. 612–624.

[IS] A. ITAI AND Y. SHILOACH, *Maximum flow in planar networks*, this Journal, 8 (1979), pp. 135–150.

[J] D. B. JOHNSON, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1–13.

[LRT] R. J. LIPTON, D. J. ROSE AND R. E. TARJAN, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 177–189.

[LT1] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[LT2] ———, *Applications of a planar separator theorem*, this Journal, 9 (1980), pp. 615–627.

[MNS] K. MATSUMOTO, T. NISHIZEKI AND N. SAITO, *An efficient algorithm for finding multicommodity flows in planar networks*, this Journal, 14 (1985), pp. 289–302.

[Ne] G. L. NEMHAUSER, *A generalized permanent label setting algorithm for the shortest path between specified nodes*, J. Math. Anal. Appl., 38 (1972), pp. 328–334.

[Ni] N. J. NILSSON, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[OS] H. OKAMURA AND P. D. SEYMOUR, *Multicommodity flows in planar graphs*, J. Combin. Theory Ser. B, 31 (1981), pp. 75–81.

[R] J. H. REIF, *Minimum $s-t$ cut of a planar undirected network in $O(n \log^2(n))$ time*, this Journal, 12 (1983), pp. 71–81.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987

© 1987 Society for Industrial and Applied Mathematics
004

AN ALL PAIRS SHORTEST PATH ALGORITHM WITH EXPECTED TIME $O(n^2 \log n)$*

ALISTAIR MOFFAT† AND TADAO TAKAOKA‡

Abstract. An algorithm is described that solves the all pairs shortest path problem for a nonnegatively weighted directed graph of n vertices in average time $O(n^2 \log n)$. The algorithm is a blend of two previous shortest path algorithms, those of Dantzig [Management Sci., 6 (1960), pp. 187–190] and Spira [SIAM J. Comput., 2 (1973), pp. 28–32]. Bloniarz [SIAM J. Comput., 12 (1983), pp. 588–600] categorised a class of random graphs called endpoint independent graphs; the new algorithm executes in the stated time on endpoint independent graphs and represents an asymptotic improvement over the $O(n^2 \log n \log^* n)$ algorithm given by Bloniarz for this class of random graphs.

Key words. graph, algorithm, shortest path, computational complexity, expected time, average time, heap

AMS(MOS) subject classifications. 68Q25, 68R10, 05C38, 05C80

1. Introduction. We consider the problem of finding the shortest distances between all pairs of vertices in a nonnegatively weighted complete graph. The best upper bound on the complexity of this problem is due to Fredman [10], who gave an $O(n^3(\log\log n/\log n)^{1/3})$ time algorithm, improving slightly upon the $O(n^3)$ requirement of the classical algorithms of Dijkstra [6] and Floyd [8]. Here the average running time needed to compute all shortest paths is considered. Under the assumption that the edge weights of the graph are independently drawn from any single but arbitrary distribution Spira [14] gave an algorithm with an expected running time of $O(n^2 \log^2 n)$. Spira's algorithm has subsequently been refined ([4], [2]) and improved ([15], [3]), with the best general implementation being the $O(n^2 \log n \log^* n)$ average time algorithm of Bloniarz.

Here an earlier $O(n^3)$ time algorithm due to Dantzig [5] is re-examined and is used jointly with Spira's method as the basis for a new algorithm. Careful blending of techniques leads to an implementation which is faster than both, and on an endpoint independent graph the expected running time of the new algorithm is shown to be $O(n^2 \log n)$, an improved bound on the complexity of the problem. For random graphs with certain restricted edge cost assignments Frieze and Grimmett [11] showed that $O(n^2 \log n)$ expected time is sufficient, using the idea of heap cleaning introduced by Fredman [9]. In [13] the authors extended Fredman's work to show that $O(n^2 \log n)$ expected time was also sufficient for the more general case of endpoint independent graphs; and here the same result is achieved based on the original paradigm employed by Dantzig and Spira.

The paper is organised as follows. The remainder of this introductory section gives a definition of the problem to be solved; introduces some useful concepts and terminology; and gives some results for the data structure that will be used. Section 2 describes the algorithms of Dantzig and Spira, setting the scene for the new hybrid algorithm. In § 3 it is shown that the definition of an endpoint independent graph is strong enough to imply certain randomness characteristics; § 4 describes a new

* Received by the editors February 12, 1986; accepted for publication (in revised form) January 5, 1987.

† Department of Computer Science, University of Canterbury, Christchurch 1, New Zealand. This research was carried out in part while this author was a Visiting Researcher in the Department of Information Science, Ibaraki University, Japan. Present address, Department of Computer Science, The University of Melbourne, Parkville, Victoria 3052, Australia.

‡ Department of Information Science, Ibaraki University, Hitachi, Japan 316.

algorithm, discusses the decisions and observations that led to it, and shows that these randomness characteristics are sufficient to imply an $O(n^2 \log n)$ average running time.

The reader is assumed to be familiar with standard graph terminology [1]. It is supposed that $G=(V, E)$ is a complete directed graph of n vertices and that $C: E \to \mathbf{R}^+ \cup \{\infty\}$ is a cost function on the edges of G. The cost of a path in G is the sum of the costs of the edges comprising the path; a *shortest path* between two vertices is a path of minimal cost. The *all pairs shortest path problem* requires the tabulation of the function L, where for two vertices i, j in V, $L(i, j)$ is the cost of a shortest path from i to j. The *single source problem* requires the tabulation of the values $L(s, v)$ for some fixed source s and v ranging over all other vertices in the graph. When the cost function C is restricted to be nonnegative the function L is defined for all i, j in V, and it is this case that is considered. These algorithms are not applicable to graphs with negative cost edges.

All unspecified logarithms will be binary; when natural logarithms are required the function ln will be used. The function $\log^* n$ is defined to be the minimum integer i such that $\ln \cdots \ln n < 1$, where the logarithm is taken i times.

The following lemmas are basic to the analyses that follow.

LEMMA 1. *Suppose that in a sequence of independent trials the probability of success at each trial is p. Then the expected number of trials until the first success is* $1/p$.

Proof. From the properties of the geometric distribution [7]. □

LEMMA 2. *Suppose that in a sequence of independent trials the probability of each of n mutually exclusive events is* $1/n$. *Then the expected number of trials until all n events have occurred at least once is approximately* $n \ln n$.

Proof. From Lemma 1 and the results given in [12, p. 74]. □

The new algorithm, like that of Spira, uses a priority queue. The properties required are that on a queue of n elements:

(i) The element of least weight can be identified and deleted in $O(\log n)$ time;

(ii) A new element of arbitrary weight can be inserted in $O(\log n)$ time; and that

(iii) If, with probability p, each element in the queue independently and simultaneously has its weight increased in a single operation, the queue can be rebuilt in $O(pn + \log n)$ expected time.

To show that this combination of properties is possible, consider the humble heap. A *heap* of n elements is an implicit binary tree stored in an array, with the element at position i having sons in positions $2i$ and $2i+1$. The elements are stored partially ordered, so that no element has a weight less than its father. The least weight element in the data structure is thus stored at the root. That properties (i) and (ii) hold for heaps is well known [1]. The following lemma demonstrates property (iii).

LEMMA 3. *Suppose that with probability p each element in a heap of n elements independently and simultaneously has its weight increased in a single operation. Then the partial ordering can be restored in* $O(pn + \log n)$ *expected time.*

Proof. The rebuilding effort required after an increase in weight of a single element is proportional to the number of levels below that position in the heap. The element in position i has $\lfloor \log n/i \rfloor$ levels below, and will be altered with probability p. The elements to be adjusted should be batched and distributed according to their level in the heap, taking $O(pn + \log n)$ time; then bottom up processing of the adjustments will give a total effort bounded in the expected sense by

$$O(pn + \log n) + \sum_{i=1,n} p(\lfloor \log n/i \rfloor + 1)$$

$$\leqq O(pn + \log n) + p(n \log n - \log n!) + pn$$

$$\leqq O(pn+\log n)+p(n\log n-(n\log n-n\log e+(\log n)/2+O(1)))+pn$$

[12, p. 111]

$$\leqq O(pn+\log n)+pn(1+\log e)$$

$$= O(pn+\log n).$$ □

2. The algorithms of Dantzig and Spira. In Dantzig's method [5] an all pairs solution is found by first sorting all of the edge lists of the graph into ascending cost order, and then solving n single source problems. For the remainder of this paper all discussion will concern the solving of one single source problem from a vertex s, and it will be assumed throughout that the edge lists are available in sorted order.

Dantzig's algorithm operates as follows. Vertex s is first assigned a shortest path cost of zero and made the only member of a set S of *labelled* vertices for which the shortest path costs are known. Then, under the constraint that members c of S are "closer" to s than nonmembers u, that is, $L(s, c) \leqq L(s, u)$, the set S is expanded until all vertices have been labelled. To make the expansion of S computationally easy, information is maintained about paths from vertices already in S to vertices still outside S, and for each vertex c in S Dantzig maintains a *candidate* edge (c, t). A candidate is defined to be *useful* if the endpoint t is outside the current S, and *useless* otherwise. The *strong rule* upon which Dantzig's algorithm is based requires that all candidates be useful; to meet this requirement the candidate for a vertex c is selected by scanning the sorted list of edges out of c in increasing cost order until a useful edge (c, t) is found. This strategy means that t is the closest (by a single edge) vertex to c that is not yet in S. Shortest path costs for labelled vertices are recorded in a vector D, so that for c in S, $D[c] = L(s, c)$, and the path cost via vertex c to candidate t is given by the candidate *weight* $D[c]+C(c, t)$, where $C(c, t)$ is the cost of the edge from c to t.

Vertex t might also be the candidate of other already labelled vertices v; each time it is a candidate there will be some weight $D[v]+C(v, t)$ associated with its candidacy. There will in total be $|S|$ candidates, with endpoints scattered amongst the $n-|S|$ unlabelled vertices.

At each stage of the algorithm the endpoint of the least weight candidate is added to S. If c_0 is a vertex such that the candidate weight $D[c_0]+C(c_0, t_0)$ is minimum over all labelled vertices c, then t_0 can be included in S and given a shortest path cost $D[t_0]$ of $D[c_0]+C(c_0, t_0)$. Then an onward candidate for t_0 is added to the set of candidates, the candidates for vertices that have become useless are revised (including that of c_0), and the process repeats, to stop when $|S| = n$. Figure 1 shows some intermediate stage in this expansion of S.

To correctly initialise this process the source s is made the only member of S, $D[s]$ is set to zero, and the candidate for s is the shortest edge out of s. Dantzig's algorithm for solving a single source problem is given as Algorithm 1.

ALGORITHM 1. DANTZIG'S ALGORITHM.

```
procedure Single_Source (s);
  S := {s}; D[s] := 0;
  initialise the set of candidates to {(s, t)}, where (s, t) is the
    shortest edge from s;
  Dantzig_expand_S (n)
end {Single_Source};
```

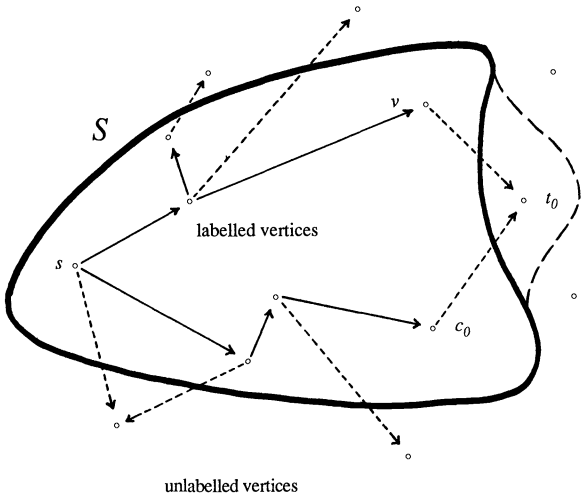


FIG. 1. *Expanding the solution set S.*

```
procedure Dantzig_expand_S (limit);
  while |S| < limit do begin
    let (c0, t0) be the candidate of least weight, where the weight
      of (c, t) is given by D[c] + C(c, t);
    S := S + {t0};
    D[t0] := D[c0] + C(c0, t0);
    if |S| = limit then break;
    add to the set of candidates the shortest useful edge from t0;
    for each useless candidate (v, t0) do
      replace (v, t0) in the set of candidates by the next shortest
        useful edge from v
  end
end {Dantzig_expand_S}
```

Bloniarz [3] states and attributes to Dantzig [5] the following observation:

LEMMA 4. *Suppose candidate* (c_0, t_0) *is such that*

$$D[c_0] + C(c_0, t_0) = \text{minimum}\{D[c] + C(c, t): (c, t) \text{ is a candidate}\}.$$

Then $L(s, t_0) = D[c_0] + C(c_0, t_0)$. □

Thus, if c_0 satisfies the requirements of Lemma 4, vertex t_0 may be added to S without violating the membership conditions of S. Dantzig's algorithm repeatedly finds the minimum weight candidate, and under the control of Lemma 4 adds the candidate to S.

The analysis of Algorithm 1 is straightforward. When $|S| = j$, $O(j)$ effort is required for a simple search in an array of candidates to find that of minimum cost, and once the minimum cost candidate is labelled, $O(j)$ effort is sufficient to check the other candidates to decide whether or not they remain useful. The total time for this is $O(n^2)$. The other major component of the running time is the effort spent scanning edge lists looking for useful edges; in the course of the algorithm each edge of the graph will be examined no more than once, which is again $O(n^2)$. The time for a single source problem is thus $O(n^2)$.

Spira independently gave an algorithm quite similar to that of Dantzig. The major difference between the two is that Spira relaxed the requirement that all candidates

be useful. Use of a *weak* candidacy rule—that all candidate edges (c, t) be such that $C(c, t) \leqq C(c, u)$ for all unlabelled vertices u—meant that the expensive scanning of adjacency lists could be reduced. Spira also suggested that the set of candidates should be implemented as a binary tournament tree, so that successive minimal weight candidates could be identified in $O(\log n)$ time each. The weakened candidacy rule means that the minimal cost candidate will no longer necessarily be useful, and so the while loop of the expand_S procedure will probably iterate more than n times in total. Candidates are examined and replaced until a useful candidate is drawn at the root of the heap; only then can S expand.

ALGORITHM 2. SPIRA'S ALGORITHM.

```
procedure Spira_expand_S (limit);
  while |S| < limit do begin
    let (c0, t0) be the candidate of least weight;
    if t0 is not in S then begin
      S := S + {t0};
      D[t0] := D[t0] + C(c0, t0);
      if |S| = limit then break;
      add to the set of candidates the shortest edge from t0
    end;
    replace (c0, t0) in the set of candidates by the next shortest
      edge from c0;
  end
end {Spira_expand_S}
```

The calling program for this procedure is almost identical to that given in Algorithm 1, and is omitted.

Lemma 5, also taken from Bloniarz, describes the conditions under which S can be expanded in Spira's algorithm, and shows that the algorithm is correct.

LEMMA 5. *Suppose candidate* (c_0, t_0) *is such that*

$$D[c_0] + C(c_0, t_0) = \text{minimum}\,\{D[c] + C(c, t)\colon (c, t) \text{ is a candidate}\}.$$

If t_0 *is outside* S *then* $L(s, t_0) = D[c_0] + C(c_0, t_0)$, *and* $L(s, t_0) \leqq L(s, v)$ *for all* v *not yet in* S. □

Spira reduced the time spent scanning adjacency lists looking for useful edges, but at the expense of an increased number of candidates that must be examined during the labelling steps. Making the crucial assumption that the minimal cost candidate leads to each of the vertices with equal probability (§ 3), Lemma 2 can be seen to apply and the total expected number of drawings will be $O(n \log n)$. Each drawing requires $O(\log n)$ effort for the corresponding tree manipulation, meaning that the total effort to solve a single source problem is on average $O(n \log^2 n)$. The worst case requirement is $\Theta(n^2 \log n)$, when every edge in the graph needs to be processed through the heap as a candidate.

3. Random graphs. An average case analysis requires a definition of the type of "randomness" over which the average is to be taken. Spira analysed the expected behaviour of his algorithm in terms of a randomness model in which the edges of a complete graph are assigned costs by $n(n-1)$ independent drawings from any random distribution. The distribution itself is arbitrary. Bloniarz redefined this randomness model, widened it to a class he calls "endpoint independent" graphs, and showed that

the average running time of a slightly modified Spira's algorithm on this wider class of random graphs is still $O(n \log^2 n)$.

Bloniarz gives this description of the properties of a graph drawn from an endpoint independent probability distribution: "Suppose a particular edge is selected from the sorted edge list by virtue of either its position on the list or the value of its cost. If P [the probability measure] is endpoint independent, then the endpoint of this edge is independent of the edge's selection; every endpoint is equally likely" ([3, p. 594]). This requires that edges in the edge list of equal cost be stored in a random order within the overall sorted order [2]; once this is done each sorted edge list will be a random permutation of endpoints, and the repeated examination of the endpoints of edges as any edge list is scanned can be taken to be a sequence of independent trials.

The average time analyses of this paper make this same assumption—that the destination of an edge is independent of the source of the edge, the cost of the edge, and the position of the edge in the sorted edge list. The analyses also require that the endpoint of the minimum cost candidate be independent of the cost of the candidate; this means that the data structure should be implemented with a tie-breaking rule so that in the case where there are two candidates of equal and minimal weight the tie should be broken without reference to the candidate endpoints. This is readily accomplished by requiring that the index of the candidate source vertex be used as a secondary key. If the priority queue is permitted to adjudicate between elements only on the basis of cost and source vertex, then endpoint independence means that each element in the queue will be storing a candidate with endpoint equally likely to be any of the possible destinations. That is, if a vertex v is chosen at random from $V-S$, and each candidate has a random member of $V-S$ as its endpoint, then each element in the queue will have endpoint v with the same probability $p=1/|V-S|$, satisfying the stipulation of Lemma 3.

A number of types of edge cost assignments meet the endpoint independence requirement. The simplest situation is when each edge in the graph is independently assigned a value from any single distribution of any sort. Also within the scope of the definition is the situation in which each source vertex has associated with it some different random distribution function and edges are assigned costs independently drawn from the distribution of the corresponding source. Other edge cost assignment schemes that meet the requirements of endpoint independence are discussed by Bloniarz [3].

4. The new algorithm. The hybrid algorithm is essentially as follows: Dantzig's algorithm, but using a heap priority queue, is used to label vertices until the *critical point* when $|S| = n - n/\log n$; thereafter Spira's algorithm is used for the labelling, but on a subset of the edges. The reason why this point in the progress of the algorithm has been identified as critical will become apparent in the development below.

Spira employed two main techniques: he used a tree to enable minimal weight candidates to be found efficiently, and he relaxed the requirements on candidates so that the edge scanning component of Dantzig's algorithm could be reduced.

Consider the algorithm that would result if Dantzig's algorithm, including the strong candidacy rule, were implemented with a heap. The running time of such an algorithm would still consist of two components: the labelling of the minimal weight candidate followed by the replacement of candidates that become useless; and the scanning for useful edges.

Suppose at some point of the computation that $|S|=j$, and assume that each candidate leads to each of the $n-j$ unlabelled vertices with equal probability. Then

in each iteration of the while loop of Dantzig_expand_S it can be expected that $1+(j-1)/(n-j)$ candidates will need to be replaced. This is because the root of the heap will always by definition become useless, and each of the remaining $j-1$ candidates will become useless with probability $p=1/(n-j)$. The description in § 2 indicated that $O(j)$ effort would be sufficient to identify the useless candidates; in fact they can be identified in time linear in the number of such candidates by recording for each vertex a list of candidates that currently have that vertex as endpoint. Lemma 3 then implies that the useless candidates can be replaced in $O(j/(n-j)+\log j)$ time on average, including the identification of those candidates affected and the unavoidable replacing of the least cost element.

This expression is $O(\log n)$ only when $j \leqq n-n/\log n$; it is this realisation that forces the identification of the critical point. Beyond the critical point too many candidates become useless at each stage for the strong candidacy rule to continue to be economical. To expand S from 1 element to $n-n/\log n$ elements will require only $O(n \log n)$ expected time for heap operations.

The second component of the running time is the effort spent scanning for useful edges. Define U to be the set of unlabelled vertices when $|S|=n-n/\log n$. Let an edge be called a *U-edge* if it leads to a vertex in the set U. There will be $n/\log n$ such U-edges scattered through each edge list. Because each edge list is a random permutation of endpoints the least cost U-edge in each list will lie on average in the $\log n$th position, and strong candidacy means that when $|S|=n-n/\log n$ each edge list pointer will be pointing at the least cost U-edge in the corresponding edge list. Thus the total scanning effort until $|S|=n-n/\log n$ must be on average $O(n \log n)$.

From this it can be seen that the call "Dantzig_expand_$S(n-n/\log n)$" will require in an expected sense $O(n \log n)$ time. The use of Dantzig's algorithm to label the remaining $n/\log n$ vertices would result in an excessive $\Theta(n^2)$ scanning effort, and so at the critical point when $|S|=n-n/\log n$ the strong candidacy rule is dispensed with. To label the remaining $n/\log n$ vertices the rule is weakened and a slight modification of Spira's procedure is used.

Spira's algorithm does not require that all candidates be useful. Rather, when a new candidate for some vertex is required, the next shortest edge in the appropriate edge list is used, irrespective of whether or not it is useful. To improve the efficiency of Spira's algorithm while labelling the last $n/\log n$ vertices, we make the following modification to procedure Spira_expand_S. Instead of the next shortest edge being chosen to be the new candidate, the next shortest U-edge is selected. With this small change, all candidates will at all times have as their endpoint a random member of the set U. Each time a new candidate is required there will be some edge scanning necessary to find a U-edge, exactly the operation that Spira sought to avoid. However, the time spent on edge list scanning is bounded, and the identification of the next U-edge in any edge list will require on average the skipping of only $\log n$ edges from the current position of the edge list pointer (Lemma 1). Moreover, since each heap operation during the course of Spira_expand_S requires $O(\log n)$ time, the $O(\log n)$ effort spent on edge scanning is absorbed by the requirement of the heap operation. The total cost of candidate replacement, involving scanning and a heap operation, is $O(\log n)$.

In the situation being considered there are $|U|=n/\log n$ vertices that must be labelled, and each candidate is a random member of the set U. Thus the expected number of while loop iterations required (Lemma 2) before all of the set U is labelled is $(n/\log n)\ln(n/\log n)$, which is $O(n)$. Each iteration will require $O(\log n)$ time, so the total expected time to label the set U will be $O(n \log n)$.

This synthesis of the ideas of Dantzig and Spira means that the program given as Algorithm 3 will solve a single source problem in $O(n \log n)$ expected time, given that:

(i) the edge lists are pre-sorted;
(ii) the graph is endpoint independent; and
(iii) the set of candidates is implemented as a heap.

The correctness of the algorithm follows directly from Lemmas 4 and 5.

ALGORITHM 3. THE NEW SINGLE SOURCE ALGORITHM.

```
procedure Fast_Single_Source (s);
  S := {s}; D[s] := 0;
  initialise the set of candidates to {(s, t)}, where (s, t) is the
    shortest edge from s;
  Dantzig_expand_S (n − n/log n);
  U := V − S;
  Spira_expand_S (n), using only the U-edges
end {Fast_Single_Source}
```

In the worst case, each stage of the first Dantzig based phase of Algorithm 3 might require an $O(n)$ time complete heap rebuilding, replacing all of the candidates. This would total $O(n^2)$. The second Spira based phase can also at most require $O(n^2)$ time for heap operations since each edge passing through the heap will require an $O(\log n)$ time heap operation and there are exactly $n^2/\log n$ U-edges that can be processed through the heap. The total edge scanning effort is also bounded at $O(n^2)$, meaning that the worst case running time of Fast_Single_Source is $O(n^2)$. This is an improvement over the $\Theta(n^2 \log n)$ worst case of Spira and Bloniarz, and is equal to the worst case bound of the original Dantzig's algorithm.

The effort required to completely pre-sort the edge lists is $\Theta(n^2 \log n)$. It is in fact not strictly necessary to sort each edge list in entirety; a heap can initially be built for the edges of each vertex in $O(n^2)$ total time, and thereafter each time the next shortest unconsidered edge from a vertex is required it can be found in $O(\log n)$ time with a single heap operation. As the analysis shows, the expected number of edges considered during one complete single source algorithm is on average only $O(n \log n)$, and so the edge sorting time can be reduced to $O(n^2 + n \log^2 n)$ time on average. This dominates the expected time requirement of all of the three single source algorithms discussed, meaning that all three can be used for the single source problem with $O(n^2)$ average case running time. For practical use the $O(n^2)$ algorithm of Dijkstra, which does not require a pre-sort and is a much simpler algorithm, would almost certainly be a better choice.

However if the all pairs problem is to be solved by n calls to a single source algorithm then the pre-sorting need only be done once, and the $\Theta(n^2 \log n)$ requirement for a complete sort can be absorbed. Section 3 showed that all of the randomness assumptions made in the text follow from the definition of endpoint independence; this then gives the result claimed in the introduction:

THEOREM 1. *Let G be a nonnegatively weighted directed graph of n vertices drawn from an endpoint independent probability distribution. Then the all pairs shortest path problem on G can be solved in $O(n^2 \log n)$ expected running time.*

Proof. From the above discussion. □

It seems very likely that the solving of all n single source problems in an endpoint independent graph will *not* require the extraction in sorted order of all n^2 edges [11], and so if edge orderings are retained from one single source problem to the next it is possible that $O(n^2)$ time for edge sorting using n heaps would be sufficient, even for

an all pairs algorithm. If this is the case then it is conceivable that the Dantzig and Spira approach could be further sharpened to yield an $o(n^2 \log n)$ average case running time for the all pairs shortest path problem. Whether in fact such further improvement is possible remains to be seen. It is certainly the case that complete sorting of the edge lists cannot be a part of an algorithm that improves upon $O(n^2 \log n)$.

5. Summary. By identifying a critical point when $|S| = n - n/\log n$, beyond which it becomes computationally expensive to maintain Dantzig's strong candidacy rule, it has been possible to develop an algorithm for the all pairs shortest path problem that requires $O(n^2 \log n)$ time on average, and $O(n^3)$ time in the worst case. This algorithm is an asymptotic improvement over the $O(n^2 \log n \log^* n)$ algorithm of Bloniarz and solves the same general families of graphs; the algorithm also includes entirely the narrow class of probability distribution for which Frieze and Grimmett obtained $O(n^2 \log n)$ running time.

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] P. A. BLONIARZ, A. MEYER AND M. FISCHER, *Some observations on Spira's shortest path algorithm*, Tech. Rept. 79-6, Comp. Sci. Dept., State University of New York at Albany, Albany, NY, 1979.

[3] P. A. BLONIARZ, *A shortest path algorithm with expected time* $O(n^2 \log n \log^* n)$, this Journal, 12 (1983), pp. 588-600.

[4] J. S. CARSON AND A. M. LAW, *A note on Spira's algorithm for the all-pairs shortest path problem*, this Journal, 6 (1977), pp. 696-699.

[5] G. B. DANTZIG, *On the shortest route through a network*, Management Sci., 6 (1960), pp. 187-190.

[6] E. W. DIJKSTRA, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269-271.

[7] W. H. FELLER, *An Introduction to Probability and its Applications*, Vol. 1, 3rd ed., John Wiley, New York, 1968.

[8] R. W. FLOYD, *Algorithm 97: shortest path*, Comm. ACM, 5 (1962), p. 345.

[9] M. L. FREDMAN, *On the decision tree complexity of the shortest path problem*, Proc. 16th IEEE Symposium on the Foundations of Computer Science, 1975, pp. 98-99.

[10] ———, *New bounds on the complexity of the shortest path problem*, this Journal, 5 (1976), pp. 83-89.

[11] A. M. FRIEZE AND G. R. GRIMMETT, *The shortest-path problem for graphs with random arc-lengths*, Discrete Appl. Math., 10 (1985), pp. 57-77.

[12] D. E. KNUTH, *The Art of Computer Programming*, Vol. 1, Fundamental Algorithms, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[13] A. M. MOFFAT AND T. TAKAOKA, *An all pairs shortest path algorithm with expected running time* $O(n^2 \log n)$, Proc. 26th IEEE Symposium on the Foundations of Computer Science, Portland, OR, October 1985, pp. 101-105.

[14] P. M. SPIRA, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time* $O(n^2 \log^2 n)$, this Journal, 2 (1973), pp. 28-32.

[15] T. TAKAOKA AND A. M. MOFFAT, *An* $O(n^2 \log n \log \log n)$ *expected time algorithm for the all shortest distance problem*, in Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Volume 88, P. Dembinski, ed., Springer-Verlag, Berlin, 1980, pp. 643-655.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
005

SORTING AND SELECTING IN ROUNDS*

NICHOLAS PIPPENGER†

Abstract. We present upper bounds for sorting and selecting the median in a fixed number of rounds. These bounds match the known lower bounds to within logarithmic factors. They also have the merit of being "explicit modulo expansion"; that is, probabilistic arguments are used only to obtain expanding graphs, and when explicit constructions for such graphs are found, explicit algorithms for sorting and selecting will follow. Using the best currently available explicit constructions for expanding graphs, we present the best currently known explicit algorithms for sorting and selecting in rounds.

Key words. sorting, selecting, median, parallel computation

AMS(MOS) subject classification. 68P10

1. Introduction and history. The three principal problems of comparison-tree-based complexity theory are sorting, merging and selecting an element of prescribed rank (such as the minimum, maximum or median). For each problem we may study the worst-case complexity using Valiant's parallel comparison-tree model [V]. There are three commonly studied degrees of parallelism for these problems: the case of a single "processor" (or a fixed number of processors), which we shall call the *serial* case; the case of a number of processors equal to the number of elements, which we shall call the *balanced* case; and the case of a number of processors large enough to allow the solution to be found in a fixed number of "rounds," which we shall call the *highly parallel* case.

For sorting n elements it has long been known that, in the serial case, $\Theta(n \log n)$ steps are needed. This implies that $\Omega(\log n)$ steps are needed in the balanced case; Ajtai, Komlós and Szemerédi [AKS] showed that $O(\log n)$ steps are sufficient. For the highly parallel case, Häggkvist and Hell [HH2] showed that $\Omega(n^{1+1/k})$ comparisons are needed to sort in k rounds; this has been improved to $\Omega(n^{1+1/k}(\log n)^{1/k})$ by Alon, Azar and Vishkin [AAV]. Bollobás and Thomason [BT] showed that $O(n^{3/2} \log n)$ comparisons are sufficient to sort in 2 rounds. This has been improved by Alon, Azar and Vishkin [AAV] to $O(n^{3/2} \log n/(\log \log n)^{1/2})$ comparisons, and generalized by Bollobás and Hell [BH1] to $O(n^{1+1/k} \log n)$ comparisons for k rounds. In this paper we shall show that $O(n^{1+1/k}(\log n)^{2-2/k})$ comparisons are sufficient to sort in k rounds. While slightly larger than the upper bound reported by Bollobás and Hell, this result has the merit of being "explicit modulo expansion," about which we shall have more to say later.

For merging two lists of n elements, it has long been known that, in the serial case, $\Theta(n)$ steps are needed. In the balanced case, Valiant [V] showed that $O(\log \log n)$ steps are sufficient; Borodin and Hopcroft [BH2] showed that $\Omega(\log \log n)$ steps are necessary. In the highly parallel case, Häggkvist and Hell [HH3] showed that $\Theta(n^{1+1/(2^k-1)})$ comparisons are needed to merge in k rounds.

For selecting an element of prescribed rank, it is desirable to distinguish between finding the minimum (or selecting an element of fixed rank), which seems to be easier than the general case, and finding the median, to which the general case can be reduced.

* Received by the editors February 18, 1986; accepted for publication (in revised form) January 21, 1987.
† IBM Almaden Research Laboratory K51-802, 650 Harry Road, San Jose, California 95120-6099.

In the serial case, $\Theta(n)$ steps are needed (the bound of $O(n)$ steps for finding the median being due to Blum, Floyd, Pratt, Rivest and Tarjan [BFPRT]). In the balanced case, Valiant [V] showed that $\Omega(\log\log n)$ steps are necessary, and that $O(\log\log n)$ steps are sufficient for finding the minimum; Ajtai, Komlós, Steiger and Szemerédi [AKSS] showed that $O(\log\log n)$ steps are sufficient for finding the median.

In the highly parallel case, Häggkvist and Hell [HH1] showed that $\Omega(n^{1+1/(2^k-1)})$ comparisons are necessary for selection in k rounds, and that $O(n^{1+1/(2^k-1)})$ comparisons are sufficient for finding the minimum in k rounds. Alon, Azar and Vishkin [AAV] have shown that $\Omega(n^{1+1/(2^k-1)}(\log n)^{2/(2^k-1)})$ comparisons are necessary for finding the median. In this paper we shall show that $O(n^{1+1/(2^k-1)}(\log n)^{2-2/(2^k-1)})$ comparisons are sufficient for finding the median in k rounds. This result is also "explicit modulo expansion."

The results that we have just described will be obtained by using a "probabilistic argument"; such an argument shows that a decision tree satisfying certain bounds exists without giving an algorithm (short of exhaustive search) for constructing one. Unlike most previous work on sorting in rounds, however, we shall use only one property of random graphs, namely, that with high probability a random graph is an "expanding graph." Considerable attention has been given to explicit constructions for expanding graphs, and it is not implausible that constructions for expanding graphs as good as those promised by probabilistic arguments will eventually be found. The results of this paper will then give explicit algorithms for sorting and selecting in rounds, achieving the bounds described above. (We have not defined formally what we mean by an "explicit construction." For the purposes of this paper we may take it to mean that the successive nodes in a decision tree may be computed in time polynomial in n.) Using what is presently known about the explicit construction of expanding graphs (see Lubotzky, Phillips and Sarnak [LPS]), we shall give explicit algorithms for sorting in k rounds using $O(n^{1+2/(k+1)}(\log n)^{2-4/(k+1)})$ comparisons, and selecting in k rounds using $O(n^{1+2^{k-2}/(3^{k-1}-2^{k-2})}(\log n)^{2-2^{k-1}/(3^{k-1}-2^{k-2})})$ comparisons.

The best previously known explicit algorithm for sorting in 2 rounds (due to Alon [A]) is $O(n^{7/4})$, based on a projective geometry of dimension 4. No better bound was known for selection. By applying Corollary 9 below with a projective geometry of dimension 3, we could give an explicit algorithm for sorting in 2 rounds using $O(n^{5/3}\log n)$ comparisons. By applying Proposition 6 with a projective geometry of dimension 2, we could give an explicit algorithm for selection in 2 rounds using $O(n^{3/2}(\log n)^2)$ comparisons. These bounds are only slightly weaker than those for $k=2$ obtained from the results of Lubotzky, Phillips and Sarnak, but projective geometries seem to give poor results for $k\geqq 3$. This difficulty may be traced to the fact that the dimension of a projective geometry must be an integer, and the results of Lubotzky, Phillips and Sarnak may be viewed as providing a surrogate for combinatorial geometries with continuously varying dimensions.

The approach taken in this paper is to assume that the number of rounds is fixed, and to minimize as far as possible the number of processors; we have only succeeded in doing this to within logarithmic factors. Another possible approach is to assume that the numbers of elements and processors are given, and to minimize as far as possible the number of rounds. For sorting, Alon, Azar and Vishkin [AAV] have shown that $\Theta(\log n/\log(1+p/n))$ rounds are necessary and sufficient with p processors and n elements. For merging, Kruskal [K] has shown that

$$\Theta\left(n/p+\log\left(\frac{\log n}{\log(1+p/n)}\right)\right)$$

comparisons are necessary and sufficient. It is natural to conjecture that a similar bound applies to finding the median, but it is not yet known whether $O(\log \log n)$ rounds suffice for $O(n/\log \log n)$ processors and n elements.

2. Statements and proofs of results.

THEOREM 1. *For every fixed integer $k \geqq 1$, there are algorithms (explicit modulo expansion) that select an element of prescribed rank from among n elements in k rounds using $O(n^{1+1/(2^k-1)}(\log n)^{2-2/(2^k-1)})$ comparisons.*

THEOREM 2. *For every fixed integer $k \geqq 1$, there are algorithms (explicit modulo expansion) that sort n elements in k rounds using $O(n^{1+1/k}(\log n)^{2-2/k})$ comparisons.*

THEOREM 3. *For every fixed integer $k \geqq 1$, there are explicitly constructed algorithms that select an element of prescribed rank from among n elements in k rounds using $O(n^{1+2^{k-2}/(3^{k-1}-2^{k-2})}(\log n)^{2-2^{k-1}/(3^{k-1}-2^{k-2})})$ comparisons.*

THEOREM 4. *For every fixed integer $k \geqq 1$, there are explicitly constructed algorithms that sort n elements in k rounds using $(n^{1+2/(k+1)}(\log n)^{2-4/(k+1)})$ comparisons.*

The proofs of all these results will use expanding graphs. The following definition of expansion is best adapted to our purposes. Let a be a nonnegative real number. An undirected graph is *a-expanding* if any two disjoint sets of vertices, each containing at least $a+1$ vertices, are joined by an edge. The key to our results is the following lemma, which is implicit in [AKSS].

LEMMA 5. *In an a-expanding graph, for every set of vertices X containing at least $5a$ vertices, there exists a set of vertices Y disjoint from X and containing at most a vertices such that every set of vertices Z disjoint from $X \cup Y$ and containing at most a vertices has at least $2|Z|$ neighbors in X, where $|Z|$ denotes the cardinality of Z.*

Proof. Let Y be a maximal (with respect to inclusion) set of vertices satisfying the following properties: (1) Y is disjoint from X, (2) Y contains at most $2a$ vertices and (3) Y has fewer than $2|Y|$ neighbors in X. Properties (2) and (3) imply that Y has fewer than $4a$ neighbors in X.

We claim first that Y contains at most a vertices. If not, then since the graph is a-expanding, all but at most a of the vertices not in Y would be neighbors of Y, so X in particular would contain at least $5a - a = 4a$ neighbors of Y, a contradiction. Finally, we observe that if Z were a set violating the conclusion of the lemma, then $Y \cup Z$ would be a proper superset of Y satisfying properties (1)-(3), contradicting the maximality of Y. □

This lemma is used to prove the following propsition. We shall assume that our algorithms deal with n distinct and totally ordered elements. If, after some comparisons have been performed, an element v is known to be too large or too small to have rank m, we shall say that v is *excluded* from rank m; otherwise we shall say that v is a *candidate* for rank m. If M is a set of ranks, we shall say that v is *excluded* from M if it is excluded from every rank in M; otherwise we shall say that it is a *candidate* for M.

PROPOSITION 6. *If n elements are compared according to the edges of an a-expanding graph, there will be $O(a \log n)$ candidates for any rank.*

Proof. Suppose that the element of rank m is being sought (where the minimum is the element of rank 1 and the maximum is the element of rank n). Let $b = 1 + \lfloor \log_2 (a+1) \rfloor = O(\log n)$. We shall show that, after the comparisons have been performed, all but at most $a(6b+2) = O(a \log n)$ of the elements with rank greater than m will be known to have rank greater than m. By the dual argument, all but at most $a(6b+2)$ of the elements with rank less than m will be known to have rank less than m, completing the proof.

If $m > n - a(6b+1)$, then there are fewer than $a(6b+1)$ elements with rank greater than m, and there is nothing to prove. Otherwise, classify the elements into three classes as follows. Let G_- contain the smallest $m+a$ elements, let G_0 contain the next $a(6b+1)$ elements, and let G_+ contain the remaining elements. Since $G_- \cup G_0$ contains just $a(6b+1)$ elements with rank greater than m, it will suffice to show that G_+ contains at most a such elements.

Further classify the elements of G_0 into b subclasses $G_1, \cdots, G_b$, with G_1 containing the smallest $6a$ elements of $G_0, \cdots$, and with G_b containing the largest $6a$ elements of G_0. Finally, classify the elments in each of the classes $G_1, \cdots, G_b$ as "good" or "bad" as follows. All of the elements of G_1 are good. Suppose that, for some $1 \leqq c \leqq b-1$, the elements of G_c have been classified and that there are at least $5a$ good elements in G_c. To classify the elements of G_{c+1}, apply Lemma 5 with X as the set of good elements in G_c. Classify as bad those elements of G_{c+1} that belong to the resulting set Y, of which there are at most a; classify as good the remaining elements of G_{c+1}, of which there are at least $6a - a = 5a$. By Lemma 5, this classification has the following property: any set Z of $z \leqq a$ good elements in G_{c+1} has at least $2z$ good neighbors in G_c.

If an element is known to be larger than at least $a+1$ good elements in G_1, then by comparisons according to an a-expanding graph, it is also known to be greater than all but at most a elements in G_-, and thus is known to have rank greater than m. We shall show by induction on c that if, for some $0 \leqq c \leqq b-1$, an element is known to be larger than at least $(a+1)/2^c$ good elements in G_{c+1}, then it is known to have rank greater than m. The basis, $c=0$, is the foregoing observation. Suppose that $c \geqq 1$ and that the element v is known to be larger than at least $(a+1)/2^c$ good elements in G_{c+1}. These good elements in G_{c+1} have at least $2(a+1)/2^c = (a+1)/2^{c-1}$ good neighbors in G_c. By transitivity, v is known to be larger than these elements in G_c, and thus by inductive hypothesis v is known to have rank greater than m. This completes the inductive step. Taking $c = b-1$, we see that if an element is known to be larger than at least one good element in G_b, then it is known to have rank greater than m.

The set G_b contains at least $5a \geqq a+1$ good elements. Thus, by comparisons according to the edges of an a-expanding graph, all but at most a elements in G_+ are known to be larger than at least one good element in G_b. Thus all but at most a elements in G_+ are known to have rank greater than m. □

To prove Theorem 1, we shall use this proposition together with the following lemma.

LEMMA 7. *For all $1 \leqq a \leqq n$ with n sufficiently large, there is an a-expanding graph with n vertices and $O((n^2 \log n)/a)$ edges.*

Proof. If $a \leqq 2 \ln n$, the bound $O(n^2)$ is trivial, so suppose $a \geqq 2 \ln n$. Let G be a random graph on n vertices in which each edge is independently present with probability p and absent with probability $1-p$, where $p = (2 \ln n)/a$. We shall show that (1) the probability that G contains two disjoint sets of $a+1$ vertices not joined by an edge is at most $1/n^2$, and (2) the probability that G contains more than $2(n^2 \ln n)/a$ edges is at most $1/(n \ln n)$. Since $1/n^2 + 1/(n \ln n) < 1$ when $n \geqq 2$, the lemma will follow.

To prove (1), we observe that there are

$$\binom{n}{a+1, a+1} < n^{2(a+1)}$$

ways to choose two disjoint sets of $a+1$ vertices, and that for each choice, the probability that no edge joins them is

$$(1-p)^{(a+1)^2} < e^{-p(a+1)^2}.$$

Thus, the probability of the event in question is at most

$$n^{2(a+1)}\, e^{-p(a+1)^2} = (n^2\, e^{-p(a+1)})^{a+1} = e^{-p(a+1)} < 1/n^2.$$

To prove (2), we observe that the mean and variance of the number of edges in G are $\binom{n}{2}p < (n^2 \ln n)/a$ and $\binom{n}{2}p(1-p) < (n^2 \ln n)/a$, respectively. Thus, by Chebyshev's inequality, the probability that G contains more than $2(n^2 \ln n)/a$ edges is at most $a/(n^2 \ln n) < 1/(n \ln n)$. □

Proof of Theorem 1. We proceed by induction on k. If $k = 1$, the bound $O(n^2)$ is trivial. If $k \geqq 2$, let

$$a = n^{1-1/(2^k-1)}/(\ln n)^{1-2/(2^k-1)}.$$

Compare the elements according to the edges of the graph supplied by Lemma 7. By Proposition 6, at most

$$t = O(a \log n) = O(n^{1-1/(2^k-1)}(\log n)^{2/(2^k-1)})$$

elements will remain as candidates. By inductive hypothesis, the element sought can be selected from these candidates in $k-1$ rounds using

$$O(t^{1+1/(2^{k-1}-1)}(\log t)^{2-2/(2^{k-1}-1)}) = O(n^{1+1/(2^k-1)}(\log n)^{2-2/(2^k-1)})$$

further comparisons. □

To prove Theorem 3, we shall use Proposition 6 together with explicitly constructed expanding graphs provided by the following lemma.

LEMMA 8. *Let p and q be distinct primes congruent to* 1 *modulo* 4, *with $p < q$. Then there is an explicitly constructed $(2(q+1)/(p+1)^{1/2})$-expanding graph with $q+1$ vertices and $(p+1)(q+1)/2$ edges.*

Proof. Lubotzky, Phillips and Sarnak [LPS] have shown that if p and q satisfy the hypotheses of the lemma, there is an explicitly constructed graph G with $q+1$ vertices and $p+1$ edges meeting every vertex such that (1) the largest eigenvalue $p+1$ of the adjacency matrix M of G has multiplicity 1 and (2) all other eigenvalues of M have magnitude at most $2p^{1/2}$. We shall show that G is $(2(q+1)/(p+1)^{1/2})$-expanding.

Suppose that A and B are disjoint sets of m elements that are not joined by an edge in G. We shall show that $m \leqq 2(q+1)/(p+1)^{1/2}$.

Let e be the function that assigns the value 1 to each vertex of G. Then $\langle e, e\rangle = q+1$ and e is the unique eigenvector of M^TM (which equals M^2, since M is symmetric) corresponding to the eigenvalue $(p+1)^2$.

Let f be the function that assigns the value 1 to vertices in A and the value 0 to all other vertices. Then $\langle f, f\rangle = m$. Let the functions $g = e\langle e, f\rangle/n$ be the part of f lying parallel to e and let $h = f - g$ be the part of f lying perpendicular to e. We shall estimate $\langle f, M^TMf\rangle = \langle Mf, Mf\rangle$ in two ways.

First, $\langle f, M^TMf\rangle = \langle g, M^TMg\rangle + \langle h, M^TMh\rangle$. Thus

$$\begin{aligned}\langle f, M^TMf\rangle &\leqq (p+1)^2\langle g, g\rangle + 4p\langle h, h\rangle \\ &= \frac{(p+1)^2m^2}{n} + \frac{4pm(n-m)}{n},\end{aligned}$$

since all eigenvalues of M^TM other than $(p+1)^2$ have magnitude at most $4p$.

Second, let d be the function that assigns the value 0 to vertices in B and 1 to all other vertices. By Cauchy's inequality, $\langle Mf, Mf\rangle \geqq \langle Mf, d\rangle^2/\langle d, d\rangle$. Now the function Mf assigns to each vertex the number of neighbors it has in A. Since no vertex of B

has a neighbor in A, Mf vanishes on all vertices for which d vanishes. Thus $\langle Mf, d\rangle = \langle Mf, e\rangle = \langle f, M^T e\rangle = (p+1)m$ and, of course, $\langle d, d\rangle = n - m$. Thus we have

$$\langle Mf, Mf\rangle \geqq \frac{(p+1)^2 m^2}{n-m}.$$

Combining this inequality with that of the preceding paragraph yields $m \leqq 2(q+1)/(p+1)^{1/2}$ and completes the proof. □

PROPOSITION 9. *For all $1 \leqq a \leqq n$ with n/a sufficiently large, there is an explicitly constructed a-expanding graph with n vertices and $O(n^3/a^2)$ edges.*

Proof. If $a \leqq 8n^{1/2}$, the bound $O(n^2)$ is trivial, so suppose $4(4n/a)^2 \leqq n$. Take $x = (4n/a)^2 - 1$ and $y = n - 1$. By the prime number theorem for arithmetic progressions (see Davenport [D], for example), the number $\pi_{1,4}(x)$ of primes congruent to 1 modulo 4 and at most x satisfies $\pi_{1,4}(x) \sim x/(2 \ln x)$. It follows that $\pi_{1,4}(2x) - \pi_{1,4}(x) \sim x/2 \ln x$. Thus for all sufficiently large n/a there is a prime p congruent to 1 modulo 4 and satisfying $x \leqq p \leqq 2x$ and therefore $(4n/a)^2 \leqq p+1 \leqq 2(4n/a)^2$. Similarly, there is a prime q congruent to 1 modulo 4 and satisfying $y \leqq q \leqq 2y$ and therefore $n \leqq q+1 \leqq 2n$. Furthermore, since $2(4n/a)^2 < n$, we have $p < q$. Applying Lemma 8, we obtain a $(2(q+1)/(p+1)^{1/2})$-expanding graph with $q+1$ vertices and $(p+1)(q+1)/2 = O(n^3/a^2)$ edges. Since $2(q+1)/(p+1)^{1/2} \leqq 4n/(p+1)^{1/2} \leqq a$, this graph is a-expanding. By identifying $q+1-n$ disjoint pairs of vertices to form single vertices we obtain a graph with n vertices, the same number of edges and the same expanding property. □

Proof of Theorem 3. We proceed as in the proof of Theorem 1, but let

$$a = n^{1-2^{k-3}/(3^{k-1}-2^{k-2})}/(\ln n)^{1-2^{k-2}/(3^{k-1}-2^{k-2})}$$

and use Proposition 9. □

To prove Theorems 2 and 4, we shall use the following corollary of Proposition 6.

COROLLARY 10. *If n elements are compared according to the edges of an a-expanding graph, then after the comparisons have been performed we may construct $O(n/(a \log n))$ sets, each containing $O(a \log n)$ elements, such that the relationship between any pair of elements is known unless they both belong to a common set.*

Proof. Partition the ranks into $O(n/(a \log n))$ contiguous intervals, each containing at most $O(a \log n)$ successive ranks. Let $[i, j]$ be such an interval. By Proposition 6, all but at most $O(a \log n)$ of the elements with rank less than i are known to have rank less than i, and all but at most $O(a \log n)$ of the elements with rank greater than j are known to have rank greater than j. Since at most $O(a \log n)$ elements have ranks in $[i, j]$, at most $O(a \log n)$ elements are candidates for the interval $[i, j]$. The sets of elements that we construct will be the sets of candidates for the intervals.

If an element is not a candidate for an interval, then either it is known to be too large to have its rank in that interval or it is known to be too small to have its rank in that interval. It follows that the family of intervals for which an element is a candidate is contiguous.

Consider any pair v and w of elements. If v and w do not both belong to a common set, then the family A of intervals for which v is a candidate is disjoint from the family B of intervals for which w is a candidate. Since these families are each contiguous, either all intervals of A are less than all intervals of B (in which case v must be less than w) or all intervals of A are greater than all intervals of B (in which case v must be greater than w). □

Proof of Theorem 2. We proceed by induction on k. If $k = 1$, the bound $O(n^2)$ is trivial. If $k \geqq 2$, let

$$a = n^{1-1/k}/(\ln n)^{1-2/k}.$$

Compare the elements according to the edges of the graph supplied by Lemma 7. By Corollary 10, we may construct

$$s = O(n/(a \log n)) = O(n^{1/k}(\log n)^{-2/k})$$

sets each containing

$$t = O(a \log n) = O(n^{1-1/k}(\log n)^{2/k})$$

elements and it will suffice to sort each of these sets. By inductive hypothesis, these sets can be sorted in $k-1$ rounds using

$$O(st^{1+1/(k-1)}(\log t)^{2-2/(k-1)}) = O(n^{1+1/k}(\log n)^{2-2/k})$$

further comparisons. □

Proof of Theorem 4. We proceed as in the proof of Theorem 2, but let

$$a = n^{1-1/(k+1)}/(\ln n)^{1-2/(k+1)}$$

and use Proposition 9. □

REFERENCES

[AKS] M. AJTAI, J. KOMLÓS AND E. SZEMERÉDI, *Sorting in c* log *n steps*, Combinatorica, 3 (1983), pp. 1-19.

[AKSS] M. AJTAI, J. KOMLÓS, W. L. STEIGER AND E. SZEMERÉDI, *Deterministic selection in O*(log log *n*) *parallel time*, ACM Symp. on Theory of Computing, 18 (1986), pp. 188-195.

[A] N. ALON, *Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory*, Combinatorica, 6 (1986), pp. 207-219.

[AAV] N. ALON, Y. AZAR AND U. VISHKIN, *Tight complexity bounds for parallel comparison sorting*, IEEE Symp. on Foundations of Computer Science, 27 (1986), pp. 502-510.

[BFPRT] M. BLUM, R. FLOYD, V. PRATT, R. RIVEST AND R. TARJAN, *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448-461.

[BH1] B. BOLLOBÁS AND P. HELL, *Sorting and graphs*, in Graphs and Order, I. Rival, ed., Reidel, Boston, 1985, pp. 169-184.

[BT] B. BOLLOBÁS AND A. THOMASON, *Parallel sorting*, Discrete Appl. Math., 6 (1983), pp. 1-11.

[BH2] A. BORODIN AND J. E. HOPCROFT, *Routing, merging, and sorting on parallel models of computation*, J. Comput. System Sci., 30 (1985), pp. 130-145.

[D] H. DAVENPORT, *Multiplicative Number Theory*, 2nd edition, Springer, New York, 1980.

[HH1] R. HÄGGKVIST AND P. HELL, *Graphs and parallel comparison algorithms*, Congr. Numer., 29 (1980), pp. 497-509.

[HH2] ———, *Parallel sorting with constant time for comparisons*, this Journal, 10 (1981), pp. 465-472.

[HH3] R. HÄGGKVIST AND P. HELL, *Sorting and merging in rounds*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 465-473.

[K] C. P. KRUSKAL, *Searching, merging and sorting in parallel computation*, IEEE Trans. Comput., 32 (1983), pp. 942-946.

[LPS] A. LUBOTZKY, R. PHILLIPS AND P. SARNAK, *Explicit expanders and the Ramanujan conjectures*, ACM Symp. on Theory of Computing, 18 (1986), pp. 240-246.

[V] L. G. VALIANT, *Parallelism in comparison problems*, this Journal, 4 (1975), pp. 348-355.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
006

GENERALIZED STRING MATCHING*

KARL ABRAHAMSON†

Abstract. Given a pattern string of length n and an object string of length m, the string matching problem asks for the positions of all occurrences of the pattern in the object string. This paper investigates a generalization of string matching, in which the pattern is a sequence of pattern elements, each compatible with a set of symbols. The alphabet of symbols is infinite, with its members encoded in a finite alphabet. In contrast to standard string matching, which can be solved in simultaneous linear time and constant space, it is shown that generalized string matching requires a time-space product of $\Omega(n^2/\log n)$ on a powerful model of computation, when the alphabet is restricted to n symbols. Our proof uses a method of Borodin. The obvious algorithm for generalized string matching requires time $O(NM)$, where N is the length of the encoding of the pattern, and M is that of the object string. We describe an algorithm which solves generalized string matching in time $O(N+M+mN^{1/2}\,\text{polylog}\,(n))$.

Key words. string matching, regular expressions, time-space tradeoff

AMS(MOS) subject classifications. 68Q25, 68Q05

1. Introduction.

One of the goals of research in the analysis of algorithms is the characterization of subproblems of a general problem according to their complexity. In particular, one would like to know the influence of certain natural features of the input on the complexity of the problem.

This paper investigates the influence of a natural feature of patterns on the complexity of pattern matching in strings. The investigation involves an application of the time-space tradeoff proof technique of Borodin et al. [5], [4], and an application of Fischer and Paterson's fast pattern matching ideas [7].

1.1. Pattern matching in strings.

In abstract form, a pattern matching problem consists of a representation of a set of strings, called the *pattern*, and a single string, called the *file*. The goal is to find all substrings of the file which are members of the set represented by the pattern.

Sets of strings can be described in many ways. For example, one could indicate by a given string the set of strings within some fixed distance (under some measure) of the given string. Such implicit, or approximate, string matching problems are studied in Ukkonen [11] and Landau and Vishkin [10]. Here, we consider more explicitly given sets.

Two common representations for patterns are regular expressions [3] and simple strings (representing singleton sets). This paper considers a class of patterns intermediate between those two. Let Σ be the alphabet used in the file, and Δ be a set of *pattern elements*. Let $\sim \subseteq \Delta \times \Sigma$ be a given relation. If $\delta \sim \sigma$, say that δ and σ are *compatible*.

A *generalized string pattern* is a finite sequence of pattern elements. Pattern $p = p_1 \cdots p_n \in \Delta^*$ represents the set $\{x_1 \cdots x_n \in \Sigma^* : p_i \sim x_i, \text{ for } i = 1, \cdots, n\}$. Simple string patterns are the special case of generalized string patterns where $\Delta = \Sigma$ and $\sim$ is equality.

We consider a specific set Δ, consisting of two kinds of pattern elements: positive and negative. Let $k \geqq 0$ and $\sigma_1, \cdots, \sigma_k \in \Sigma$. A positive pattern element has the form $\langle \sigma_1 \cdots \sigma_k \rangle$, and is compatible with each of $\sigma_1, \cdots, \sigma_k$, and nothing else. A negative pattern element has the form $[\sigma_1 \cdots \sigma_k]$ and is compatible with every member of Σ

* Received by the editors August 26, 1985; accepted for publication (in revised form) January 23, 1987.
† Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada.

other than $\sigma_1, \cdots, \sigma_k$. For example, pattern $\langle ab\rangle[\ ][a]$ matches any string of length 3 beginning with a or b and not ending with a.

As a convenience, Σ is presumed to contain a distinguished symbol ϕ, which is compatible with every pattern element. Symbol ϕ will not explicitly be written in pattern elements.

It is an important property of both simple string and regular expression patterns that arbitrarily large alphabets can be dealt with, by encoding into a fixed alphabet. Unwanted matches which begin in the middle of an encoded symbol can be avoided by a suitable choice of encoding. Such encoding is possible for the simple reason that equality is a congruence relation.

But $\sim$ is not necessarily a congruence relation. In order to permit encoding of arbitrarily large alphabets, it is necessary to treat the file alphabet Σ not as a primitive alphabet, but as a set of strings which encode its members. Throughout this paper, Σ is presumed to be the infinite set $\{\phi, a_1, a_2, \cdots\}$, where a_i is represented by the string $\#\bar{\imath}$, and $\bar{\imath}$ is the binary representation of i, without leading zeros. Symbol ϕ is represented by itself.

Some terminology will aid in the discussion of string matching. The word *symbol* will be reserved for members of Σ. A member of Δ is called a pattern element. If symbol σ occurs in pattern element δ, write $\sigma \in \delta$. Relation $\in$ should not be confused with relation $\sim$. For example, $a \in [ab]$. A *character* is a member of the finite alphabet used for encoding symbols and pattern elements, including $\{\#, 0, 1, \phi\}$ and brackets.

Throughout, the pattern is $p = p_1 \cdots p_n$ and the file is $f = f_1 \cdots f_m$. $\Sigma_p \subseteq \Sigma$ is the set of symbols occurring in p. Both p and f have more than one associated length measure, depending on the level of detail at which they are viewed. Throughout, n denotes the number of pattern elements in p, N the number of characters in p, and $\hat{N}$ the number of symbols in p. For example, if $p = \langle \#1\#10\rangle\langle \#11\rangle$, then $n = 2$, $\hat{N} = 3$, and $N = 12$. Similarly, $m \geqq n$ denotes the number of symbols in f, and M denotes the number of characters in f.

An *alignment* consists of n consecutive symbols of f, and is referred to by its final symbol position in f. A *match*, or *matching alignment*, is an alignment which is in the set represented by p.

1.2. The model of computation. The standard models for string matching are finite state machines, multi-tape Turing machines, and unit and logarithmic cost random access machines [3]. Each has its drawbacks. Finite state machines are too weak. Turing machines do not permit efficient search structures, such as binary search trees, or efficient representation of graphs. Unit cost RAMs permit algorithms which build up large numbers, unrealistically performing arithmetic on them at unit cost. Logarithmic cost RAMs introduce a logarithmic factor where it is unreasonable to have one, needlessly obscuring analysis. For example, the obvious algorithm to count the length of the input takes $O(n \log n)$ time on a length n input. A model is introduced below which circumvents these problems.

Our algorithms are for a version of random access machines, called time-bounded-magnitude random access machines (TBM-RAMs). Such a machine is a unit cost RAM with one modification. No cell, including the accumulator, may contain a number which exceeds the number of steps executed so far. If a program attempts to generate a number in excess of the time counter, the machine spins its wheels until a time is reached when the number may be stored. Any polynomial function of time would serve equally well as a bound on the magnitude of cells. But typical RAM programs use less space than time, so the TBM-RAM model is suitable.

The input to a TBM-RAM is presumed to be given in a separate, read-only memory, one character per cell. The output is written into a special write-only output memory. Input and output space are not counted in space complexity.

TBM-RAMs provide a compromise between the unit cost and logarithmic cost RAMs, permitting the simplicity and realism of the unit cost RAM for reasonable programs, without permitting abuse of the privilege of assigning unit cost to each instruction. TBM-RAMs simulate, with at most a constant factor loss, the models usually used for string matching, including multitape Turing machines, and random access machines as they are typically employed. Hence, they provide a suitable framework for string matching algorithms.

Some of the numbers generated by our algorithms are very large. Such numbers are not intended to be stored in a single cell; doing so would cause the algorithms to take exponential time on a TBM-RAM. Instead, large numbers are stored in many cells, one bit per cell. It will be apparent, without need for comment, which numbers are large enough to warrant special treatment.

1.3. Summary of results. Pattern matching with simple string patterns is referred to as standard string matching. Standard string matching is solvable in linear time (in $N+M$) [3], [6], [8], [9]. Galil and Seiferas [8] show that standard string matching can, in fact, be solved in simultaneous linear time and constant space.

We have no nontrivial lower time bound for generalized string matching. It is nevertheless possible to prove that, in terms of simultaneous time and space complexity, generalized string matching is harder than standard string matching. Our first result is an $\Omega(n^2/\log n)$ lower bound on the product of time and space complexities of any algorithm for a small subproblem of generalized string matching, for which $m=2n$. For $\log n \leqq S \leqq n$, the subproblem admits space $O(S)$, time $O(n^2 \log (n)/S)$ algorithms.

Regular expression matching is known to be solvable in time $O(NM)$ [3], and no faster algorithm is known. Our second result is an $O(N+M+m\hat{N}^{1/2}\,\text{polylog}\,(n))$ time algorithm for generalized string matching. The existence of such an algorithm is strong circumstantial evidence that generalized string matching is easier than regular expression matching. In the absence of good lower bounds on regular expression matching, no more can be said.

The remainder of this paper is organized as follows. The time-space tradeoff result is proved in § 2. In §§ 3–6, three algorithms for generalized string matching are developed, each building on the ideas of the previous one. Algorithm A is a practical, but quadratic time, algorithm. Algorithm B runs in close to linear time when Σ is finite, but takes more than quadratic time for infinite Σ. Algorithm C is the promised sub-quadratic time algorithm.

2. A time-space tradeoff. This section is concerned with a subproblem of generalized string matching, called the all-out-of-place problem. For the size n problem, the pattern is fixed as $[a_1][a_2]\cdots[a_n]$. The file has $2n$ symbols, which must be among $\{a_1, \cdots, a_n\}$.

We will show that any time T, space S algorithm for the all-out-of-place problem must satisfy $T \cdot S = \Omega(n^2/\log n)$. The algorithm of Galil and Seiferas solves standard string matching over the same alphabet $\{a_1, \cdots, a_n\}$ (suitably encoded) in time $O(N+M) = O(n \log n)$ and constant space; that is, a time-space product of $O(n \log n)$. So, in terms of time-space product, generalized string matching is more difficult than standard string matching.

The all-out-of-place problem actually exhibits a tradeoff. For every $\log n \leqq S \leqq n$, there is a space $O(S)$, time $O(n^2 \log (n)/S)$ algorithm.

2.1. An algorithm demonstrating the tradeoff. We want to find all matching alignments of pattern $[a_1] \cdots [a_n]$ in $f_1 f_2 \cdots f_{2n}$. Notice that each symbol in f is incompatible with exactly one of the pattern elements. In fact, knowing that $f_i = a_j$ is enough to determine one mismatching alignment, namely the position ending at f_{i+n-j}. Suppose we are willing to use $O(n)$ space. Then keep a bit vector MISMATCH (k), for $k = n, \cdots, 2n$. Initialize the vector to all zeros, and then scan the input $f_1 \cdots f_{2n}$, setting MISMATCH $(i+n-j) = 1$ for each $f_i = a_j$ and $0 \leqq i-j \leqq n$. Then the matching alignments are those k such that MISMATCH (k) is 0 at the end.

Suppose we have only $S < n$ bits of space available for the MISMATCH vector. Then simply make $\lceil n/S \rceil$ passes over the input, storing a different section of the MISMATCH vector during each pass. At the end of each pass, output the match positions found during that pass. Each pass requires $O(n \log n)$ time to read f, character by character. The total time required is $O(n^2 \log n/S)$.

We now turn to the lower bound. The lower bound proof technique was introduced by Borodin et al. [5], and has been applied to sorting [5], [4] and other problems [1], [13].

2.2. A model of computation. The lower bound proof involves a nonuniform model of computation, similar to the familiar comparison tree model of sorting programs, and the string matching model of Yao [12]. Following Borodin et al., algorithms in the nonuniform model are called *branching programs.* For the all-out-of-place problem, the pattern is built into the program, and the input is the file, of fixed length $2n$.

The model provides powerful queries. Each node contains a number i, $1 \leqq i \leqq 2n$, representing the query "what is the ith symbol of the input?" There are n arcs exiting the node, each labeled by a symbol. See Fig. 1. The ability of a branching program to extract, in a single query, the value of an entire symbol partially explains the discrepancy between the upper bound of $O(n^2 \log n)$ and the lower bound of $\Omega(n^2/\log n)$.

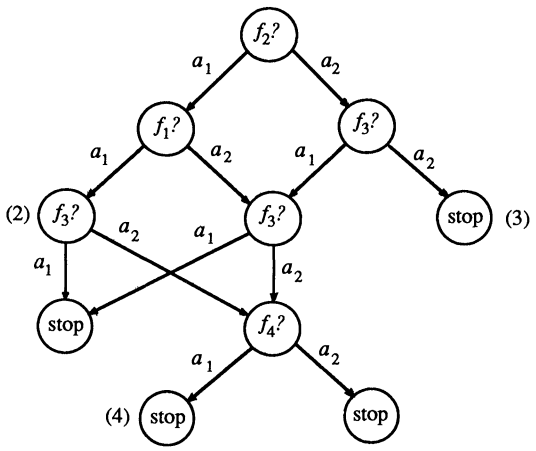


FIG. 1. *A branching program for pattern* $[a_1][a_2]$, $m = 4$.

At various points, an algorithm may conclude that there is a match at some particular alignment. The model provides a means for reporting matching alignments. Each node may be labeled by a collection of output numbers, representing the ending locations of matches. In Fig. 1, outputs are in parentheses beside nodes. Output of the selected matching alignments occurs when the program enters the node. There is

no provision for giving mismatch answers. Instead, the mismatch answers are implicit. A program is correct if, for each input, it outputs all of the matching alignments.

Programs are not restricted to trees, but may be arbitrary directed graphs. One of the nodes is designated the start node, and some are designated stop nodes. The time complexity of a program is the number of queries in the longest computation.

The analogue to space complexity of a branching program is its *capacity*. The capacity of a branching program is the log (base 2) of the number of nodes in the program. Notice that, as is commonly done, we are not counting read-only input space or write-only output space as part of an algorithm's space requirement. We are interested in the amount of temporary storage that an algorithm needs.

Space on TBM-RAMs and capacity of branching programs are related by the following lemma, whose proof is elementary.

LEMMA 2.1. *Suppose there exists a TBM-RAM program P which solves every size n instance of a string matching problem X in time at most T and space at most S. Then there is a branching program for the size n instances of X of time complexity at most T and capacity at most* $S \log T + c$, *where c depends only on P, not on S, T or n.*

Sketch of proof. Let the nodes of the branching program correspond to instantaneous descriptions of the TBM-RAM. The factor of $\log T$ in the capacity accounts for the capacities of individual cells. □

It is convenient to make some assumptions about the structure of branching programs. Say that a branching program is in *normal form* if (1) its nodes can be divided into *levels*, such that the start node is in level 0, and each arc goes from a node in level i to one in level $i+1$, for some i; and (2) all of the stop nodes are in the same level.

Note that a normal form program must be acyclic, and its time complexity is just the largest level number. Any branching program can be put into normal form, by the conceptual addition of a clock which is included in the state space and is incremented at each step. Doing so adds $\log T$ to the capacity of a program of time complexity T. For our purposes, T is polynomial in n, so the added capacity is $O(\log n)$. From now on, we will assume that all branching programs are in normal form. Our results only hold for capacities of at least $\log n$.

Tree programs are a useful restriction of branching programs. Note that a tree program is in normal form if all of its leaves are at the same level. The *depth* of a tree program is its time complexity. Say that a tree program is *nonredundant* if no computation asks the same query twice. Given a depth $q \leqq n$ tree program for a size n problem, one can easily find an equivalent nonredundant, normal form, depth q tree program by simply eliminating redundant queries and then adding worthless but nonredundant queries to pad all paths to the same length.

2.3. Two technical lemmas. Before proving the main theorem, we need two technical lemmas. The first concerns (directed) paths in branching programs. We are only concerned with programs for the all-out-of-place problem. Paths are defined in the usual way; a path of length q has q edges and $q+1$ nodes.

The edges along a path of length q are labeled by the responses to the q queries contained in all but the last node in the path. Say that an input string f is *query-consistent* with a path π provided f is consistent with the responses in π. For example, if π contains the response "a_3" to query "f_9?," then f is query-consistent with π only if $f_9 = a_3$.

The nodes on a path π may have outputs associated with them. Say that an input f is *output-consistent* with π provided the outputs given by π are correct for input f.

For example, if π contains the output 34, then f is output-consistent with π only if f matches $[a_1]\cdots[a_n]$ ending on f_{34}.

The *success set* of a path π is the set of inputs with which π is both query-consistent and output-consistent. Say that π is a *k-path* if it gives at least k outputs. A path is *nonredundant* if it does not contain two nodes with the same query.

LEMMA 2.2. *Any nonredundant k-path of length q has a success set of size at most* $(n-k(n-q)/(2n-q))^{2n-q}$.

Proof. The path makes exactly q distinct queries, and receives the values of q input symbols. There remain $2n-q$ *unqueried* input symbols.

Each output declares a match of $[a_1]\cdots[a_n]$ at some alignment in the input. At least $n-q$ of the n matched positions of each output must be unqueried. Therefore, each output restricts the success set in at least $n-q$ unqueried positions. Specifically, one symbol is forbidden in each position.

Notice that two overlapping matches restrict the input in disjoint ways. For example, a match beginning at the third input symbol f_3 requires that f_6 not be a_4, while a match beginning at f_5 requires that f_6 not be a_2. Altogether, the k outputs must forbid at least $k(n-q)$ symbol occurrences, where a symbol occurrence is a pair (symbol, input position).

A product of positive numbers $x_1x_2\cdots x_r$ is maximized subject to $x_1+\cdots+x_r\leqq c$ where the x_i are all equal. It follows that the success set has maximum size when the $k(n-q)$ forbidden symbol occurrences are distributed uniformly among the $2n-q$ unqueried positions. In that case, there are at most $n-k(n-q)/(2n-q)$ allowable symbols in each unqueried position. There is only one allowable symbol in each queried position, its value determined by the response to the query. The total number of inputs which are both query-consistent and output-consistent with the path is therefore at most $(n-k(n-q)/(2n-q))^{2n-q}$. □

The second technical lemma concerns the frequency of inputs which have many matching alignments. One would expect matches of the pattern $[a_1]\cdots[a_n]$ to be common. The following lemma meets that expectation. Say that a length $2n$ input string is *admissible* if it matches $[a_1]\cdots[a_n]$ in at least $n/10$ alignments.

LEMMA 2.3. *There are at least* $n^{2n}/6$ *admissible inputs, for all* $n\geqq 2$.

Proof. The probability that a randomly chosen input matches $[a_1]\cdots[a_n]$ at a given alignment is $((n-1)/n)^n$, which is at least $1/4$ for $n\geqq 2$. Therefore, the total number of matches, summed over all inputs and all alignments, is at least $(n/4)n^{2n}$. Suppose a fraction r of the inputs match $[a_1]\cdots[a_n]$ in less than $n/10$ alignments. The remaining fraction $1-r$ match in at most n alignments, since a little thought reveals that there must be at least one mismatch for every input (consider the value of f_n). Thus $(rn/10+(1-r)n)n^{2n}\geqq(n/4)n^{2n}$, from which it follows that $r\leqq 5/6$. So at least $1/6$ of the n^{2n} inputs match in $n/10$ or more alignments. □

2.4. A time capacity tradeoff.

THEOREM 2.4. *There is a constant* $c>0$ *such that, for* $S\geqq\log n\geqq 1$, *every time T and capacity S branching program for the all-out-of-place problem satisfies* $S\cdot T\geqq cn^2$.

Proof. Consider a normal form branching program P which solves the all-out-of-place problem in time T and capacity S. Divide P into disjoint *stages*, each containing $q+1<T$ consecutive levels. See Fig. 2. If necessary, pad $T+1$ to a multiple of $q+1$; q will be chosen sufficiently small that the padding is insignificant. Obtain a new program P' by unwinding P within each stage, duplicating shared nodes, so that each stage becomes a collection of disjoint depth q trees. By a leaf of such a tree, we mean a node at the deepest level in a stage. Modify the trees so that each is nonredundant, and every path from a root to a leaf has length exactly q.

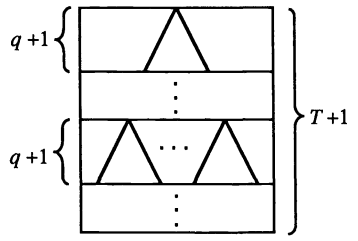


FIG. 2. *A staged branching program.*

In what follows, by a path in P', we mean a path from the root to a leaf of one of the trees just constructed. Each tree has exactly n^q paths, all nonredundant. There are at most 2^S trees in P', since the root of each tree corresponds to a distinct node in P. Hence, there are at most $2^S n^q$ paths in P'.

Imagine running P' on an admissible input. P' must give at least $n/10$ outputs, and there are only $(T+1)/(q+1) < T/q$ stages, so some stage must give at least $nq/10T$ outputs. Hence, one of the paths in one of the stages is an $(nq/10T)$-path. Indeed, for every admissible input, there is an $(nq/10T)$-path which is followed during computation on that input. The paths need not all be distinct, since one path may serve many inputs. Let $h_1, \cdots, h_r$ be all of the $(nq/10T)$-paths in P'.

Let s_i be the success set of h_i, for $i = 1, \cdots, r$. Since each admissible input must have an associated h_i, Lemma 2.3 ensures that

$$|s_1| + \cdots + |s_r| \geqq \frac{n^{2n}}{6}. \tag{2.1}$$

But each h_i is a nonredundant $(nq/10T)$-path of length q. By Lemma 2.2, for $i = 1, \cdots, r$,

$$|s_i| \leqq \left(n - \frac{nq(n-q)}{10T(2n-q)}\right)^{2n-q}. \tag{2.2}$$

Combine and simplify inequalities (2.1) and (2.2), to obtain

$$rn^{-q}\left(1 - \frac{q(n-q)}{10T(2n-q)}\right)^{2n-q} \geqq \frac{1}{6}.$$

But $r \leqq 2^S n^q$, since there are only that many paths in P'. Let $q = n/2$. Then

$$2^S\left(1 - \frac{n}{60T}\right)^{3n/2} \geqq \frac{1}{6}.$$

Taking logs and applying the inequalities $\log(1-x) \leqq -x \log e$ (for $0 \leqq x < 1$) and $T \geqq n$ gives

$$S - \frac{cn^2}{T} \geqq -\log 6 \tag{2.3}$$

for some constant $c > 0$. There are two cases. If $2T \log 6 \geqq cn^2$ then T itself is sufficiently large, and we are done. If $2T \log 6 < cn^2$, then inequality (2.3) implies $ST \geqq cn^2/2$. □

COROLLARY 2.5. *Any TBM-RAM program for the all-out-of-place problem which solves every size n instance in time at most $T(n)$ and space at most $S(n)$ satisfies $S(n) \cdot T(n) = \Omega(n^2/\log n)$.*

3. Overhanging matches. Until now, we have considered only full matches. It is convenient to find *overhanging* matches as well. A left-overhanging match is a suffix of the pattern which matches a prefix of the file. A right-overhanging match is a prefix of the pattern which matches a suffix of the file.

There are two advantages to finding overhanging matches: (a) They provide a means of understanding what is going on at early stages of string matching algorithms, which tend to begin by looking for left-overhanging matches; (b) The file can be broken up into blocks of n symbols and the string matching algorithm run independently on each block. Right-overhanging matches of block b_i are easily combined with left-overhanging matches of block b_{i+1} to obtain full matches for b_ib_{i+1}.

Our algorithms find both left- and right-overhanging matches. In order to include overhanging matches in the definition of matching, extend file $f_1 \cdots f_m$ to $f_{1-n} \cdots f_{m+n-1}$, where, for $i<1$ and $i>m$, f_i is presumed to be ϕ.

4. Algorithm A. This section presents some basic definitions and lemmas, leading to an algorithm for generalized string matching which, although of quadratic time complexity, is quite practical and includes some of the basic ideas of later algorithms. This section contains no proofs, due to the simplicity of the results.

Algorithm A is based on the ideas underlying the Knuth, Morris and Pratt [9] string matching algorithm, without the failure function computation. Extend the relation $\sim$ to strings in the obvious way. Define the collection of sets

$$R_k = \{j\colon 1 \leqq j \leqq n \text{ and } p_1 \cdots p_j \sim f_{k-j+1} \cdots f_k\},$$

for $k = 0, \cdots, m$. Inspection of the definition of R_k reveals that the alignment ending on f_k matches p if and only if either $0 \leqq k \leqq m$ and $n \in R_k$ or $m < k \leqq m+n-1$ and $m+n-k \in R_m$. Thus, it suffices to compute the sets R_k for $k = 0, \cdots, m$. The following definitions and proposition provide a means of computing R_k. For $\sigma \in \Sigma$, and S a set of natural numbers let

$$S+1 = \{x+1\colon x \in S\},$$

$$M(\sigma) = \{j\colon 1 \leqq j \leqq n \text{ and } p_j \sim \sigma\}.$$

PROPOSITION 4.1.

(a) $R_0 = \{1, \cdots, n\}$,

(b) $R_k = ((R_{k-1}+1) \cup \{1\}) \cap M(f_k)$ *for* $k > 0$.

The sets R_k must be represented somehow. A natural representation of a set $S \subseteq \{1, \cdots, n\}$ is the n-bit binary integer $\text{rep}(S) = \sum_{i \in S} 2^{i-1}$. Let $r_k = \text{rep}(R_k)$ and $\mu(\sigma) = \text{rep}(M(\sigma))$. Then Proposition 4.1 implies the following.

PROPOSITION 4.2.

(a) $r_0 = 2^n - 1$,

(b) $r_k = (2 \cdot r_{k-1}+1)\ \&\ \mu(f_k)$ *for* $k > 0$,

where & *is the bitwise Boolean* <u>*and*</u> *operation on binary representations.*

Algorithm A computes r_k, for $k = 0, \cdots, m$, by the recurrences of Proposition 4.2. Doing so requires $O(m)$ arithmetic and Boolean operations on n bit numbers, plus the operations required to compute $\mu(f_k)$, for $k = 1, \cdots, m$.

Compute the μ values as follows. Construct a search structure for $\sigma \in \Sigma_p$, such that σ can be found in time proportional to the length of its encoding. In one pass over p, construct representations, for each $\sigma \in \Sigma_p$, of the sets

$$P^- = \{j\colon p_j \text{ is a negative element}\},$$

$$M^+(\sigma) = \{j\colon \sigma \in p_j \text{ and } p_j \text{ is a positive element}\},$$

$$M^-(\sigma) = \{j\colon \sigma \in p_j \text{ and } p_j \text{ is a negative element}\}.$$

Then compute $\mu(\sigma)$, for each $\sigma \in \Sigma_p$, according to

$$\mu(\sigma) = \text{rep}\,(M^+(\sigma)) \cup (P^- \cap \overline{M^-(\sigma)}).$$

The total time required is $O(N + n|\Sigma_p|)$. Whenever $\mu(\sigma)$ is needed, it is looked up in the search structure. If σ is not found, then $\sigma \notin \Sigma_p$, and $\mu(\sigma) = \text{rep}\,(P^-)$.

On a TBM-RAM, Algorithm A takes time $O(M + N + nm + n|\Sigma_p|)$. In subsequent sections, asymptotically faster Algorithms B and C are developed. But, in spite of its quadratic time complexity, Algorithm A is the most practical of the three algorithms for the following reason.

In typical applications, patterns containing more than a few dozen pattern elements are rare. But the numbers involved in Algorithm A are just n bits long. Since arithmetic and Boolean operations on 32 bit numbers can typically be done in a single fast instruction, it is reasonable to presume that arithmetic and Boolean operations take constant time. In that case, Algorithm A runs in time $O(N+M)$, with a reasonably small constant hidden in the "O."

5. Algorithm B. Fischer and Paterson [7] describe an algorithm for a subproblem of generalized string matching: string matching with don't cares. In our notation, their algorithm permits patterns of two forms, $\langle\sigma\rangle$ and [ ], for $\sigma \in \Sigma$. This section applies Fischer and Paterson's ideas to generalized string matching.

Just as Algorithm A is based on the sets R_k, Algorithm B is based on a collection of functions. For $i = n+1, \cdots, n+m-1$, define p_i to be [ ]. For $k = 0, \cdots, m$ and $i = 1, \cdots, m+n-1$, let

$$G_k(i) = |\{j\colon 1 \leqq j \leqq i \text{ and } p_j \not\sim f_{k+j-i}\}|,$$

$$\nu_k(i) = \begin{cases} 0 & \text{if } p_i \sim f_k, \\ 1 & \text{if } p_i \not\sim f_k. \end{cases}$$

$G_k(i)$ counts the number of incompatible pairs of symbols when $p_1 \cdots p_i$ is aligned with $f_{k-i+1} \cdots f_k$. An analogue to Proposition 4.1 is easily obtained for G_k.

PROPOSITION 5.1.

(a) $G_0(i) = 0$ *for* $i = 1, \cdots, m+n-1$,

(b) $G_k(1) = \nu_k(1)$ *for* $k = 0, \cdots, m$,

(c) $G_k(i) = G_{k-1}(i-1) + \nu_k(i)$ *for* $k > 0$ *and* $i > 1$.

G_k can be viewed as an alternate representation of R_k, since, for $1 \leqq i \leqq n$, $G_k(i) = 0$ if and only if $i \in R_k$. G_k can itself be represented by an integer g_k. Let d be the smallest power of 2 larger than n, and let

$$g_k = \sum_{i=1}^{m+n-1} G_k(i) d^{i-1}, \qquad \mu_k = \sum_{i=1}^{n} \nu_k(i) d^{i-1}.$$

The value μ_k is merely an encoding, in base d, of the set $M(f_k)$ of § 4. Although the encoding is different from that of § 4, notably in polarity, both are named μ. An analogue to Proposition 4.2 follows from Proposition 5.1.

PROPOSITION 5.2.

(a) $g_0 = 0$,

(b) $g_k = d \cdot g_{k-1} + \mu_k$ *for* $k > 0$.

Note that g_k encodes a vector of integers, and the integer addition in part (b) simulates component-wise vector addition. The choice of $d > n$ ensures that carries cannot cross component boundaries.

An obvious algorithm computes g_k, for each k, according to Proposition 5.2. It turns out, however, that it is enough to compute the single quantity g_m. Notice that

G_m counts incompatible pairs when a prefix of p is aligned with a suffix of f; that is, G_m looks for right-overhanging matches. But *every* match of $p_1 \cdots p_n$ in $f_1 \cdots f_m$ is a right-overhanging match of $p_1 \cdots p_n[\ ] \cdots [\ ]$ in $f_{1-n} \cdots f_m$. So p matches the alignment ending on f_k, for $k = 1, \cdots, m+n-1$, if and only if $G_m(m+n-k) = 0$. Thus, all of the desired information can be extracted from g_m. From Proposition 5.2,

$$g_m = \mu_m + \mu_{m-1} d + \mu_{m-2} d^2 + \cdots + \mu_1 d^{m-1}. \tag{5.1}$$

The value of μ_k depends only on the value of f_k. If $f_k = \sigma$, say that $\mu_k = \mu_\sigma$. Any two symbols σ, $\tau \notin \Sigma_p$ are equivalent w.r.t. pattern p, so $\mu_\sigma = \mu_\tau$. Define μ^- to be the value of μ_σ for $\sigma \notin \Sigma_p$. If a symbol σ occurs more than once in f, then μ_σ occurs more than once in the r.h.s. of equation (5.1). Let $I_\sigma = \{k: f_k = \sigma\}$ and $I^- = \{k: f_k \notin \Sigma_p\}$. Collect like coefficients together to obtain

$$g_m = \sum_{\sigma \in \Sigma_p} \mu_\sigma \sum_{k \in I_\sigma} d^{m-k} + \mu^- \sum_{k \in I^-} d^{m-k}. \tag{5.2}$$

The procedure described in § 4 can be used to compute μ^- and μ_σ, for all $\sigma \in \Sigma_p$. Because μ_σ is an $n \log d \approx n \log n$ bit number, the computation time climbs from $O(N + |\Sigma_p| n)$ to $O(N + |\Sigma_p| n \log n)$.

Equation (5.2) includes $|\Sigma_p| + 1$ terms of the form $\Sigma_k\, d^{n-k}$. Each represents an $m \log d \approx m \log n$ bit number. They can all be computed, in a single pass over f, in time $O(M + (|\Sigma_p| + 1) m \log n)$.

Now equation (5.2) involves $|\Sigma_p| + 1$ multiplications of $n \log n$ by $m \log n$ bit numbers. Algorithm B uses the Schönhage-Strassen integer multiplication algorithm [2], [9], which multiplies an r bit number by an s bit number, $r \leqq s$, in time $O(s \log r \log \log r)$. The time required to carry out the $|\Sigma_p| + 1$ multiplications is therefore $O((|\Sigma_p| + 1) m \log^2 n \log \log n)$.

THEOREM 5.3. *Algorithm* B *solves generalized string matching in time* $O(N + M + (|\Sigma_p| + 1) m \log^2 n \log \log n)$.

For fixed finite Σ, Algorithm B runs in time $O(m \log^2 n \log \log n)$. But for infinite Σ, $|\Sigma_p|$ may be as large as $\hat{N}$, the number of symbol occurrences in p. In that case, Algorithm B is worse than Algorithm A.

6. Algorithm C. A common technique in algorithm design is to break a problem into a few subproblems of the same kind, each exhibiting different characteristics, and to apply to each subproblem an algorithm tailored to that subproblem's characteristics. That is the approach taken in designing Algorithm C.

Two methods of breaking a generalized string matching problem into subproblems are used. The first breaks up strings according to alphabets. For $f \in \Sigma^*$, $p \in \Delta^*$, and $\Gamma \subseteq \Sigma$, let $f(\Gamma)$ be the string obtained from f by replacing each symbol in $\Sigma - \Gamma$ by ϕ and let $p(\Gamma)$ be the pattern obtained by erasing every occurrence of every symbol in $\Sigma - \Gamma$ from p. For example, $\langle a_1 a_2 a_3 \rangle [a_1 a_4](\{a_1, a_2\}) = \langle a_1 a_2 \rangle [a_1]$. Then the matching alignments of p in f are just those alignments where $p(\Gamma)$ matches $f(\Gamma)$ and $p(\Sigma - \Gamma)$ matches $f(\Sigma - \Gamma)$.

A second breakdown divides the pattern into positive and negative parts. Let p^+ be the pattern obtained from p by replacing each negative pattern element by $[\ ]$. Similarly, obtain p^- from p by replacing each positive element by $[\ ]$. Then the matching alignments of p in f are precisely those alignments where both p^+ and p^- match.

Algorithm B performs badly when $|\Sigma_p|$ is close to $\hat{N}$. But in that case, most of the symbols must occur infrequently in p. An obvious idea is to break the problem up according to frequency of occurrence of symbols in p. Let z be a function of $\hat{N}$ and

n, to be determined later. Define

$$\Sigma' = \{\sigma \in \Sigma_p : \sigma \text{ occurs at most } z \text{ times in } p\},$$

$$\Sigma'' = \Sigma - \Sigma'.$$

The matching alignments of p in f are just those where $p(\Sigma'')$ matches $f(\Sigma'')$, $p^-(\Sigma')$ matches $f(\Sigma')$ and $p^+(\Sigma')$ matches $f(\Sigma')$. Each subproblem is solved separately.

(1) To find the matches of $p(\Sigma'')$ in $f(\Sigma'')$, use Algorithm B. The time required is $O(M+N+(|\Sigma''|+1)m \log^2 n \log\log n)$. But each member of Σ'' occurs at least z times in p. Since p contains only $\hat{N}$ symbols, $|\Sigma''| \leqq \hat{N}/z$. So part (1) takes time $O(M+N+(\hat{N}/z)m \log^2 n \log\log n)$.

(2) Pattern $p^-(\Sigma')$ contains only negative pattern elements. Each occurrence of a symbol σ in $f(\Sigma')$ is responsible for at most z mismatches, one for each occurrence of σ in p. In time $O(N)$, a list of occurrence positions of every member of Σ' in p can be constructed. Then, in time $O(M+zm)$, all of the mismatch location can be recorded, in a bit array.

(3) For brevity, let p^+ denote $p^+(\Sigma')$ and f denote $f(\Sigma')$. The matching alignments of p^+ in f can be found by counting, for each $k = 1, \cdots, m+n-1$, the size of the set $\{j: p_j^+ \not\sim f_{k-n+j}\}$; a count of zero indicates a match.

Notice that incompatible pairs can only occur when a positive element of p^+ is aligned with a non-ϕ symbol of f. Let $\pi(k,j)$ stand for the statement "p_j^+ is positive and $f_{k-n+j} \neq \phi$," and $\alpha(k,j)$ stand for "$p_j^+ \sim f_{k-n+j}$." Then

$$\begin{aligned}|\{j: p_j^+ \not\sim f_{k-n+j}\}| &= |\{j: \neg\alpha(k,j) \text{ and } \pi(k,j)\}| \\ &= |\{j: \pi(k,j)\}| - |\{j: \alpha(k,j) \text{ and } \pi(k,j)\}|.\end{aligned}$$

The term $|\{j: \alpha(k,j) \text{ and } \pi(k,j)\}|$ can be computed by a method very similar to that used for subproblem 2. Where the algorithm for p^- records a mismatch, the algorithm for p^+ increments a counter. There are at most zm pairs (k,j) satisfying $\alpha(k,j)$ and $\pi(k,j)$, so the time required is $O(N+M+zm)$.

Term $|\{j: \pi(k,j)\}|$ can be computed as follows. Form pattern q from p^+ by replacing each positive element by $\langle\ \rangle$. Note that q has only two distinct elements, $\langle\ \rangle$ and $[\ ]$. Form file g by replacing each non-ϕ symbol in f by some fixed symbol $c \neq \phi$. Then run Algorithm B on pattern q and file g. The output is a count, for each alignment of q in g, of the number of incompatible pairs. But that is precisely $|\{j: \pi(k,j)\}|$. Since $|\Sigma_q| = 0$, Algorithm B takes time $O(N+M+m \log^2 n \log\log m)$, which is low order compared to the cost of the rest of the algorithm.

Putting all of the parts together, Algorithm C runs in time $O(N+M+zm+(\hat{N}/z)m \log^2 n \log\log n)$.

THEOREM 6.1. *When* $z = \hat{N}^{1/2} \log n \log\log^{1/2} n$, *Algorithm* C *solves generalized string matching in time*

$$O(N+M+m\hat{N}^{1/2} \log n \log\log^{1/2} n) = O(N+MN^{1/2} \log N \log\log^{1/2} N).$$

Algorithm C is far from being practical. Its purpose is merely to show that generalized string matching does not require $\Omega(NM)$ time. Both Algorithms A and B achieve less than order NM time for some classes of inputs: Algorithm A is better for $n \ll N$ or $m \ll M$, and Algorithm B is better when $|\Sigma_p|$ is small. But Algorithm C is faster than order NM for all sufficiently large N and M, without assumptions about the input.

7. String matching with don't cares. In the case where patterns are restricted to $[\ ]$ and $\langle\sigma\rangle$, for $\sigma \in \Sigma$, there is a faster algorithm than Algorithm C. Suppose all symbols

have the same encoded length K, and suppose the code for σ is $x_1 \cdots x_K$. Then $\langle\sigma\rangle$ can be replaced by $\langle x_1\rangle \cdots \langle x_K\rangle$, [ ] can be replaced by [ ] $\cdots$ [ ] (K times), and matching can be done at the level of characters. This is essentially the technique employed by Fischer and Paterson [7]. The time required to perform the matching, by Algorithm B, is $O(Km \log^2 (Kn) \log\log (Kn))$.

As the string matching problem has been formulated, the input is not given in a constant length code. But, in time $O(M+N)$, it is possible to convert p and f to a constant length code. By breaking f into blocks of length n, as discussed in § 3, and using a different code for each block, it is possible to get by with a coded length of $K = \lceil \log n \rceil$. The total time, including recoding and matching, is $O(M+N+m \log^3 n \log\log n)$.

8. Conclusion and open problems. Our time-space tradeoff result provides a separation between standard and generalized string matching, on the basis of time-space product complexity. The time-space product lower bound is the best lower bound known for generalized string matching. A nonlinear lower bound on time would represent a more clear separation between standard and generalized string matching. Alternatively, a linear or nearly linear time algorithm would greatly increase the significance of the time-space tradeoff. The existence of a linear time algorithm for generalized string matching remains open.

Generalized string patterns represent a particular feature of patterns, namely, that they consist of a concatenation of sets of individual symbols. Do patterns with less strict features admit subquadratic time algorithms? Regular expressions with the equivalent of negative pattern elements represent an obvious generalization of generalized string patterns. Other obvious ones are: (a) patterns representing sets of uniform length strings; (b) star-free regular expressions.

Generalized string patterns with only positive pattern elements can be viewed as star-free regular expressions, in which concatenation operations may not occur within the scope of union operators. Aho and Corasick [2] describe a linear time algorithm for matching with star-free expressions in which union operators may not occur within the right scope of concatenation operators. Although our algorithm and theirs are incompatible, the existence of both algorithms suggests that there might be a subquadratic time algorithm for pattern matching with arbitrary star-free regular expressions.

REFERENCES

[1] K. ABRAHAMSON, *Time-space tradeoffs for branching programs contrasted with those for straight-line programs*, in Proc. 27th Annual IEEE Symposium Foundations of Computer Science, 1986, pp. 402-409.

[2] A. V. AHO AND M. J. CORASICK, *Efficient string matching: an aid to bibliographic search*, Comm. Assoc. Comput. Mach., 18 (1975), pp. 333-340.

[3] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[4] A. BORODIN AND S. COOK, *A time-space tradeoff for sorting on a general sequential model of computation*, in Proc. 12th Annual ACM Symposium Theory of Computing, 1980, pp. 294-301.

[5] A. BORODIN, M. J. FISCHER, D. G. KIRKPATRICK, N. A. LYNCH AND M. TOMPA, *A time-space tradeoff for sorting on non-oblivious machines*, J. Comput. System. Sci., 22 (1981), pp. 351-364.

[6] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 762-772.

[7] M. J. FISCHER AND M. S. PATERSON, *String matching and other products*, in Complexity of Computation, R. M. Karp, ed., (SIAM-AMS Proceedings 7), American Mathematical Society, Providence, RI, 1974, pp. 113-125.

[8] Z. GALIL AND J. SEIFERAS, *Time-space optimal string matching*, in Proc. 13th Annual ACM Symposium Theory of Computing, 1981, pp. 106–113.

[9] D. E. KNUTH, J. H. MORRIS, JR. AND V. R. PRATT, *Fast pattern matching in strings*, this Journal, 6 (1977), pp. 323–350.

[10] G. M. LANDAU AND U. VISHKIN, *Efficient string matching with k mismatches*, Theoret. Comput. Sci., 43 (1986), pp. 239–249.

[11] E. UKKONEN, *Finding approximate patterns in strings*, J. Algorithms, 6 (1985), pp. 132–137.

[12] A. C. YAO, *The complexity of pattern matching for a random string*, Technical Report STAN-CS-77-629, Dept. Computer Science, Stanford University, Stanford, CA, 1977.

[13] Y. YESHA, *Time-space tradeoffs for matrix multiplication and discrete Fourier transform on any general sequential random access computer*, J. Comput. System Sci., 29 (1984), pp. 183–197.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987

© 1987 Society for Industrial and Applied Mathematics
007

ON THE EXACT SOLUTION OF RANDOM TRAVELLING SALESMAN PROBLEMS WITH MEDIUM SIZE INTEGER COEFFICIENTS*

A. M. FRIEZE†

Abstract. Let edge weights for the complete graph on vertex set $\{1, 2, \cdots, n\}$ be chosen independently and uniformly from $\{0, 1, \cdots, B(n)-1\}$ where $B(n) = o(n/\log\log n)$. We show that there exists a polynomial time $(O(n^3 \log n))$ algorithm which solves the associated travelling salesman problem with probability tending to 1 as n tends to ∞.

Key words. travelling salesman problem, probabilistic analysis

AMS(MOS) subject classifications. 68A20, 68F15

1. Introduction. The Travelling Salesman Problem (TSP) is one of the most studied problems in combinatorial optimisation and a recent book [16] is devoted entirely to its analysis.

We shall consider the *symmetric* problem in which one is given edge weights $c: E(K_n) \to Z$ where $E(K_n)$ is the edge set of the complete graph K_n with vertex set $V_n = \{1, 2, \cdots, n\}$. The problem is to find a tour of hamilton cycle of K_n of minimum total weight.

We shall look at the problem from the point of view of probabilistic analysis. Previous work e.g. Karp [14], [15] or Halton and Terada [13], has concentrated on solving the problem approximately with high probability. In this paper we make some progress on the problem of finding a polynomial time algorithm that can solve the problem *exactly* with probability tending to 1 as n tends to infinity.

In order to do this, we have to be more restrictive about edge weights than previous authors. We assume that $\{c(e): e \in E(K_n)\}$ is a set of independent random variables where each $c(e)$ is chosen uniformly from $\{0, 1, \cdots, B-1\}$ where $B = B(n)$. We prove the following.

THEOREM 1.1. *If* $B(n) = o(n/\log\log n))$ *then there is a polynomial time* $(O(n^3 \log n))$ *algorithm ALGT such that*

$$\lim_{n\to\infty} \Pr(\textit{ALGT solves TSP exactly}) = 1. \qquad \square$$

ALGT can be considered as an extension of the algorithm HAM described in Bollobás, Fenner and Frieze [4] for finding hamilton cycles in random graphs, although we have had to make several changes in the description and proof of validity. To see the relationship, consider the set of edges of length zero. The subgraph G_0 of K_n induced by these edges is distributed like $G_{n,p}$, $p = 1/B(n)$, where each edge of K_n is independently included with probability p and excluded with probability $1-p$. If p is large enough then G_0 is hamiltonian with probability tending to 1 and HAM can be used to find such a cycle with probability tending to 1. Here p large enough is equivalent to $B(n) \approx n/\log n$. If $B(n) > n/\log n$ then G_0 is almost always nonhamiltonian and this paper is concerned with dealing with this problem for $B(n)$ growing faster but not too fast.

The paper is in 2 parts. In the first part, §§ 2–4, we describe an algorithm which we can prove correct for $B(n) = o(n/\sqrt{\log n})$ and which runs in $O(n^{2+o(1)} \log n^{(\log n/\log d)})$ time, where $d = n/B(n)$. If d grows proportionally to $\log n$

* Received by the editors October 23, 1985; accepted for publication (in revised form) February 13, 1987.

† Department of Mathematics, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, and Department of Computer Science and Statistics, Queen Mary College, Mile End Road, London E1 4NS, England.

then this is nearly as good as is claimed in the abstract. But if d grows like $\sqrt{\log n}$ we can only claim $O(n^{4+o(1)})$. The second part is devoted to actually proving Theorem 1.1. The reason for describing 2 methods is that the first method is deterministic. The only randomisation needed is in the input, i.e., we have a deterministic algorithm that works on almost all problems. Our second algorithm is randomised: it requires the selection of a random subset of the zero length edges. The methods are similar enough so that only a few extra lemmas are needed to prove the validity of the second approach.

Solution strategy. This is quite simple at the top level. We identify a set of "troublesome" vertices X_0 which we "cover" as cheaply as possible by a set of vertex disjoint paths F^* (Phase 1). Having done this we construct a tour through V_n which contains F^* and zero length edges otherwise. It is not then difficult to see that this will be a minimum length tour. Unfortunately there are quite a lot of details.

Notation. For ease of reference we gather together in one place the main notation common to the rest of the paper.

Let $p = 1/B(n) = \omega\sqrt{\log n/n}$ initially, where $\omega = \omega(n) \to \infty$, and let $d = np$. We assume that $d \leqq 1.1 \log n$. (For $d > 1.1 \log n$ we look for a hamilton cycle in G_0.)

Let $E_k = \{e \in E(K_n)\colon c(e) \leqq k\}$, $0 \leqq k < B$ and let $G_k = (V_n, E_k)$. Note that G_k is distributed like $G_{n,(k+1)p}$.

For $v \in V_n$ let $d_k(v)$ denote the degree of v in G_k and

$$\delta_k = \min\{d_k(v)\colon v \in V_n\}.$$

For a graph G, $W \subseteq V(G)$ let $N(W, G) = \{v \in V(G) - W\colon \dagger w \in W$ such that $\{v, w\} \in E(G)\}$

$$S(m, p, k) = \sum_{t=0}^{k} \binom{m}{t} p^t (1-p)^{m-t},$$

$\lambda = \lceil\sqrt{\log n}\rceil$ and $\mu = \lceil 2 \log n/d\rceil$.

An event $\mathscr{E}_n$ dependent on n will be said to occur *almost always* (a.a.) if $\lim_{n\to\infty} \Pr(\mathscr{E}_n) = 1$.

In what follows we will sometimes use real values where we should round up or down. Where we do so the reader can be assured that by rounding up or down the desired objective can be achieved. Furthermore when we claim certain inequalities we will implicitly only claim them for n large.

2. Structural properties of the G_k. Our main problem is in dealing with the following set of vertices:

$$X_0 = \{v \in V_n\colon d_0(v) \leqq d/2\}$$

that are equal to the set of vertices incident with "few" edges of weight zero. The following lemma summarises part of what we usually expect of X_0 and $G_{2\mu}\colon X_0$ is small, $G_{2\mu}$ is reasonably sparse and contains few small cycles. X_0 is (almost) randomly distributed over V_n and so is spread out and a long way from any small cycle.

LEMMA 2.1. *The following properties hold a.a.*

(a) $|X_0| \leqq ne^{-d/10}$.

(b) $G_{2\mu}$ *is connected.*

(c) $Z \subseteq V_n$, $|Z| \leqq 3\lambda$ *and* $|Z \cap X_0| \geqq \lambda$ *implies* $G_{2\mu}[Z]$ *is not connected* {*i.e., X_0 is "spaced out" in $G_{2\mu}$*}.

(d) C *is a cycle in* $G_{2\mu}$, $|C| \leqq 2\lambda$ *implies* $|C \cap X_0| \leqq |C|/10$.

(e) $G_{2\mu}$ *contains fewer than* $(\log n)^9$ *cycles of length* 8 *or less.*

(f) *The maximum degree in $G_{2\mu}$ is $\leqq 30 \log n$.*

Proof. (a)

$$E(|X_0|) = n \sum_{k=0}^{d/2} \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

$$\leqq c_1 \binom{n-1}{\lfloor d/2 \rfloor} p^{\lfloor d/2 \rfloor} (1-p)^{n-1-\lfloor d/2 \rfloor}$$

(for some constant c_1. We use this notation for unspecified constants)

$$\leqq c_2 n \left(\frac{de}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor} e^{-d} \qquad \text{using } \binom{n}{k} \leqq \left(\frac{ne}{k}\right)^k \text{ and } 1+x \leqq e^x$$

$$\leqq c_3 n (e/2)^{-d/2}$$

$$\leqq c_3 n e^{-.15d}.$$

Now use the Markov inequality $-\Pr(|X_0| \geqq x) \leqq E(|X_0|)/x$ for $x > 0$.

(b) This follows from Erdös and Rényi [5] as $G_{2\mu}$ "has more edges" than $G_{n,4 \log n/n}$.

(c) If (b) holds and (c) fails, then there exists $Z \subseteq V_n$, $E \subseteq E_{2\mu}$ such that

(i) (Z, E) is a tree,

(ii) $|Z \cap X_0| \geqq \lambda$, and

(iii) $|Z| = k = 3\lambda$.

Let $\alpha =$ the number of triples $(Z, Z \cap X_0, E)$ in $G_{2\mu}$ satisfying (i), (ii) and (iii). Then

$$\Pr(\alpha \neq 0) \leqq E(\alpha) \leqq \binom{n}{k}\binom{k}{\lambda} k^{k-2} ((2\mu+1)p)^{k-1} S(n-k, p, d/2)^\lambda$$

$$\leqq c_4 \left(\frac{ne}{k}\right)^k \left(\frac{ke}{\lambda}\right)^\lambda k^{k-1} \left(\frac{4 \log n}{n}\right)^{k-1} \left(1 + \frac{3d}{4 \log n}\right)^{k-1} e^{-.15d\lambda}$$

$$\leqq c_5 e^{\log n + 3\lambda \log\log n + 9\lambda - .15\lambda d}$$

$$= o(1).$$

(d) Let $\beta =$ the number of pairs $(C, C \cap X_0)$ where C is a cycle of length $\leqq 2\lambda$ in $G_{2\mu}$ and $|C \cap X_0| \geqq |C|/10$. Then

$$\Pr(\beta \neq 0) \leqq E(\beta) \leqq \sum_{k=3}^{2\lambda} \binom{n}{k} ((k-1)!/2) \binom{k}{\lceil k/10 \rceil}$$

$$\times ((2\mu+1)p)^k S(n-k, p, d/2)^{\lceil k/10 \rceil}$$

$$\leqq c_6 \sum_{k=3}^{2\lambda} \left(\frac{ne}{k}\right)^k ((k-1)!/2)(10e)^{k/10} \left(\frac{8 \log n}{n}\right)^k e^{-.015dk}$$

$$\leqq \sum_{k=3}^{2\lambda} (c_7 \log n \, e^{-.015d})^k$$

$$= o(1).$$

(e) The expected number of cycles of length 8 or less in $G_{2\mu}$ is $O((\log n)^8)$. Now use the Markov inequality.

(f) Let γ equal the number of vertices of degree exceeding $30 \log n$ in $G_{2\mu}$. Then

$$\begin{aligned}\Pr(\gamma \neq 0) \leqq E(\gamma) &\leqq n\binom{n-1}{\lceil 30 \log n \rceil}((2\mu+1)p)^{\lceil 30 \log n \rceil} \\ &\leqq n\left(\left(\frac{ne}{30 \log n}\right)\left(\frac{8 \log n}{n}\right)\right)^{30 \log n} \\ &= o(1)\end{aligned}$$

(we assumed that $d \leqq 1.1 \log n$). □

The next lemma shows that a.a. all vertices have some small incident edges.

LEMMA 2.2. $\delta_\mu \geqq (\log n)/3$ *a.a.*

Proof.

$$\begin{aligned}\Pr(\dashv v \in V_n\colon d_\mu(v) < (\log n/3) &\leqq E(\text{number of vertices of degree} < (\log n)/3) \\ &\leqq n \sum_{k=0}^{(\log n)/3} \binom{n-1}{k} \\ &\quad \times ((\mu+1)p)^k (1-(\mu+1)p)^{n-1-k} \\ &\leqq 2n \sum_{k=0}^{(\log n)/3} \left(\frac{ne}{k}\right)^k \\ &\quad \times \left(\left(\frac{2 \log n}{n}\right)\left(1+\frac{np}{\log n}\right)\right)^k e^{-(np+2 \log n)} \\ &\leqq 2n \sum_{k=0}^{(\log n)/3} \left(\frac{ne}{k}\right)^k \left(\frac{2 \log n}{n}\right)^k n^{-2} \\ &= o(1)\end{aligned}$$

as the largest term in the above summation $= O(n^{-1.06})$. □

3. Phase 1 of ALGT. With these preliminaries taken care of we describe Phase 1 of ALGT in detail.

We first describe a simple lower bound on the length of any tour. Let

$\mathcal{F} = \{F \subseteq E(K_n)$: F induces a set of vertex disjoint paths such that:
(1) the endpoints of each path of F are in $V_n - X_0$,
(2) every vertex in X_0 is in the interior of some path of F,
(3) $e \cap X_0 \neq \varnothing$ for all $e \in F\}$.

If $|X_0| < n/2$ (as it a.a. is by a long way—by Lemma 2.1) then, for any tour T

$$c(T) \geqq LB = \min\{c(F)\colon F \in \mathcal{F}\} \tag{3.1}$$

where for $S \subseteq E(K_n)$, $c(S) = \sum_{e \in S} c(e)$.

To show (3.1) simply delete all edges of T joining 2 vertices not in X_0 in order to obtain a member of $\mathcal{F}$. This also shows that $\mathcal{F} \neq \varnothing$ in this case. The main result of this paper is that in polynomial time we can a.a. find an F^* and extend it to a tour T^* for which

$$c(T^*) = LB \tag{3.2}$$

which implies T^* is optimal.

Phase 1 is concerned with finding F^*.

Gurevich and Shelah [12] also use the idea of covering "troublesome" vertices with paths.

Let L2.1 and L2.2 denote the events defined in Lemmas 2.1 and 2.2. (We shall use this notation to describe several events.)

Let $\mathcal{F}^* = \{F \in \mathcal{F}: c(F) = LB\}$.

LEMMA 3.1. *If* L2.1 $\cap$ L2.2 *occurs and* $F^* \in \mathcal{F}^*$, *then* $F^* \subseteq E_{2\mu}$, *for large n.*

Proof. Assume L2.1 $\cap$ L2.2 occurs. Suppose then that $e = \{a, b\} \in F^*$ and $c(e) > 2\mu$. Assume first that $e \subseteq X_0$. Let $A = \{a_1, a_2, \cdots, a_r\}$, $B = \{b_1, b_2, \cdots, b_s\}$, $r, s \geqq (\log n)/3 - 1$ be such that $\{a, a_i\} \in E_\mu$ and $\{b, b_j\} \in E_\mu$ for $i = 1, 2, \cdots, r$, $j = 1, 2, \cdots, s - A$, B exist because of L2.2., L2.1(d) implies that $|A \cap B| \leqq 1$ and L2.1(c) implies that at most 3λ members of $A \cup B$ are vertices of paths of F^*. Thus we can pick $a_k \neq b_l$ such that $a_k, b_l \in V_n - X_0$ and $F' = (F^* - \{e\}) \cup \{\{a, a_k\}, \{b, b_l\}\} \in \mathcal{F}$. But then $c(F') < c(F)$—contradiction. The case where $a \in X_0$, $b \notin X_0$ is similar. We replace $\{a, b\}$ by an edge $\{a, a_k\} \in E_\mu$ chosen as above. □

Phase 1 of ALGT.

begin

check for the occurrence of L2.1 $\cap$ L2.2—this can easily be done in $O(n^3)$ time;

if either event fails to occur *then* terminate unsuccessfully

else find $F^* \in \mathcal{F}^*$

end

LEMMA 3.2. *If* L2.1 $\cap$ L2.2 *occurs and n is large, then* F^* *can be found in* $O(|E_{2\mu}|)$ *time.*

Proof. Assume that L2.1 $\cap$ L2.2 occurs. Let $E^* = \{e \in E_{2\mu}: e \cap X_0 \neq \emptyset\}$ and $V^* = \{v \in V_n: \dashv e \in E^*$ such that $v \in e\}$. Let $G^* = (V^*, E^*)$. Lemma 3.1 implies that to find F^* we must cover X_0 as cheaply as possible by vertex disjoint paths in G^* all of whose endpoints are in $V^* - X_0$. We note that L2.1(f) implies

$$|V^*| \leqq 30n\, e^{-d/10} \log n = o(n) \tag{3.3}$$

and that L2.1 implies that G^* is a forest. (Every edge of G^* contains a member of X_0 and so by L2.1(d) there are no cycles of size 2λ or less. But a larger cycle will contradict L2.1(c).) We are thus left with a set of path covering problems on trees. We can solve these in linear time using the dynamic programming approach of Bern, Lawler and Wong [2].

Let T be a component of G^* and let $r \in T$ be chosen arbitrarily as a root. Let

$$\Phi = \{X \subseteq E(T): X \text{ induces a set of vertex disjoint paths}\}.$$

For $X \in \Phi$ let $I(X) = \{$internal nodes of the paths of $X\}$ and $O(X) = \{$endpoints of the paths of $X\}$ and $V(X) = I(X) \cup O(X)$. Let

$$\Phi(T, r) = \{X \in \Phi: \text{(i) } X_0 - \{r\} \subseteq I(X), \text{(ii) } V(T) - X_0 \supseteq O(X) - \{r\}\}$$

and

$$\Phi_i(T, r) = \{X \in \Phi(T, r): r \in A_i(X)\}, \qquad i = 1, 2, 3$$

where $A_1(X) = V(T) - V(X)$, $A_2(X) = I(X)$ and $A_3(X) = O(X)$. Assume now that $\Phi(T, r)$ and the $\Phi_i(T, r)$ are ordered by total weight, i.e., if $X, X' \in \Phi(T, r)$, then $X \leqq X' \leftrightarrow c(X) \leqq c(X')$.

Then

$$\Psi(T, r) = \begin{cases} \min \Phi(T, r) & \text{if } r \notin X_0, \\ \min \Phi_2(T, r) & \text{if } r \in X_0, \end{cases}$$

is the answer to our problem—the min operator breaks ties arbitrarily. Now if (T_1, r_1), (T_2, r_2) are vertex disjoint rooted subtrees of (T, r) then we define a rooted tree $(T_1 \circ T_2, r_1)$ where $V(T_1 \circ T_2) = V(T_1) \cup V(T_2) \cup \{r\}$ and $E(T_1 \circ T_2) = E(T_1) \cup E(T_2) \cup \{e\}$ where $e = \{r_1, r_2\}$. (T, r) can be constructed from a set of single vertices by $|V(T)| - 1$ applications of $\circ$.

All we have to demonstrate is that if we know $\min \Phi_i(T_j, r_j)$ for $i = 1, 2, 3$ and $j = 1, 2$ then we can compute $\min \Phi_i(T_1 \circ T_2, r_1)$ in $O(1)$ time. Let

$$\Psi_i = \min \{Z \in \Phi_i(T_1 \circ T_2, r_1) : e \notin Z\}$$

and

$$\Psi_i' = \min \{Z \in \Phi_i(T_1 \circ T_2, r_1) : e \in Z\}.$$

Then $\min \Phi_i(T_1 \circ T_2, r_1) = \min \{\Psi_i, \Psi_i'\}$ and

$$\Psi_i = (\min \Phi_i(T_1, r_1)) \cup \Psi(T_2, r_2), \qquad i = 1, 2, 3,$$

$$\Psi_1' = \varnothing,$$

$$\Psi_2' = (\min \Phi_3(T_1, r_1)) \cup \{e\} \cup Y$$

where

$$Y = \begin{cases} \min \{\min \Phi_1(T_2, r_2), \min \Phi_3(T_2, r_2)\} & \text{if } r_2 \notin X_0, \\ \min \Phi_3(T_2, r_2) & \text{if } r_2 \in X_0, \end{cases}$$

and

$$\Psi_3' = (\min \Phi_1(T_1, r_1)) \cup \{e\} \cup Y.$$

To make the implied operations obviously executable in constant time we keep the minimal weights $c(\Phi_i(T_j, r_j))$ plus flags indicating how they were constructed and then work backwards to construct the optimal solution. □

Let $X_1 = \{\text{interior vertices of paths of } F^*\}$.

LEMMA 3.3. *If* L2.1 ∩ L2.2 *occurs, then* $|X_1| = O(n e^{-d/10})$. □

4. Phase 2 of ALGT. Assuming that L2.1 ∩ L2.2 occurs and F^* has been constructed we apply a modification of HAM of [4] in order to find a tour containing F^* which only uses additional edges from E_0 and hence satisfies (3.2). We remove edges in $E_0 - F^*$ which are incident with X_1. Thus let

$$H = (V_n, (E_0 - \{e \in E_0 : e \cap X_1 \neq \varnothing\}) \cup F^*).$$

Any hamilton cycle of H will be a tour of length LB and optimal, if one exists.

Phase 2 of ALGT. We first check (see Lemma 4.3) that H is connected and $\delta(H) \geqq 2$. If either condition fails we terminate unsuccessfully. Let now F^* consist of paths $R_1, R_2, \cdots, R_s$ and let R_i have endpoints a_{2i-1}, a_{2i} and be of length l_i for $i = 1, 2, \cdots, s$ and let $\text{ENDF}^* = \{a_1, a_2, \cdots a_{2s}\}$. The following idea has been used by many authors (e.g. [1], [4], [6], [7], [9], [10], [11]): given a path $P = (v_1, v_2, \cdots, v_k)$ plus an edge $e = (v_k, v_i)$ where $1 \leqq i \leqq k-2$, we can create another path of length $k-1$

by deleting edge $\{v_i, v_{i+1}\}$ and adding e. Thus let

$$\text{ROTATE}(P, e) = (v_1, v_2, \cdots, v_i, v_k, v_{k-1}, \cdots, v_{i+1}), \text{ and}$$

$$\text{NEW}(P, e) = v_{i+1}.$$

v_1 is called the *fixed endpoint*, v_k is called the *rotated endpoint* and e is called the *rotation edge* of the rotation.

The algorithm we describe proceeds by a sequence of stages. At the beginning of the kth stage we have a path P_k of length k, with endpoints $w_0 \leqq w_1$. We try to extend P_k from w_1. If we fail but $\{w_0, w_1\} \in E(H)$ then connectivity tells us that we can find a longer path. Failing this, we do a sequence of rotations with w_0 as fixed endpoint which creates new paths that we can try to extend or close. We apply the same construction to all these paths and so on until either we have succeeded in obtaining a path of length $k+1$ or we have run out of paths to rotate. We then take this set of paths and treat each of them like P_k but using w_0 as the first rotated endpoint.

We construct our sequence of paths in a "depth-first" manner. Suppose the "current" path is Q. One end u will be kept fixed. Suppose its other end v has neighbours $x_1, x_2, \cdots, x_p$ where $x_1 \in Q$. We replace Q by ROTATE$(Q, \{v, x_1\})$ and continue with this "new" Q *before* considering x_2 and and "old" Q. This will be done after backtracking. We also place a limit on the depth of the rotation sequence before backtracking. This is one way of keeping the algorithm polynomial, but its most important role is in the proof that HAM a.a. succeeds. We now give a formal description:

```
ALGORITHM HAM.
    begin
      let P_0 be the degenerate path consisting of v_1 = min (V_n − X_1) alone;
      k := 0;
L0    begin [stage k begins here]
        longerpathfound := false;
        let P_k have endpoints w_0, w_1 where w_0 ≦ w_1;
        storepaths := true;          [When this variable is true the new paths
                                     [generated by rotations during SEARCH
                                     [are stored for later.
        END_k := {w_1}               [We keep track of the endpoints of some
                                     [paths.
        SEARCH(P_k, w_0, P);         [Do rotations with w_0 as fixed endpoint
        if longerpathfound then      [i.e. if SEARCH has found a path
                                     [longer than P_k
        begin
L1        k := |P| = the number of edges in P;
          P_k := P;
          goto L0
        end else
        storepaths := false;         [No need to store paths now
        for w ∈ END_k do             [SEARCH(P_k, w_0, P) constructs a set of
                                     [paths {P(w_0, w): w ∈ END_k} where
                                     [P(w_0, w) has endpoints w_0 and w.
        begin
          SEARCH(P(w_0, w), w, P);
          if longerpathfound then goto L1
        end;
```

```
    terminate unsuccessfully   [successful termination with a hamilton
                               [cycle containing F* occurs in SEARCH
end;
procedure SEARCH(Q, u, P);
begin
  let v be the endpoint of Q other than u;
  if v = a_j ∈ ENDF* and Q ⋣ R_⌈j/2⌉ then
  begin
    Q := Q + R_⌈j/2⌉;          [catenation of paths
    longerpathfound := true
  end else
  DFS(u, −, 0)
end;
procedure DFS(v, y, t);        [vertex y is such that
                               [ROTATE(Q, {v, y}) reverses the
                               [rotation made immediately prior to
                               [this call of DFS.
S1:
  if t > T = [log n/(log d − log log d)] + 1 then return else
  begin
    Suppose that the edges incident with v, not contained in Q, and not
    incident with ENDF* are {v, x_1}, · · · {v, x_p} where x_1 < x_2 < · · · < x_{p−1}
    and x_p = y;
    if ∃i such that x_i ∉ Q then
    begin
      j := min {i: x_i ∉ Q}
      P := Q + {v, x_j}; {extension]
      longerpathfound := true
    end else
    if u ∈ {x_1, · · · , x_{p−1}} then
    begin                      [cycle extension
      let C be the cycle Q + {u, v};
      if C is a hamiltonian cycle then terminate HAM successfully
      else
      begin
        starting from u, let a be the first vertex along Q which is
        adjacent to some vertex not in C; let b be the lowest
        numbered neighbour of a not in C and let a_1 and a_2 be the
        neighbours of a on C where a_1 < a_2, then
        if {a, a_1} ∈ F* then a_1 := a_2;
        P := C + {a, b} − {a, a_1}; longerpathfound := true
      end
    end else
    for i = 1 to p − 1 do      [x_i ∈ Q − u
S2:
    if not longerpathfound then
    begin
      Q := ROTATE(Q, {v, x_i}); v' := NEW(Q, {v, x_i});
      if storepaths and v' ∉ END_k then
      begin
```

```
            END_k := END_k ∪ {v'};
            P(w_0, v') := Q
          end;
          DFS(v', x_i, t+1)
        end;
      end
      if t > 0 then Q := ROTATE(Q, {v, y})[backtrack to the 'parent path'
  end;
```

The running time of HAM will be estimated later. We now introduce some notation used in the analysis of HAM. Suppose that HAM terminates unsuccessfully in stage k. Let

$$\mathrm{END}(H)=\{v\in \mathrm{END}_k\colon P(w_0, v) \text{ is obtained from } P_k \text{ by } t \text{ rotations, } 0\leqq t\leqq T-1\}.$$

For $v\in \mathrm{END}(H)$ let

$$\mathrm{END}(H, v)=\{x\colon \nexists \text{ a path } Q \text{ with } v, x \text{ as endpoints which is obtained from } P(w_0, v) \text{ by } t \text{ rotations with } v \text{ as fixed endpoint, where } 0\leqq t\leqq T-1\}.$$

We note that,

(4.1) H cannot contain an edge $\{x, y\}$ where $x\in \mathrm{END}(H)$ and $y\in \mathrm{END}(H, x)$.

We must of course show that HAM a.a. succeeds and (4.1) is a simple yet key observation. We see (Lemmas 4.1–4.5) that if HAM fails then $|\mathrm{END}(H)|\geqq n/16$ and $|\mathrm{END}(H, v)|\geqq n/16$ for $v\in \mathrm{END}(H)$ a.a. Given (4.1) we can see that this is unlikely. However the edges referred to in (4.1) are heavily conditioned and we use an argument of Fenner and Frieze [6] to deal with this problem.

The next lemma yields some important structural properties of H. It states that sets of vertices disjoint from X_0 have "large" neighbourhoods. This enables us to deduce that the number of path endpoints grows quickly from iteration to iteration.

LEMMA 4.1. *Assuming only that $d\to\infty$, the following properties hold a.a.*

(a) $S\subseteq V_n-X_0$, $|S|\leqq n/d$ *implies* $|N(S, G_0)|\geqq d|S|/6$.

(b) $S\subseteq V_n-X_0$, $n/d\leqq|S|\leqq n/2$ *implies* $|N(S, G_0)|\geqq|S|/2$.

Proof. (a) A similar result has been proved in several other papers but with different parameters, so we only give an outline here. (See [4], [9].) First consider the case where $s=|S|\leqq s_0=n/d^5$. By considering $N(S, G_0)$ we see that if (a) fails then there exists a set T, $\lceil d/2\rceil=t_0\leqq t=|T|\leqq t_1=\lfloor n/d^4\rfloor$ which contains at least $3t/2$ edges. The probability of this event is no more than

$$\sum_{t=t_0}^{t_1}\binom{n}{t}\binom{\binom{t}{2}}{\lceil 3t/2\rceil}p^{\lceil 3t/2\rceil}\leqq\sum_{t=t_0}^{t_1}(te^5d^3/27n)^{t/2}=o(1).$$

For $|S|\geqq s_0$ we ignore the condition $s\subseteq V_n-X_0$. The probability that S exists is no more than

$$\begin{aligned}&\sum_{s=s_0}^{n/d}\binom{n}{s}\sum_{t=0}^{\lfloor sd/6\rfloor}\binom{n-s}{t}(1-(1-p)^s)^t(1-p)^{s(n-s-t)}\\&\quad\leqq c_8\sum_{s=s_0}^{n/d}\left(\frac{ne}{s}\right)^s\left(\frac{ne}{sd/6}\right)^{sd/6}(sp)^{sd/6}e^{-s(n-s-sd/6)p}\\&\quad\leqq c_8\sum_{s=s_0}^{n/d}\left(\left(\frac{ne}{s}\right)\cdot(6e)^{d/6}\cdot e^{-d}\cdot e\cdot e^{d/6}\right)^s\\&\quad=o(1).\end{aligned}$$

(b) The probability that an S exists contradicting (b) is no more than

$$\sum_{s=n/d}^{n/2} \binom{n}{s}\binom{n}{s/2}(1-p)^{s(n-3s/2)} = o(1)$$

as the largest term is $O((8d^{3/2}e^{-d/4})^{n/d})$. □

We can now "translate" these results to H:

LEMMA 4.2. *If* L2.1 ∩ L2.2 ∩ L4.1 *occurs, then for n large*

(a) $\delta(H) \geqq 2$,

(b) $S \subseteq V_n - X_1, |S| \leqq n/d$ *implies* $|N(S, H)| \geqq d|S|/7$ (X_1 *is defined prior to Lemma* 3.3),

(c) $S \subseteq V_n - X_1, |S| \geqq n/d$ *implies* $|N(S, H)| \geqq n/7 - |S|$,

(d) H *is connected.*

Proof. Assume L2.1 ∩ L2.2 ∩ L4.1 occurs. Each vertex in X_1 has degree 2 in H. Each vertex in $V_n - X_1$ has degree at least $d/2$ in G_0. But from L2.1(c)

(4.2) no vertex is adjacent to λ or more vertices in X_1.

(a) This follows immediately.

(b) This follows from L4.1(a) and (4.2).

($X_1 \supseteq X_0$ and so $|N(S, H)| \geqq |N(S, G_0)| - |N(S, G_0) \cap X_1| \geqq d|S|/6 - \lambda|S|$).

(c) This follows from (b) by taking $S' \subseteq S$ of size $\lfloor n/d \rfloor$ and using

$$|N(S, H)| \geqq |N(S', H)| - |S - S'|.$$

(d) Suppose there exists a partition of V_n into 2 sets $A, B, |A| \leqq |B|$ such that $N(A, H) = \emptyset$. Now clearly $A \not\subseteq X_1$. If $|A - X_1| \leqq n/d$ then (b) and (4.2) with $S = A - X_1$ yield the contradiction, $d|A - X_1|/7 \leqq |A \cap X_1| \leqq \lambda|A - X_1|$. If $|A - X_1| > n/d$ we can use Lemma 4.1(b) and Lemma 3.3. □

We now consider what happens if HAM terminates unsuccessfully in stage k of Phase 2.

Consider P_k, the initial path in stage k. It is the final path in a sequence $P^{(0)}, P^{(1)}, \cdots, P^{(M)} = P_k$ where $P^{(i+1)}$ is obtained from $P^{(i)}$ by a single extension, cycle extension or rotation.

Let $W(H) = \{$edges in $P^{(i)}$, $i = 1, \cdots, M\} \cup \{\{u, v\}$: HAM executes a cycle extension on a path with endpoints u and $v\}$.

Let $X \subseteq E_0$ be *deletable* if:

(4.3a) No edge of X is incident with a vertex of degree $\leqq d/2 + 1$ in G_0;

(4.3b) No pair of edges of X are incident with a common vertex;

(4.3c) $X \cap W(H) = \emptyset$.

The colouring argument of [6] essentially shows that there is a.a. a deletable set X of size $\approx d/2$. Being deletable means that HAM can ignore it up to the last stage and follow the same execution path. We can thus use (4.1) because X is now "unconditioned"—this is only an intuitive interpretation of what follows.

For $X \subseteq E_0$ we define H_X to be the graph obtained from $(V_n, E(K_n) - X)$ by applying Phase 1 but using $E_0 - X$ everywhere E_0 has been used previously.

Remark 4.1. For later purposes it is convenient to assume that the check for L2.1 ∩ L2.2 is not carried out in this case and that if $X = \emptyset$ then $H_X = G_0$. Thus H_X is always defined but sometimes H itself may not be. We note

LEMMA 4.3. *If* L2.1 ∩ L2.2 *occurs and X is deletable, then*

(a) $E(H_X) = E(H) - X$.

(b) *If HAM terminates unsuccessfully in stage k on input H, then HAM will also terminate unsuccessfully in stage k on* H_X.

Proof. (a) is clear and for (b) note that HAM will actually generate P_k at the start of stage k via the same sequence $P^{(i)}$, $i=1,\cdots,M$, as for H. □

We also note that similar calculations to those used for Lemma 2.1(a) yield

LEMMA 4.4. $|\{v\in V_n: d_0(v)\leqq d/2+1\}|\leqq ne^{-d/10}$ *a.a.* □

We now show that failure a.a coincides with a large number of endpoints:

LEMMA 4.5. *Suppose* L2.1 ∩ L2.2 ∩ L4.2 ∩ L4.4 *occurs and HAM terminates unsuccessfully in stage k on H. Suppose* $X\subseteq E_0$ *is deletable. then for n large*

$$|\mathrm{END}(H_X)|\geqq n/16, \tag{4.4}$$

$$|\mathrm{END}(H_X,x)|\geqq n/16 \quad \textit{for } x\in\mathrm{END}(H_X). \tag{4.5}$$

Proof. Assume L2.1 ∩ L2.2 ∩ L4.2 ∩ L4.4 occurs. As noted in Lemma 4.3, HAM will generate the same P_k at the start of stage k as for H. HAM will fail to find a path of length $k+1$ in H_X because HAM failed on H. Suppose P_k has endpoints w_0 and w_1. Note that $w_0, w_1\notin X_1$. Let $S_t=\{v:$ ∄ a path Q_s, $s=s(v)$, with endpoints w_0 and v, such that SEARCH (P_k, w_0, P) constructs Q_s using t rotations$\}$. The rotations above are, of course, concerned with the execution of HAM on H_X.

We fix one value $s(v)$ for each v. We prove (4.4) by showing that

$$\left|\bigcup_{i=0}^{T-1} S_t\right|\geqq\frac{n}{16}. \tag{4.6}$$

To do this we show first that

$$|S_t|\leqq\frac{n}{d} \quad \text{implies } |S_{t+1}|\geqq\frac{d|S_t|}{15}. \tag{4.7}$$

Note that $|S_0|=1$ and let $t\geqq 0$ be such that $|S_t|\leqq n/d$. We use separate arguments for $t\leqq 20$ and $t>20$. Suppose first that $t\leqq 20$.

Consider now pairs (v, w) where $v\in S_t$ and $w\in W(v)=N(\{v\}, H_X)$. If $\{v, w\}$ is not an edge of $Q_{s(v)}$ let $x=x(v,w)=\mathrm{NEW}(Q_{s(v)},\{v,w\})$ and let $x=v$ otherwise. Let $Z=S_0\cup S_1\cup\cdots\cup S_t$. If $w\notin$ ENDF* and $\{v,w\}\notin Q_{s(v)}$ then $x\in S_{t+1}$. Thus

$$\begin{aligned}|S_{t+1}|&=|\{x=x(v,w): v\in S_t, \{v,w\}\notin Q_{s(v)} \text{ and } w\in W(v)-\mathrm{ENDF}^*\}|\\ &\geqq|\{x=x(v,w): v\in S_t, w\in W(v)\}|-\lambda|S_t|\end{aligned} \tag{4.8}$$

using the fact that for each $v\in S_t$ there are at most $\lambda-1$ edges incident with v and a vertex of ENDF* and there is the final edge of $Q_{s(v)}$. Now each $v\in S_t$ is adjacent to at most t vertices w for which $x=x(v,w)$ satisfies $\{w,x\}\notin P_k$ since $Q_{s(v)}$ contains at most t edges not in P_k. For each $w\in N_t=N(S_t,H_X)$ choose $v=v(w)\in S_t$ such that $\{v,w\}\in E(H_x)$ and then let $x(w)=x(v,w)$.

It follows from the above that

$$|\{w\in N_t: \{w,x(w)\}\notin P_k\}|\leqq t|S_t|. \tag{4.9a}$$

Furthermore

$$\begin{aligned}&\text{There can be at most 2 distinct vertices } w, w'\in N_t \text{ such that}\\ &x(w)=x(w') \text{ and } \{x,w\},\{x,w'\}\in P_k.\end{aligned} \tag{4.9b}$$

Thus, using (4.9)

$$|N_t| \leqq 2|\{x(w): w \in N_t \text{ and } \{x, x(w)\} \in P_k\}| + t|S_t|.$$

Hence, using (4.8) we have

$$\begin{aligned} |S_{t+1}| &\geqq |N_t|/2 - (\lambda + t/2)|S_t| \\ &\geqq (d/14 - (\lambda + t/2))|S_t| \end{aligned} \tag{4.10}$$

and (4.7) follows for $t \leqq 20$, in fact for $t \leqq \min\{d/105 - 2\lambda, T-1\}$. Suppose now that $T > t \geqq 21$. Let $X_W = \{x(w): w \in N_t\}$ and for $x \in X_W$ let $W^{-1}(x) = \{w \in N_t: x = x(w)\}$ and $S(x) = \{v(w): w \in W^{-1}(x)\}$. Let

$$Y = \bigcup_{\substack{x \in X_W \\ |W^{-1}(x)| \geqq 2}} W^{-1}(x)$$

and let

$$A = \{v \in V_n: v \text{ lies on some cycle of length 8 or less of } G_{2\mu}\}.$$

Then

$$|N_t| \leqq |X_W| + |A| + |Y - A|. \tag{4.11}$$

Now we can write

$$|Y - A| \leqq |Y_1| + |Y_2| + \cdots + |Y_p| \tag{4.12}$$

where

$$Y_i = W^{-1}(x_i) - A, \quad |W^{-1}(x_i)| \geqq 2 \quad \text{for } i = 1, 2, \cdots, p.$$

Let now $Z_i = \{v(w): w \in Y_i\}$. We note that

$$|Z_i| = |Y_i|, \qquad i = 1, 2, \cdots, p, \tag{4.13a}$$

$$|Z_i \cap Z_j| \leqq 1, \qquad 1 \leqq i < j \leqq p. \tag{4.13b}$$

It follows from (4.12) and (4.13) that

$$|Y - A| \leqq \frac{3|S_t|}{2}$$

and so from (4.11) and L2.1(e)

$$|X_W| \geqq |N_t| - 8(\log n)^9 - \frac{3|S_t|}{2}.$$

Using (4.8) we obtain

$$\begin{aligned} |S_{t+1}| &\geqq |X_W| - \lambda|S_t| \\ &\geqq |N_t| - (\lambda + 2)|S_t| \end{aligned}$$

as if $t > 20$, $|S_t| \geqq (\log n/169)^{10}$ by (4.7) for $t \leqq 20$ and the proof of (4.7) is complete.

We deduce that for some $\tau \leqq T-1$, $|S_\tau| \geqq n/d$. Choose a subset $S \subseteq S_\tau$ of size $\lfloor n/d \rfloor$ and apply the argument that proved (4.7) using S in place of S_τ and $S_0 \cup S_1 \cup \cdots \cup S_{\tau-1} \cup S$ in place of Z. We deduce that $|S_{\tau+1}| \geqq d|S|/15$ and (4.4) follows.

To prove (4.5) consider $v \in \mathrm{END}(H_X)$ and redefine $S_t = \{x: \nexists$ a path Q with endpoints v, x such that SEARCH$(P(w_0, v), v, P)$ constructs Q using t rotations$\}$.

Now apply the argument used to prove (4.7), using Q in place of P_k, to prove (4.5). □

We now consider the size of W and the running time of HAM:

LEMMA 4.6. (a) $|W| \leqq 3n \log n/(d \log d)$ *a.a.*

(b) *HAM runs in* $O(n^2(30 \log n)^T \log n/d) = O(n^{3+o(1)})$ *time a.a.*

Proof. Assume L2.1 ∩ L2.2 ∩ L4.2 ∩ L4.4 occurs.

(a) Let $Z = \{v \in V_n - X_0: c(v, w) > 2\mu \text{ for all } w \in X_0\}$. These vertices are not used at all in the construction of F^* and

$$|Z| \geqq n - 30n \log n\, e^{-d/10}. \tag{4.14}$$

At the start of stage k, let $Z_k = Z - P_k$ and $P_k - P_{k-1} = \{v_k\}$. Note that v_k is an endpoint of P_k. Now let

$$\xi_k = \begin{cases} 1 & \text{if } v_k \in Z \text{ and there is an edge of } E_0 \text{ incident with } v_k \text{ and } Z_k, \\ 0 & \text{otherwise} \end{cases}$$

(if there is no stage k, $\xi_k = 0$).

Then

$$\Pr(\xi_k = 1 | v_k \in Z) \geqq 1 - (1-p)^{|Z_k|} \tag{4.15}$$

using the FKG inequality [8], as the conditioning introduced only tells us that the degrees of v_k and the members of Z_k in G_0 are at least $d/2$. Furthermore

$$|W| \leqq \sum_{k=0}^{n-1} \xi_k + 2T \sum_{k=0}^{n-1} (1 - \xi_k). \tag{4.16}$$

Now, using (4.14) and (4.15) we have

$$\begin{aligned} E\left(\sum_{k=0}^{n-1} \xi_k\right) &\geqq \sum_{r=0}^{|Z|-1} (1-(1-p)^r) \\ &\geqq |Z| - \frac{1}{p} \\ &\geqq n - 30n \log n\, e^{-d/10} - n/d \\ &\geqq n - \frac{5n}{4d} \quad \text{for } n \text{ large} \end{aligned}$$

and so

$$E\left(\sum_{k=0}^{n-1} (1 - \xi_k)\right) \leqq \frac{5n}{4d}.$$

As the lower bound in (4.15) holds whatever the algorithm's history up to stage k, we have

$$\sum_{k=0}^{n-1} (1 - \xi_k) \leqq \frac{3n}{2d} \quad \text{a.a.} \tag{4.17}$$

as it would if the ξ_k were independent. (a) now follows from (4.16).

(b) Let us count the number of rotations in each stage. If $\xi_k = 1$ there are none, otherwise, using L2.1(f), each execution of SEARCH requires $O((30 \log n)^T)$ rotations. Now Angluin and Valiant [1] describe a data structure which enables one to do a rotation in $O(\log n)$ time. Thus, using (4.17), the time spent doing rotations is $O(n^2(30 \log n)^T \log n/d)$. This dominates the time for the other operations. □

We complete the proof of Theorem 1.1 by using the colouring argument of Fenner and Frieze [6]. Having generated our costs we colour the edges of E_0 blue and green. $e \in E$ is coloured blue with probability $1-1/n$ and green with probability $1/n$. These colourings are done independently. Let E_{0b} denote the blue edges and E_{0g} the green edges.

We define 2 events:

$$\mathscr{E}_1 = L \cap \{\text{HAM terminates unsuccessfully in Phase 2}\}$$

where $L = \text{L2.1} \cap \text{L2.2} \cap \text{L4.2} \cap \text{L4.4} \cap \text{L4.6}$,

$$\mathscr{E}_2 = \mathscr{E}_1 \cap \{X = E_{0g} \text{ is deletable}\}.$$

We note that if $\mathscr{E}_2$ occurs, then

(4.18) HAM terminates unsuccessfully on H_X, and

(4.19) $|\text{END}(H_X)| \geqq n/16$ and $|\text{END}(H_X, x)| \geqq n/16$ for all $x \in \text{END}(H_X)$.

(4.20) There does not exist $e = \{x, y\} \in X$ such that $x \in \text{END}(H_X)$ and $y \in \text{END}(H_X, x)$.

We prove that

$$\Pr(\mathscr{E}_2 | \mathscr{E}_1) \geqq (1-o(1))(1-1/n)^{(3n \log n/d \log d)+n}, \tag{4.21}$$

$$\Pr(\mathscr{E}_2) \leqq (1-p/n)^{n^2/512}. \tag{4.22}$$

(The restricted version of) Theorem 1.1 follows immediately from (4.21), (4.22) and the fact that $\Pr(L) = 1-o(1)$.

Proof of (4.21). This follows from

$$\Pr(\mathscr{E}_2 | E_i, i = 0, 1, \cdots B(n)-1) \geqq (1-o(1))(1-1/n)^{(3n \log n/d \log d)+n} \tag{4.23}$$

for all $E_0, E_1, \cdots$ compatible with the occurrence of $\mathscr{E}_1$. Once $E_0, E_1, \cdots$ are fixed then (4.23) follows easily from L4.4, L4.6, L2.1(f) and (4.2).

Proof of (4.22). Let $X_{0b} = \{v \in V_n$: degree of v in $(V_n, E_{0b}) \leqq D/2\}$ and for $i \geqq 1$, $E_{ib} = \{e \in E_i;\ e \cap X_{0b} \neq \emptyset\}$. We prove (4.22) by showing that

$$\Pr(\mathscr{E}_2 | E_{ib}, i = 0, 1, \cdots B(n)-1) \leqq (1-p/n)^{n^2/512}. \tag{4.24}$$

We note that if $E_{0b}, E_{1b}, \cdots$ are given then so is H_X. If either of (4.18) or (4.19) do not hold then the probability in (4.24) is zero. So assume that $E_{0b}, E_{1b}, \cdots$ are such that (4.18) and (4.19) hold. Consider next the set $Y = \{\{x, y\}: x \in \text{END}(H_X)$ and $y \in \text{END}(H_X, x)\}$. Note that $Y \cap E_{ib} = \emptyset$ for $i \geqq 0$. For each $e \in Y$ we have, given $E_{0b}, E_{1b}, \cdots$, $\Pr(e \in E_{0g}) = p/(n-d+1)$ and the events $e \in E_{0g}$ are independent. Clearly

$$\Pr(\mathscr{E}_2 | E_{ib}, i = 0, 1, \cdots B(n)-1) \leqq \Pr((4.20) | E_{ib}, i = 0, 1, \cdots B(n)-1) \leqq (1-p/n)^{|Y|}$$

and (4.24) and the theorem follows. □

5. The general case.

Change of assumptions.

(1) $d = \omega \log\log n \leqq \log n/2$.

(2) $\lambda = \lceil \log n/\log\log n \rceil$.

Before proving Theorem 1.1 for $B(n) = o(n/\log\log n)$ we will indicate the places where the proof we have given so far breaks down.

Problem A. We have no problems with Phase 1 but because now $d < \lambda$ it is possible that we may find some vertices not in X_1 which lose all of their incident zero edges when we construct H. Our solution, essentially, is to absorb these vertices into X_0 beforehand.

Problem B. The estimate for the execution time of HAM has a term $(\log n)^{\log n/\log d}$ in it, which is not always polynomial for $d = \omega \log\log n$.

Problem C. Our final colouring argument relies on $|W|/n = o(d)$. The upper bound for $|W|$ in Lemma 4.6 is not strong enough to prove this.

Having given the problems, we proceed to their solution.

Overview. We define a new set X_0 which takes care of Problem A and we cover it with paths as before. We then make a small change to HAM which will take care of the other 2 problems.

We start with Problem C. Our solution is a little bizarre: given E_0 we randomly partition it into 2 sets $E_{0+} = \{\text{positive edges}\}$ and $E_{0-} = \{\text{negative edges}\}$. If $e \in E_0$ we place it in E_{0+} with probability $\frac{1}{2}$ and in E_{0-} with probability $\frac{1}{2}$. Naturally these placings are done independently of each other and so $(V_n, E_{0+}) = G_{n,p/2}$. We then alter algorithm HAM as follows:

(5.1a) The depth of rotation check S_1 is abandoned,

(5.1b) S_2 is changed to
if $e = \{v, x_i\} \in E_{0+}$ *and not* longerpathfound *and* e has not been used previously as a rotation edge in the *current* execution of SEARCH *then*

Thus, foolishly it seems, we only use edges in E_{0-} to extend paths and not to rotate them. On the other hand, in the final edge-colouring argument, we only colour edges in E_{0-}. Since we make $\leqq n$ cycle extensions we have $|E_0 - \cap W| \leqq 2n$ which gets around the $|W| = o(nd)$ problem as we will see later. One consequence of (5.1b) is that each execution of SEARCH requires no more than $|E_{0+}|$ rotations—each $e \in E_{0+}$ is used at most once to rotate. Now $|E_{0+}|$ is a binomial random variable with expectation $(n-1)d/4$ and so is a.a. less than nd. By Lemma 4.6 the number of stages that require rotations is a.a. $O(n/d)$. Putting this together we obtain the following lemma.

LEMMA 5.1. *HAM runs in* $O(n^3 \log n)$ *time a.a.* □

Thus Problem B is definitely solved, we have intimated that Problem C is solved and we now turn to Problem A: In this part of the paper X_0 will be redefined. So let

$$Y_0 = \{v \in V_n\text{: degree of } v \text{ in } (V_n, E_{0+}) \text{ is no more than } d/4\}.$$

It is convenient to define a set $X_{0,m}$ for positive integer $m \geqq 2$ and use $X_0 = X_{0,7}$ in the algorithm.

Construction of $X_0 = X_{0,m}$. Let Y_0 be as above and $Y_k = Y_0 \cup \{w_1, \cdots, w_k\}$ be defined inductively by $w_k = \min\{w \in V_n - Y_{k-1}$: $\exists$ independent edges $e_1, \cdots, e_m \in E_{2\mu+}$ (where $E_{k+} = E_k - E_{0-}$ for $k \geqq 0$) and edges $f_1, \cdots, f_m \in E_{2\mu+}$ such that

$$e_i \cap Y_{k-1} \neq \emptyset, \qquad i = 1, 2, \cdots, m,$$

$$w_k \in f_i, \qquad i = 1, 2, \cdots, m,$$

$$|e_i \cap f_i| = 1, \qquad i = 1, 2, \cdots, m\}.$$

The process continues until $k = s+1$ is such that no w_k exists. We take $X_0 = Y_s$. It is important to note that

(5.2) The set X_0 constructed is actually independent of the numbering of the vertices of $G_{2\mu+}$.

Before proving properties of X_0 we motivate its definition. Having constructed X_0 we shall carry out Phase 1 as previously and cover X_0 with a set of vertex disjoint paths F^*—we use the edges in E_{0-} in this phase as normal. Consider now a vertex v that is not an interior vertex of one of these paths and consider the edges of E_{0+} which are incident with v and with vertices on F^*. Let these edges be $\{v, w_i\}$, $i=1,2,\cdots,k$ and let w_i be incident with edges a_i, b_i where $a_i=b_i$ if w_i is an endpoint. As $v\notin X_0$ the set $\{a_1, b_1, \cdots, a_k, b_k\}$ does not contain a subset of seven independent edges. Furthermore it is easy to see that $G_{2\mu+}$ a.a. contains no pair of triangles or quadrilaterals with a common vertex. (The expected number of such pairs is $O((\log n)^8/n)$.)

Defining X_1 as prior to Lemma 3.3 we have

(5.3) $v\notin X_1$ is adjacent in $G_{2\mu+}$ to at most 7 vertices of F^*.

Thus our definition of X_0 will solve Problem A provided the equivalent of Lemma 2.1 can be shown, so that we can compute F^* quickly. For (5.3) tells us that an endpoint not in X_1 has many ($\geqq d/4-7$) edges in E_{0+} that can be used for rotations, if necessary.

We ought to be sure that X_0 can be computed quickly. But this is no problem as given Y_k, to test whether $w\notin Y_k$ can be added we have to look at the edges of $G_{2\mu+}$ which are incident with neighbours of w and contain vertices of Y_k and see whether they contain seven independent edges. We can clearly do all this within the $O(n^3\log n)$ time bound.

We now give the equivalent of Lemma 2.1. Its proof is rather long and technical and we advise the reader to omit it at first reading.

LEMMA 5.2.

(a) $\Pr(|X_0|\geqq 16ne^{-d/10})=o(e^{-d\log n})$.

(b) *$Z\subseteq V_n$, $|Z|\leqq 3\lambda$ and $|Z\cap X_0|\geqq\lambda$ implies $G_{2\mu+}[Z]$ is not connected a.a.*

(c) *C is a cycle in $G_{2\mu+}$ implies $|C\cap X_0|\leqq|C|/10$ a.a.*

Proof. It is convenient to start by proving:

(5.4a) $\Pr(|Y_0|\geqq ne^{-d/10})=o(e^{-d\log n})$,

(5.4b) $S\subseteq V_n$, $|S|\leqq N/(\log n)^6$ implies that S contains less than $5|S|/4$ edges of $G_{2\mu+}$ a.a.,

(5.4c) $\Pr(\dashv S\subseteq V_n: ne^{-d}\leqq|S|\leqq n/(\log n)^6$ and S contains at least $5|S|/4$ edges of $G_{2\mu+})=o(e^{-d\log n})$,

(5.4d) $S\subseteq V_n$, $|S|\leqq 4\lambda$ implies that S contains at most $|S|+4$ edges of $G_{2\mu+}$ a.a.

Proof of (5.4a). We use the Markov inequality

$$\Pr(X\geqq a)\leqq E(f(X))/f(a) \tag{5.5}$$

for any nonnegative random variable X, any $a\geqq 0$ and any nonnegative function f (see, for example, Grimmett and Stirzaker [11, pp. 185–186]). We take $f(X)=\max\{0, X(X-1)(X-2)\cdots(X-t+1)\}$ where $t=\lceil(\log n)^2\rceil$ and $a=ne^{-d/10}$. Note that $f(|Y_0|)$ is the number of t-tuples of elements of Y_0. Hence

$$E(f(|Y_0|)\leqq t!\binom{n}{t}S(n-t,(2\mu+1)p,d/2)^t$$

$$\leqq n^t e^{-.15dt}.$$

The result goes through with a lot to spare on using (5.5).

Proof of (5.4b). The probability that S exists contradicting (5.4b) is no more than

$$\sum_{k=4}^{n/(\log n)^6}\binom{n}{k}\binom{\binom{k}{2}}{\lceil 5k/4\rceil}((2\mu+1)p)^{(5k/4)}$$
$$\leqq \sum_{k=4}^{n/(\log n)^6}((k/n)^{1/4}(\log n)^{5/4}(6/5)^{5/4}e^{9/4})^k$$
$$= O((\log n)^5/n).$$

Proof of (5.4c). The calculations that we have done support this.

Proof of (5.4d). The probability that S exists contradicting (5.4c) is no more than

$$\sum_{k=5}^{4\lambda}\binom{n}{k}\binom{\binom{k}{2}}{k+5}((2\mu+1)p)^{\lceil k+5\rceil}=o(1)$$

after using $(\log n)^\lambda \leqq n \log n$.

Proof proper. (a) We shall prove that $\Pr(|X_{0,m}|\geqq 16ne^{-d/10})=o(e^{-d\log n})$ for all constants $m\geqq 2$. Define $Y_0'=Y_0$ and Y_k' inductively by $Y_k'=Y_{k-1}'\cup e_1\cup f_1\cup\cdots\cup e_m\cup f_m$ where the e_i, f_i are those associated with w_k. We let $E_k'=\{e\in E_{2\mu}: e\subseteq Y_k'\}$. Then

(5.6a) $$|Y_0|+k\leqq|Y_k'|\leqq|Y_0|+(m+1)k, \text{ and}$$

(5.6b) $$|E_k'|\geqq(2m/(m+1))(|Y_k'|-|Y_0|)$$

follow easily by induction.

Suppose then that $|X_0|\geqq 16ne^{-d/10}$. Then since, by (5.4a), $|Y_0|\leqq ne^{-d/10}$ with probability $1-o(e^{-d\log n})$we can assume that there exists $k\geqq 15ne^{-d/10}$ such that $|Y_k|=16ne^{-d/10}$. But then (5.6) implies

$$\frac{|Y_k'|}{|E_k'|}\leqq((m+1)/2m)(|Y_k'|/(|Y_k'|-|Y_0|)$$
$$\leqq((m+1)/2m)(16/15)$$
$$\leqq 4/5 \quad \text{as } m\geqq 2.$$

But this happens with probability $o(e^{-d\log n})$—(5.4c).

(b) If $G_{2\mu+}$ is connected, and [5] implies that it a.a. is, then the failure of (b) implies the existence of a set $Z\subseteq V_n$, $E\subseteq E_{2\mu+}$ satisfying

(i) (Z, E) is a tree,
(ii) $|Z\cap X_0|\geqq\lambda$, and
(iii) $|Z|=3\lambda$.

Let us first note that

(5.7) There a.a. exist no Z, E satisfying (i), (ii)′ and (iii)

where

(ii)′ $|Z\cap Y_0|\geqq\lambda/14$.

The proof of this follows the lines of the proof of Lemma 2.1(c). Now the probability π_0 that Z exists contradicting (b) is no more than

$$\binom{n}{3\lambda}\binom{3\lambda}{\lambda}(3\lambda)^{3\lambda-2}\pi_1+\Pr(\bar{L})$$

where $\pi_1 = \max\{\Pr(A_T)\}$ and $L = \cap$(the events that have been proved so far to a.a. occur) and A_T is the event $\{\{1, 2, \cdots, \lambda\} \subseteq X_0$, some fixed tree T with vertices $\{1, 2, \cdots, 3\lambda\}$ exists in $G_{2\mu}\} \cap L$. We show that

$$\pi_1 \leqq e^{-\alpha\lambda d}((2\mu+1)p)^{3\lambda-1} \quad \text{for some constant } \alpha > 0. \tag{5.8}$$

Hence

$$\begin{aligned}\pi_0 &\leqq (ne/3\lambda)^{3\lambda} 3^{3\lambda} (3\lambda)^{3\lambda-2} (3\log n/n)^{3\lambda-1} e^{-\alpha\lambda d} \\ &= O(e^{-\alpha\omega\lambda \log\log n/2}) \\ &= o(1).\end{aligned}$$

It remains to prove (5.8). We do this by demonstrating that if A_T occurs then so does some other event that we can show is unlikely. Assume A_T occurs.

Now T contains at most $6\lambda/7$ vertices of degree $\geqq 7$ in T and hence at least $\lambda/7$ vertices of X_0 which have degree 6 or less in T. Now $|T \cap Y_0| \leqq \lambda/14$ by (5.7) and so there are at least $\lambda/14$ vertices in $X_0 - Y_0$ which have degree 6 or less in T. Let these vertices be denoted by T_0.

Now for $v \in T_0$ there are edges $e_i(v), f_i(v)$, $i = 1, 2, \cdots, 7$ such that the $e_i(v)$'s are independent and meet X_0, the $f_i(v)$'s meet v and $|e_i(v) \cap f_i(v)| = 1$ for $i = 1, 2, \cdots, 7$. But at most 6 out of $f_1(v), \cdots, f_7(v)$ can be edges of T. Assume then that $f_1(v)$ is not an edge of T. At most 4 edges of the set $\{f_1(v): v \in T_0\}$ are incident with 2 vertices of T (5.4d) and so at least $\lambda/14 - 4$ vertices in T_0 are such that $f_1(v)$ is incident with a vertex outside T. Let T_1 denote these vertices of T.

Let $K = \{e_1(v): v \in T_1\}$. (5.4d) implies that at most 8 of these edges intersect another edge in K. Thus by removing at most 8 edges we are left with an independent set of edges $K_1 \subseteq K$, $|K_1| \geqq \lambda/14 - 12$. Now (5.4d) implies that there are at most 4 vertices not in T which are adjacent to 2 or more vertices of T and no vertex not in T is adjacent to 6 or more members of T. Also at most 4 edges not incident with T can have both endpoints adjacent to members of T. Thus by removing at most 24 edges from K_1 we obtain a set $K_2 \subseteq K_1$, $|K_2| \geqq \lambda/15$ such that $e \in K_2$ implies (i) $e \cap X_0 \neq \emptyset$, (ii) $e \cap T = \emptyset$, and (iii) exactly one endpoint of e is adjacent to T.

Let now $Z_1 = \{v \notin X_0: \dagger w \in T$ and $e \in K_2$ such that $\{v, w\} \in E_{2\mu+}$ and $v \in e\}$ and $Z_2 = (\cup_{e \in K_2} e) - Z_1$. (5.4d) implies that

(5.9) no vertex in $V' = V_n - (T \cup Z_1)$ is within distance 2 in $G_{2\mu+}$ of 5 or more members of $T \cup Z_1$.

Let $G' = G_{2\mu+}[V']$. We claim that if we apply the process used to construct X_0 to G', taking $m = 3$, then $X_{0,7}(G) - (T \cup Z_1) \subseteq X_{0,3}(G')$, where we have given the $X_{0,m}$ an extra parameter to indicate from which graph they are constructed.

If not let $X_{0,7}(G) = Y_0(G) \cup \{w_1, w_2, \cdots, w_s\}$ where the w_i are in the order in which they were selected and let $t = \min\{j: w_j \notin T \cup Z_1 \cup X_{0,3}(G')\}$. Let e_i, f_i, $i = 1, 2, \cdots, 7$ be the edges used to place w_t in $X_0(G)$. (5.9) implies that at least 3 of these can be used to place w_t in $X_0(G')$, a contradiction.

We have thus proved that if T exists, L occurs and $|X_0 \cap T| \geqq \lambda$ then there exist sets Z_1, Z_2 with

(5.10a) $|Z_1| \leqq |Z_2| \leqq 6\lambda$,

(5.10b) $\lambda/15 \leqq |Z_2|$,

(5.10c) $Z_2 \subseteq X_{0,3}(G')$,

(5.10d) Each vertex of Z_1 is adjacent to a vertex of T,

(5.10e) Each vertex of Z_2 is adjacent to a vertex of $T \cup Z_1$.

Note that once T, Z_1, Z_2 are given the event (5.10c) is independent of (5.10d) and (5.10e). Thus

$$\pi_1 \leqq \Pr(T \text{ exists and } \nexists\, Z_1, Z_2 \text{ satisfying (5.10)})$$
$$\leqq \sum_{z_1=\lceil \lambda/15\rceil}^{6\lambda} \sum_{z_2=z_1}^{6\lambda} \binom{n}{z_1}\binom{n}{z_2}((2\mu+1)p)^{3\lambda-1}(3\lambda(2\mu+1)p)^{z_1}(9\lambda(2\mu+1)p)^{z_2}\pi_2$$

where $\pi_2 = \Pr(\mathrm{LOW} \subseteq X_{0,3}(G'))$ and LOW is the $\lceil \lambda/15\rceil$ lowest numbered vertices of $V(G')$ and so

$$\pi_1 \leqq ((2\mu+1)p)^{3\lambda-1}(\log n)^{24\lambda}\pi_2.$$

Bearing in mind (5.3), we see that by symmetry

$$\Pr(\mathrm{LOW} \subseteq X_{0,3}(G') \mid |X_{0,3}(G')| = n_1) = \binom{n_1}{\lceil \lambda/15\rceil} \Big/ \binom{|V(G')|}{\lceil \lambda/15\rceil} \tag{5.11}$$
$$\leqq (n_1/|V(G')|)^{\lceil \lambda/15\rceil}.$$

Now

$$n - 9\lambda \leqq |V(G')| \leqq n - 3\lambda$$

and so by (a) (with n_1 in place of n) we see that

$$\Pr(|X_{0,3}(G')| \geqq ne^{-d/10}) = o(e^{-d \log n}).$$

This combined with (5.11) yields (5.8) and (b).

The proof of (c) is similar and slightly easier as we have to deal with a cycle rather than an arbitrary tree. □

6. Modifications for the second phase. We must now prove that HAM a.a. succeeds. Let H be defined as before. Assume that HAM fails on H in stage k. Let $W(H)$ be defined as prior to (4.3). Because the edges of E_{0-} are only used for extensions we have

$$|E_{0-} \cap W(H)| \leqq 2n. \tag{6.1}$$

Let $X \subseteq E_{0-}$ be *deletable* if

(6.2a) $X \cap W(H) = \varnothing$.

(6.2b) No edge of X is incident with a vertex which is distance 2 or less from X_0 in $G_{2\mu+}$.

Note that if L4.1 ∩ L5.2 occurs then

(6.3) the number of vertices at distance 2 or less from X_0 in $G_{2\mu+}$ is $O(n(\log n)^2 e^{-d/20}) = o(n)$.

If H_X is defined as in § 4 then Lemmas 4.1 and 4.3 continue to hold, with d replaced by $d/2$.

LEMMA 6.1. *Assume* L4.1 ∩ L5.2 *occurs and HAM terminates in stage k on H. Suppose* $X \subseteq E_{0-}$ *is deletable. Then for n large*

$$|\mathrm{END}(H_X)| \geqq n/67. \tag{6.4}$$

$$|\mathrm{END}(H_X, x)| \geqq n/67 \text{ for } x \in \mathrm{END}(H_X). \tag{6.5}$$

Proof. Assume L4.1 ∩ L5.2 occurs. Let $H_{X+} = (V_n, E(H_X) \cap E_{0+})$. We modify the argument of Posa [17] to show that $Z = \mathrm{END}(H_X)$ has relatively few neighbours in H_{X+}. In fact we show that

$$|N(Z, H_{X+}) - \mathrm{ENDF}^*| \leqq 2|Z| \tag{6.6}$$

(recall that ENDF* is the set of endpoints of the paths in F^*). This follows once we have shown that if $x \notin (Z \cup \text{ENDF}^*)$ is adjacent to some $z \in Z$ then x is adjacent on P_k to some $z' \in Z$. Suppose x, z contradict this statement. As $x \notin Z$ there will not be an execution of DFS(x, ?, ?) during SEARCH(P_k, w_0, P). As $z \in Z$ there will be an execution of DFS(z, ?, ?) and the edge $\{z, x\}$ will be a rotation edge. But at this time the neighbours of x on Q will be its neighbours on P_k. For if a P_k edge gets deleted by a rotation then one of its endpoints is placed in Z. We have thus proved (6.6).

It follows from (5.3) and $Z \cap X_1 = \varnothing$ that

$$|N(Z, G_{0+})| \leqq 9|Z|. \tag{6.7}$$

Applying Lemma 4.1 (with d replaced by $d/2$) shows that $|Z| > 2n/d$. Let $Z' \subseteq Z$ be of size $\lfloor 2n/d \rfloor$. Now

$$\begin{aligned} |N(Z, G_{0+})| &\geqq |N(Z', G_{0+})| - |Z| \\ &\geqq \lfloor 2n/d \rfloor (d/12) - |Z|. \end{aligned}$$

Formula (6.4) follows on using (6.7). Formula (6.5) is proved similarly. □

We can finish the proof using our colouring argument. Having generated our costs and chosen our partition of E_0, we randomly colour the edges in E_{0-} blue with probability $1 - 1/n$ and green with probability $1/n$. Our 2 events are

$$\mathscr{E}_1 = L \cap \{\text{HAM terminates unsuccessfully in Phase 2}\}$$

where $L = \text{L4.1} \cap \text{L5.2}$ and

$$\mathscr{E}_2 = \mathscr{E}_1 \cap \{X = E_{0+g} \text{ is deletable}\}.$$

Proceeding as before we can prove

$$\Pr(\mathscr{E}_2 | \mathscr{E}_1) \geqq (1 - o(1))(1 - 1/n)^{2n} \tag{6.8}$$

and

$$\Pr(\mathscr{E}_2) \leqq (1 - p/2n)^{n^2/9000}. \tag{6.9}$$

(To fix H_X, fix E_{0+}, E_{0-b}, $E_1, \cdots, E_{2\mu}$, and the values of the edges incident with the corresponding X_0.)

Theorem 1.1 follows. □

We should say something about the restriction $B(n) = o(n/\log\log n))$. We could replace ω in the definition of p by a suitably large constant. We have allowed this "constant" to go to infinity, as it makes our inequalities more obvious and only at marginal loss of generality. The problem with smaller values of p is with Phase 1. For suppose that $B(n)$ grows faster than $n/\log\log n$. Heuristic arguments suggest that G^* has $m = ne^{-\alpha d}$ edges (α constant) and average degree growing with m. Thus G^* will almost certainly have a "giant" component, making the construction of F^* (presumably) hard.

On the positive side HAM has no problem, even for $B(n) = \alpha n$, $\alpha > 0$ sufficiently small. We are thus now able to give polynomial time algorithms which construct the large cycles in Fenner and Frieze [6], [7] and Frieze [9]. We report on this in a further paper [10].

Note finally that the method extends naturally to finding k edge disjoint tours of minimum weight, k constant. We optimally cover X_0 with k sets of vertex disjoint paths and apply HAM k times.

REFERENCES

[1] D. ANGLUIN AND L. VALIANT, *Fast probabilistic algorithms for hamilton circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155-193.

[2] M. W. BERN, E. L. LAWLER AND A. L. WONG, *Why certain subgraph computations require only linear time*, Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 117-125.

[3] B. BOLLOBÁS, *Random Graphs*, Academic Press, New York, 1985.

[4] B. BOLLOBÁS, T. I. FENNER AND A. M. FRIEZE, *An algorithm for finding hamilton cycles in random graphs*, Proc. 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 430-439.

[5] P. ERDÖS AND A. RÉNYI, *On random graphs* 1, Publ. Math. Debrecen, 6 (1959), pp. 290-297.

[6] T. I. FENNER AND A. M. FRIEZE, *On the existence of hamilton cycles in a class of random graphs*, Discrete Math., 45 (1983), pp. 301-305.

[7] ———, *Hamilton cycles in random regular graphs*, J. Combin. Theory Ser. B, 37 (1984), pp. 103-112.

[8] C. M. FORTUIN, P. W. KASTELYN AND J. GINIBRE, *Correlation inequalities on some partially ordered sets*, Comm. Math. Phys., 22 (1971), pp. 89-103.

[9] A. M. FRIEZE, *On large matchings and cycles in sparse random graphs*, Discrete Math., 59 (1986), pp. 243-256.

[10] ———, *Finding hamilton cycles in sparse random graphs*, J. Combin. Theory, to appear.

[11] G. GRIMMETT AND D. STIRZAKER, *Probability and Random Processes*, Clarendon Press, Oxford, 1982.

[12] Y. GUREVICH AND S. SHELAH, *Expected computation time for hamiltonian path problem and clique problem*, to appear.

[13] J. H. HALTON AND R. TERADA, *A fast algorithm for the euclidean travelling salesman problem: optimal with probability* 1, this Journal, 11 (1982), pp. 28-46.

[14] R. M. KARP, *Probabilistic analysis of partitioning algorithms for the travelling salesman problem in the plane*, Math. Oper. Res., 2 (1977), pp. 209-224.

[15] ———, *A patching algorithm for the non-symmetric travelling salesman problem*, this Journal, 8 (1979), pp. 561-573.

[16] E. L. LAWLER, J. K. LENSTRA, A. H. G. RINOOY KAN AND D. B. SHMOYS, *The Travelling Salesman Problem*, John Wiley, New York,

[17] L. POSA, *Hamilton circuits in random graphs*, Discrete Math., 14 (1976), pp. 359-364.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987

© 1987 Society for Industrial and Applied Mathematics
008

PADDED LISTS REVISITED*

MICHA HOFRI† AND ALAN G. KONHEIM‡

Abstract. We study a data structure **L** referred to variously as a *padded list, controlled density array* or *sparse table* containing records $\{R_i\}$ each uniquely identified by a key $\{k(R_i)\}$. **L** is required to support the operations Search (**L**, k), Insert (**L**, k) and Delete (**L**, k) to *search, insert* and *delete* a record with key k. To optimize Search (**L**, k), records are stored with their keys in sorted order. If the order of the keys is to be maintained under insertion, records currently in **L** must be moved to free space. To improve the efficiency of Insert (**L**, k), records are stored in a circular buffer with "gaps" so that insertion necessitates moving only records up to the next gap. The array is expanded and contracted during a sequence of insertions and deletions depending upon the current number of gaps. In this paper we assess the amount of work required to insert a sequence of records.

Key words. padded list, controlled density array, sparse table, hashing

AMS(MOS) subject classification. 68P10

1. Introduction and summary. Ordered lists are basic data aggregates in many applications. Such a data structure **L** containing records $\{R_i\}$, each uniquely identified by a key, is required to support the operations:

Search (**L**, k): Search **L** for a record with the key k.
Insert (**L**, k): Insert the record with key k into **L**.
Delete (**L**, k): Delete the record with key k from **L**.

Commonly, such lists are maintained in various tree-based data structures [2], [6], [7], and tend to suffer from relatively substantial implementation-related overhead generated by operations on the list. Recently, several authors [3], [5], [4] suggested a new mode of storage for such lists referred to as *a padded list, controlled density array* or *sparse table*, linear arrays with capacity exceeding the immediate storage requirement. A single (table origin) pointer TO allows the user to regard the array as a circular buffer. To optimize Search (**L**, k), records are stored in **L** with their keys sorted. If the order of the keys is to be maintained under insertion, records currently in the table must be shifted to free space for the record to be inserted. The *gaps* in the table improve the efficiency of Insert (**L**, k); with gaps, insertion will require relocation of records only up to the next gap. When the number of gaps falls below some threshold, the table is deemed too dense and it is *reconfigured*; the table is extended with gaps introduced in some regular fashion. The reverse situation, in which deletions have rendered the table too sparse, is treated similarly by reducing the table size to a more desirable level. The attractiveness of the above schemes is derived from the simplicity of the operations Search (**L**, k), Insert (**L**, k) and Delete (**L**, k) on them. The thrust of this paper is the analytic evaluation of the cost of insertions.

The performance of these schemes is measured in part by the number of *moves* needed to insert a new record. In this paper we analyze the number of moves under a sequence of insertions. This problem was first studied by Franklin [3] (for the padded list) and Melville and Gries [5] (for the controlled density array). The reader is also referred to these papers for implementation details and considerations of the worst

* Received by the editors November 24, 1986; accepted for publication (in revised form) February 10, 1987.

† Computer Science Department, The Technion-Israel Institute of Technology, Haifa, Israel.
‡ Computer Science Department, University of California, Santa Barbara, California 93106.

case behavior of these schemes compared with other data structures. In both of the above papers the authors draw upon results related to open hashing with linear probing. The two processes

key insertion in a padded list, open hashing with linear probing

are similar, but while each "slot" (memory location) is hypothesized to have the same probability of being the target of an insertion in open hashing, the gaps in a padded list introduce local distortions to this uniformity. In [5] the authors recognized this inaccuracy in their presentation, commenting that the two schemes were not entirely equivalent.

The present work is very close in spirit and methodology to that reported on in [4]. There, reconfiguration introduced $m-1$ gaps for each existing record (making the density of records $1/m$ following a reconfiguration), while in a padded list a single gap is introduced for each of $r-1$ records (so that the density of records following a reconfiguration is $1-1/r$. This distinction results surprisingly in significant differences in the analysis.

The analysis we offer is combinatorial; we count the number of insertion sequences (of keys) that give rise to various configurations of the table. By assuming that all insertion sequences of a given collection of keys are equally likely, we are then able to assign a probability to the event $\{M_r(n, N)=m\}$: *inserting a record into a table of size nr presently containing N records requires moving m records.* We obtain the probability generating function of $M_r(n, N)$ for finite tables and its expectation, as well as the limiting formulae

$$\lim_{n\to\infty} E\{M_r(n, n\rho r)\} = \frac{\rho}{2(r-1)(1-\rho)^2} - 1$$

and

$$\lim_{n\to\infty} \Pr\{M_r(n, n\rho r)=m\} = \frac{(1-\rho)r}{1-r+\rho r} \sum_{\substack{i:\ 1\leqq i<\infty \\ ir\geqq m+2}} \frac{1}{ir-1}\binom{ir-1}{i}\left(\frac{(r-1)^{r-1}(1-r+\rho r)}{(\rho r)^r}\right)^i$$

where ρ is the density of records in the list.

A word about the paper's organization: in § 2 we define the *padded list schemata*; the combinatorial analysis is given in §§ 3-5 and applied in § 6 to obtain the distribution and expectation of the number of moves required to insert a record in a padded table; numerical results are given in § 7.

2. Padded list schemata. Records are stored with their keys sorted, say in increasing order, so that searching may be efficiently performed. If records are stored contiguously and the order of the keys is to be maintained under insertion, then existing records must be *moved* to free space for the record to be inserted. The efficiency of insertion will be improved by *padding* the table, storing N records in a table of *size* (or record capacity) $m>N$. A key is assigned to each address in the table by introducing fictitious or *dummy* keys. A dummy key takes the value of the first genuine key "to its right" circularly. We describe the *state of the table* by the vector

$$\bar{y}=(y_0, y_1, \cdots, y_{m-1}), \qquad y_0\leqq y_1\leqq\cdots\leqq y_{m-1}$$

indicating with this notation that the (genuine or dummy) key (with value) y_i is stored at address i for $0\leqq i<m$. The table may be considered as a circular array; address 0 follows address $m-1$ and address calculations are made modulo m. The corresponding

state of memory is denoted by $\bar{z} = (z_0, z_1, \cdots, z_{m-1})$. A *table origin pointer* TO translates between the actual address in memory and the logical address in the table

$$\text{actual address} = (\text{TO} + \text{logical address}) \text{ (modulo } m)$$

so that the table and memory states are related by

$$y_i = z_{(i+\text{TO})(\text{modulo } m)}, \qquad 0 \leqq i < m.$$

The number of genuine keys in the table $\bar{y}$ will be denoted by $\|\bar{y}\|$ and the size of $\bar{y}$ by $|\bar{y}|$. The *skeleton* of the table $\bar{y} = (y_0, y_1, \cdots, y_{m-1})$ is the vector

$$\sigma = (\sigma_0, \sigma_1, \cdots, \sigma_{m-1})$$

defined by

$$\sigma_i = \begin{cases} 1 & \text{if } y_i \text{ is a genuine key,} \\ 0 & \text{if } y_i \text{ is a dummy key.} \end{cases}$$

The building of a table by means of the insertion of keys is specified by two sequences of positive integers

$$\{n_i : 1 \leqq i < \infty\}, \qquad \{r_i : 1 \leqq i < \infty\}$$

satisfying

(2.1a) $$n_i(r_i - 1) < n_{i+1}(r_{i+1} - 1) \leqq n_i r_i + 1, \qquad 1 \leqq i < \infty,$$

(2.1b) $$2 \leqq r_i < \infty, \qquad 1 \leqq i < \infty$$

which have the following interpretation:

(i) The size of the table can assume only the values $n_i r_i$ for $i = 1, 2, \cdots$. At the start of the ith-*stage* the table is of size $n_i r_i$ and contains $n_i(r_i - 1)$ genuine keys distributed as shown below.

k_0	k_0	k_1	$\cdots$	k_{r_i-1}	$\cdots$	$k_{(n_i-1)r_i}$	$k_{(n_i-1)r_i}$	$k_{(n_i-1)r_i+1}$	$\cdots$	$k_{n_i(r_i-1)}$

(ii) When the size of the table is $n_i r_i$, the number of genuine keys in the table N satisfies $n_i(r_i - 1) \leqq N < n_{i+1}(r_{i+1} - 1)$.

(iii) To insert the key k into a table of size $n_i r_i$ presently containing N genuine keys with $n_i(r_i - 1) \leqq N < n_{i+1}(r_{i+1} - 1) - 1$, the address s satisfying

$$y_{s-1} < k < y_s, \quad 0 \leqq s \leqq n_i r_i \quad (y_{-1} = -\infty, y_{n_1 r_1} = +\infty)$$

is determined, for example, by a binary search. There are three possibilities:

(a) y_s is a dummy key $(0 \leqq s < n_i r_i)$;
y_s is replaced by k yielding the table

$$(y_0, y_1, \cdots, y_{s-1}, k, y_{s+1}, \cdots, y_{m-1}).$$

No moves are required to insert k in (a).

(b) y_s is a dummy key.

$$s = n_i r_i, \; y_0 \neq y_1 \neq \cdots \neq y_t = y_{t+1}, \; t \geqq 0.$$

k is larger than all keys presently in the table. The block of genuine keys $(y_0, y_1, \cdots, y_{t-1})$ is shifted one place to the right *displacing* the dummy key with value $y_t = y_{t+1}$, k is inserted at address 0, the pointer TO is adjusted yielding the table

$$(y_0, y_1, \cdots, y_{t-1}, y_{t+1}, y_{t+2}, \cdots, y_{m-1}, k).$$

t moves are required to insert k in (b).

(c) y_s is a genuine key and

$$y_s \neq y_{s+1} \neq \cdots \neq y_{s+t-1} \neq y_{s+t} = y_{s+t+1}.$$

The sequence of t genuine keys $(y_s, y_{s+1}, \cdots, y_{s+t-1})$ is *shifted* circularly "to the right" one address displacing the dummy key with value $y_{s+t} = y_{s+t+1}$, the pointer TO is adjusted if necessary, and k is inserted at address s yielding the table

$$(y_0, y_1, \cdots, y_{s-1}, k, y_s, \cdots, y_{s+t-1}, y_{s+t+1}, \cdots, y_{m-1}).$$

t moves are required to insert k in (c).

(iv) A *reconfiguration* of the table takes place when the key k is to be inserted into a table of size $n_i r_i$ presently containing $n_{i+1}(r_{i+1}-1)-1$ genuine keys. The size of the table is first increased $n_{i+1}r_{i+1}$; then k and the $n_{i+1}(r_{i+1}-1)-1$ genuine keys currently in the table are placed in the expanded table arranged in n_{i+1} blocks each of size r_{i+1} with their values in increasing order. Within each block, the "leftmost" key is a dummy key equal in value to the key standing "to its right." This produces a table with skeleton

$$(0, (1)^{(r_{i+1}-1)}, 0, (1)^{(r_{i+1}-1)}, \cdots, 0, (1)^{(r_{i+1}-1)})$$

where $(1)^{(s)}$ denotes a vector with s 1's. The pointer TO is reset to 0. The *cost* of the reconfiguration, as measured by the number of records moved, is $n_{i+1}(r_{i+1}-1)$.

The ratio $\rho = N/n_i r_i$ with $n_i(r_i-1) \leqq N < n_{i+1}(r_{i+1}-1)$ is the *density* of genuine keys in the table. Note that $1-1/r_i \leqq \rho \leqq 1$.

Remark 2.1. If in the table $\bar{y}$ with $|\bar{y}| = nr$ we have $y_{s+t} = y_{s+t+1}$ then y_{s+t} is a dummy key and y_j is a dummy key only *if* $j-(s+t) = 0$ (modulo r).

Example 2.2. If $r_i = r \geqq 2$ $(1 \leqq i < \infty)$, $n_i = \lfloor C\varepsilon^i (0 \leqq i < \infty)$ where $\lfloor$ is the *integer-part-of* and $1 < \varepsilon < r/(r-1)$, the $\{r_i\}$ and $\{n_i\}$ satisfy (2.1a), (2.1b) for suitable C. The cost of a reconfiguration per key inserted (between consecutive reconfigurations) is proportional to $\varepsilon(r-1)/(\varepsilon-1)$, independent of i.

Example 2.3. The sequences

$$\{n_i : 1 \leqq i < \infty\} = \{1, 3, 4, 7, 6, \cdots\},$$
$$\{r_i : 1 \leqq i < \infty\} = \{3, 2, 2, 2, 3, \cdots\}$$

satisfy (2.1a), (2.1b) and show that neither $\{n_i\}$ nor $\{r_i\}$ need be monotone. The state after the insertion of $(k_0, k_1, \cdots, k_6) = (0, 2, 3, 5, 6, 9, 10)$ is

$$\bar{y} = (0, 0, 2, 2, 3, 3, 5, 5, 6, 6, 9, 9, 10, 10), \qquad \text{TO} = 0,$$
$$|\bar{y}| = n_4 r_4 = 14, \qquad \|\bar{y}\| = n_4(r_4 - 1) = 7.$$

Inserting the keys $\{k_7, k_8, k_9, k_{10}\} = \{1, 7, 8, 11\}$ produces the table state

$$\bar{y}' = (0, 1, 2, 3, 3, 5, 5, 6, 6, 7, 8, 9, 10, 11), \qquad \text{TO} = 1.$$

While the table state $\bar{y}'$ is independent of the order in which the keys $\{1, 7, 8, 11\}$ are inserted, the number of moves

$$M(k_7, k_8, k_9, k_{10}) = (m_7, m_8, m_9, m_{10})$$

depends on order of insertion as shown below:

$M(1,7,8,11)=(0,0,1,0)$, $M(1,7,11,8)=(0,0,0,1)$, $M(1,8,7,11)=(0,0,2,0)$,
$M(1,8,11,7)=(0,0,0,2)$, $M(1,11,7,8)=(0,0,0,1)$, $M(1,11,8,7)=(0,0,0,2)$,
$M(7,1,8,11)=(0,0,1,0)$, $M(7,1,11,8)=(0,0,0,1)$, $M(8,1,7,11)=(0,0,2,0)$,
$M(8,1,11,7)=(0,0,0,2)$, $M(11,1,7,8)=(0,0,0,1)$, $M(11,1,8,7)=(0,0,0,2)$,
$M(7,8,1,11)=(0,1,0,0)$, $M(7,11,1,8)=(0,0,0,1)$, $M(8,7,1,11)=(0,2,0,0)$,
$M(8,11,1,7)=(0,0,0,2)$, $M(11,7,1,8)=(0,0,0,1)$, $M(11,8,1,7)=(0,0,0,2)$,
$M(7,8,11,1)=(0,1,0,0)$, $M(7,11,8,1)=(0,0,1,0)$, $M(8,7,11,1)=(0,2,0,0)$,
$M(8,11,1,7)=(0,0,2,0)$, $M(11,7,8,1)=(0,0,1,0)$, $M(11,8,7,1)=(0,0,2,0)$.

If the key value 4 is to be inserted next into the table, reconfiguration is required; the table is expanded to size $n_5 r_5 = 18$ and the $n_5(r_5-1)=12$ genuine keys

$$\{0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11\} \cup \{4\}$$

are distributed into $n_5=6$ blocks (of size $r_5=3$), each containing $r_5-1=2$ genuine keys, yielding the state

$$\bar{y}''=(0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, 9, 10, 10, 11)$$

and the pointer TO is reset to 0.

DEFINITION 2.4. We write $\bar{y}=\text{INSERT}(k_0, k_1, \cdots, k_{N-1})$ to indicate that the *insertion sequence* $(k_0, k_1, \cdots, k_{N-1})$ has produced the table state $\bar{y}$.

Remark 2.5. It is clear that the number of "operations" needed to build a table under a sequence of insertions depends on the values of the keys only to the extent of their relative ordering; more precisely, if the permutation $\eta=(\eta_0, \eta_1, \cdots, \eta_{N-1})$ of $0, 1, \cdots, N-1$ sorts both of the insertion sequences $k_0, k_1, \cdots, k_{N-1}$ and $k'_0, k'_1, \cdots, k'_{N-1}$,

$$k_{\eta_0} < k_{\eta_1} < \cdots < k_{\eta_{N-1}}, \qquad k'_{\eta_0} < k'_{\eta_1} < \cdots < k'_{\eta_{N-1}},$$

then the number of moves needed to build the tables with states

$$\text{INSERT}(k_0, k_1, \cdots, k_{N-1}), \qquad \text{INSERT}(k'_0, k'_1, \cdots, k'_{N-1})$$

are the same. Hence throughout the paper the N keys may be taken as the integers 0 to $N-1$.

3. Outline and combinatorial preliminaries. In § 6 the "probability" distribution of the number of moves to insert an $(N+1)$st-*key* k_N into a table $\bar{y}$ with $|\bar{y}|=nr$, $\|\bar{y}\|=N$ which has been constructed by a padded list schema is calculated using the results of this and the succeeding two sections.

The idea underlying the calculation is easy to describe; the mapping INSERT is from *insertion sequences* to table states $\bar{y}$

$$\text{INSERT}: (k_0, k_1, \cdots, k_{N-1}) \to \bar{y}, \qquad |\bar{y}|=nr \,\|\, \bar{y}\|=N.$$

Without loss of generality, we may assume that the $N+1$ keys $k_0, k_1, \cdots, k_N$ are some permutation of the integers $0, 1, \cdots, N$. If $k_N=i$ and its insertion into

$$\bar{y}=\text{INSERT}(k_0, k_1, \cdots, k_{N-1})$$

requires the movement of m keys, then $\bar{y}$ must have one of the following two configurations (Figs. 3.1 and 3.2) where $\varnothing$ denotes a dummy key:

$\cdots$	$\varnothing$	$\cdots$	$i+1$	$i+2$	$\cdots$	$i+m$	$i+m+1$	$i+m+1$	$\cdots$

FIG. 3.1. *No "Wraparound": $i+m<nr$.*

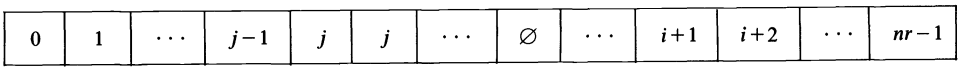


FIG. 3.2. "*Wraparound*": $(i+m+1)$ (modulo nr) $=j$.

To calculate the probability that the insertion of $k_N = i$ causes the movement of m records, we must enumerate the insertion sequences $(k_0, k_1, \cdots, k_{N-1})$ which give rise to the table as in Fig. 3.1 or 3.2.

This enumeration is carried out as follows: First in Lemma 3.1 we show that the mapping INSERT $(k_0, k_1, \cdots, k_{N-1})$ is "constant" on certain subsets of the $N!$ insertion sequences (of the key values $0, 1, \cdots, N-1$) related to the keys that are inserted *before* and *after* the last reconfiguration. Each partition $\mathbf{K}$ corresponds to $(n(r-1))! \times (N-n(r-1))!$ insertion sequences. INSERT therefore induces a mapping, denoted by PINSERT, on partitions $\mathbf{K} = (\mathbf{K}(-), \mathbf{K}(+))$ of the set of key values.

Next, each partition $\mathbf{K}$ is in a one-to-one correspondence with an *assignment vector* $\bar{u} = (u_{K,0}, u_{K,1}, \cdots, u_{K,n})$ (Definition 3.2) and the enumeration is carried out in terms of these vectors.

When there is no "wraparound" the table state $\bar{y}$ decomposes into two subtables (see Fig. 3.3) which are "independent" as clarified in Remark 3.9. Note that

- $|\bar{y}''| = kr$ and $\|\bar{y}''\| = kr-1$ for some k with $kr > m$;
- the smallest key value in the subtable $\bar{y}''$ is a dummy key;
- a dummy key with value $i+m+1$ is contained in the subtable $\bar{y}'$.

As discussed in Remark 3.9, the number of insertion sequences (partitions) which gives rise to this type of table state $\bar{y}$ is the product of the number which gives rise to the table state $\bar{y}'$ and the number which gives rise to the table state $\bar{y}''$. The number of ways of obtaining the table substate $\bar{y}''$ is given in § 4.1, those for $\bar{y}'$ in § 5.2.

When there is "wraparound", the memory state $\bar{z}$ decomposes into two memory substates (Fig. 3.4).

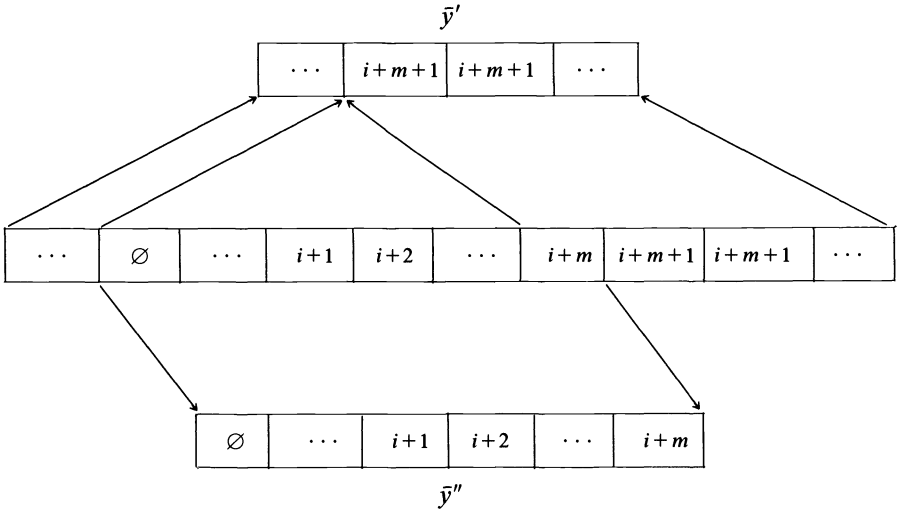


FIG. 3.3. *Decomposition without "Wraparound"*.

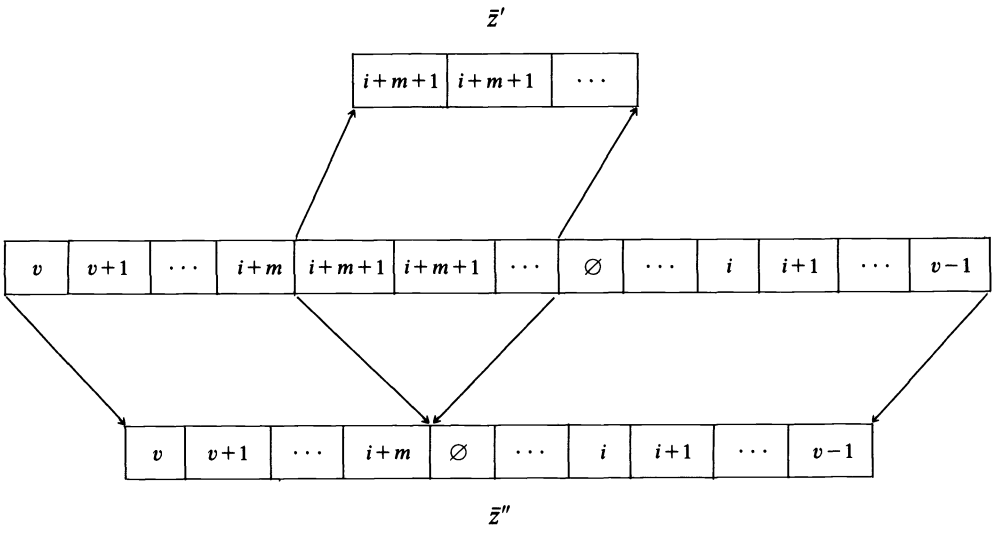


FIG. 3.4. *Decomposition with "Wraparound".*

The number of ways the memory substate $\bar{z}''$ can arise is counted in § 4.2, those for the memory substate $\bar{z}'$ in § 5.1.

As the numeration is made between consecutive reconfigurations, we simplify the notation writing n for n_i and r for r_i to indicate the "current" values.

Let $\bar{y}=(y_0, y_1, \cdots, y_{nr-1})$ be a table containing N genuine keys with $n(r-1)\leqq N<nr$. The mapping INSERT from insertion sequences $k_0, k_1, \cdots, k_{N-1}$ (of fixed length N) to table states $\bar{y}=(y_0, y_1, \cdots, y_{nr-1})$ with $\|\bar{y}\|=N$ is clearly many-to-one. More precisely

LEMMA 3.1. INSERT $(k_0, k_1, \cdots, k_{N-1})=$ INSERT $(k'_0, k'_1, \cdots, k'_{N-1})$ *if*

$$k'_0, k'_1, \cdots, k'_{n(r-1)-1} \quad \textit{is a permutation of } k_0, k_1, \cdots, k_{n(r-1)-1},$$

$$k'_{n(r-1)}, k'_{n(r-1)+1}, \cdots, k'_{N-1} \quad \textit{is a permutation of } k_{n(r-1)}, k_{n(r-1)+1}, \cdots, k_{N-1}.$$

Proof. It is clear that permuting the entries in the insertion subsequence $k_0, k_1, \cdots, k_{n(r-1)-1}$ does not alter the table $\bar{y}=$ INSERT $(k_0, k_1, \cdots, k_{N-1})$ since the keys $k_0, k_1, \cdots, k_{n(r-1)-1}$ are sorted when they are placed in the expanded table at the last reconfiguration. The invariance of INSERT $(k_0, k_1, \cdots, k_{N-1})$ under permutations of the insertion subsequence $k_{n(r-1)}, k_{n(r-1)+1}, \cdots, k_{N-1}$ is proved by induction on N with $N\geqq n(r-1)+1$. When $N=n(r-1)+1$, the assertion is trivial; by induction, it suffices to show that the two insertion sequences

$$k_0, k_1, \cdots, k_{N-3}, k_{N-2}, k_{N-1}, \qquad k_0, k_1, \cdots, k_{N-3}, k_{N-1}, k_{N-2}$$

determine the same table state $\bar{y}$. Suppose

(a) $\bar{y}=$ INSERT $(k_0, k_1, \cdots, k_{N-3}, k_{N-2})$,

(b) $y_{s-1}<k_{N-1}<y_s$,

(c) $y_s\neq y_{s+1}\neq\cdots\neq y_{s+t-1}\neq y_{s+t}=y_{s+t+1}, \qquad t\geqq 0$

so that y_{s+t} is the first dummy key "to the right" of the genuine key y_s. There are two cases to be examined:

Case 1: $k_{N-2} \in \{y_s, y_{s+1}, \cdots, y_{s+t-1}\}$. Suppose $k_{N-2} = y_{s+u}$. Since the insertion of k_{N-2} displaces some dummy key[1], we conclude from Remark 2.1 and (c) above that the displaced dummy key had a value equal to one of the (genuine) keys $y_{s+t-r+1}$, $y_{s+t-2r+1}, \cdots$. Hence

$$\begin{aligned}\text{INSERT}\,(k_0, k_1, \cdots, k_{N-3}, k_{N-2}) \\ = (\cdots, y_{s-1}, y_s, \cdots, y_{s+u-1}, k_{N-2}, y_{s+u+1}, \cdots, y_{s+t-vr-1}, \\ y_{s+t-vr+1}, \cdots, y_{s+t}, y_{s+t+1}, \cdots)\end{aligned}$$

and

$$\begin{aligned}\text{INSERT}\,(k_0, k_1, \cdots, k_{N-3}, k_{N-2}, k_{N-1}) \\ = (\cdots, y_{s-1}, k_{N-1}, y_s, \cdots, y_{s+u-1}, k_{N-2}, y_{s+u+1}, \cdots, y_{s+t-vr-1}, \\ y_{s+t-vr+1}, \cdots, y_{s+t-1}, y_{s+t+1}, \cdots).\end{aligned}$$

If the order of insertion of k_{N-2} and k_{N-1} is reversed, then k_{N-1} displaces y_{s+t-vr}

$$\begin{aligned}\text{INSERT}\,(k_0, k_1, \cdots, k_{N-3}, k_{N-1}) \\ = (\cdots, y_{s-1}, k_{N-1}, y_s, \cdots, y_{s+t-vr-1}, y_{s+t-vr+1}, \cdots, y_{s+t}, y_{s+t+1})\end{aligned}$$

and

$$\begin{aligned}\text{INSERT}\,(k_0, k_1, \cdots, k_{N-3}, k_{N-1}, k_{N-2}) \\ = (\cdots, y_{s-1}, k_{N-1}, y_s, \cdots, y_{s+u-1}, k_{N-2}, \\ y_{s+u+1}, \cdots, y_{s+t-vr-1}, y_{s+t-vr+1}, \cdots, y_{s+t-1}, y_{s+t+1}, \cdots)\end{aligned}$$

so that both insertion orders produce the same table.

Case 2: $k_{N-2} \in \{y_s, y_{s+1}, \cdots, y_{s+t-1}\}$. In this case the insertions of k_{N-1} and k_{N-2} involve disjoint blocks of keys (i.e., separated by dummy keys that have not been affected by the insertions so far), so that both insertion sequences trivially yield the same table. □

Lemma 3.1 shows that the state of the table $\bar{y}$, containing the collection of records $\mathbf{K}$ is determined by the partition $\mathbf{K} = (\mathbf{K}(-), \mathbf{K}(+))$, such that

$$\mathbf{K}(-) = \{k_0, k_1, \cdots, k_{n(r-1)-1}\}, \tag{3.1a}$$

$$\mathbf{K}(+) = \{k_{n(r-1)}, k_{n(r-1)+1}, \cdots, k_{N-1}\}, \tag{3.1b}$$

$$\mathbf{K}(-) \cap \mathbf{K}(+) = \varnothing. \tag{3.1c}$$

Let $\Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$ denote the family of partitions as in (3.1a)-(3.1c) of the key values $\{k_0, k_1, \cdots, k_{N-1}\}$. Lemma 3.1 shows that INSERT induces a mapping, which we denote by PINSERT, from partitions $\mathbf{K} \in \Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$ into table states $\bar{y}$ with $|\bar{y}| = nr$ and $\|\bar{y}\| = N$.

Let $\eta_K = (\eta_{K,0}, \eta_{K,1}, \cdots, \eta_{K,n(r-1)-1})$ denote the permutation of the integers $0, 1, \cdots, n(r-1)-1$ which sorts the key values in $\mathbf{K}(-)$:

$$k_{\eta_{K,0}} < k_{\eta_{K,1}} < \cdots < k_{\eta_{K,n(r-1)-1}}.$$

[1] We write the table states as if $u > 1$ (or $k_{N-2} > y_{s+1}$), but the special cases $u = 0, 1$ are similarly simple to handle.

The dummy keys in the table immediately following the last reconfiguration have the values $\{k_{\eta_{K,j(r-1)}}: 0 \leqq j < n\}$.

We shall say that the jth dummy key is *displaced* (by the insertion of the keys in $\mathbf{K}(+)$) if no dummy key with the value $k_{\eta_{K,j(r-1)}}$ is contained in $\bar{y} = \text{PINSERT}(\mathbf{K})$; equivalently, the associated memory state satisfies $z_{jr} \neq z_{jr+1}$.

DEFINITION 3.2. The *assignment vector* of a partition

$$\mathbf{K} \in \Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$$

is the vector $\bar{u}_K = (u_{K,0}, u_{K,1}, \cdots, u_{K,n})$ defined by

$$u_{K,i} = \begin{cases} |\{k \in \mathbf{K}(+): k < k_{\eta_{K,0}}\}| & \text{if } i = 0, \\ |\{k \in \mathbf{K}(+): k_{\eta_{K,(i-1)(r-1)}} < k < k_{\eta_{K,i(r-1)}}\}| & \text{if } 1 \leqq i < n, \\ |\{k \in \mathbf{K}(+): k > k_{\eta_{K,(n-1)(r-1)}}\}| & \text{if } i = n \end{cases} \tag{3.2}$$

where $|E|$ denotes the cardinality of the (set) E.

Note that (3.2) implies

$$u_{K,0} + u_{K,1} + \cdots + u_{K,n} = N - n(r-1). \tag{3.3}$$

Remark 3.3. If the insertion sequence $\{k_0, k_1, \cdots, k_{N-1}\}$ is a permutation of the integers $0, 1, \cdots, N-1$, then (3.2) shows that

$$k_{\eta_{K,i(r-1)}} = u_0 + u_1 + \cdots + u_i + i(r-1), \qquad 0 \leqq i < n.$$

DEFINITION 3.4. Let $U(m, M)$ denote the set $\bar{u} = (u_0, u_1, \cdots, u_{m-1})$ of ordered partitions of M into m parts

$$u_i \geqq 0 \quad (0 \leqq i < m), \qquad u_0 + \cdots + u_{m-1} = M.$$

LEMMA 3.5. *When $n(r-1) \leqq N < nr$, every $\bar{u} \in U(n+1, N-n(r-1))$ is an assignment vector for*

$$\prod_{1 \leqq i \leqq n} \binom{u_i + r - 2}{r-2}$$

partitions $\mathbf{K} \in \Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$.

Proof. Let $\bar{u} \in U(n+1, N-n(r-1))$ and introduce the notation

$$U_j = u_0 + u_1 + \cdots + u_j.$$

Consider a partition of the integers $0, 1, \cdots, N-1$ as before:

$$\mathbf{K}(-) = \{k_0, k_1, \cdots, k_{n(r-1)-1}\}, \qquad \mathbf{K}(+) = \{k_{n(r-1)}, k_{n(r-1)+1}, \cdots, k_{N-1}\}$$

where without loss of generality we may assume

$$k_0 < k_1 < \cdots < k_{n(r-1)-1}, \qquad k_{n(r-1)} < k_{n(r-1)+1} < \cdots < k_{N-1}.$$

By Remark 3.3 the vector $\bar{u}$ above may serve as an assignment vector for the partition $\mathbf{K}$ if for $0 \leqq j < n$ it satisfies (i)-(iii) below and for $0 \leqq s < u_0$ it satisfies (iv) below:

(i) $k_{j(r-1)} = U_j + j(r-1)$,

(ii) $\{k_{j(r-1)+s}: 1 \leqq s < r-1\} \subseteq \{U_j + j(r-1) + s: 1 \leqq s < u_{j+1} + r - 1\}$,

(iii) $\{k_{n(r-1)+U_j+s}: 0 \leqq s < u_{j+1}\} = \{U_j + j(r-1) + s: 1 \leqq s < u_{j+1} + r - 1\}$,

(iv) $k_{n(r-1)+s} = s, \qquad 0 \leqq s < u_0$.

Since the number of keys in $\mathbf{K}(+)$ that are assigned to the jth block is

$$|\{k_{n(r-1)+U_j+s}: 0 \leqq s < u_{j+1}\}| = u_{j+1}$$

and they are selected from a range of values of size

$$|\{U_j+j(r-1)+s\colon 1\leqq s<u_{j+1}+r-1\}|=u_{j+1}+r-2$$

and $|\{k_{j(r-1)+s}\colon 1\leqq s<r-1\}|=r-2$ there are $\binom{u_{j+1}+r-2}{r-2}$ partitions of the set

$$\{U_j+j(r-1)+s\colon 1\leqq s<u_{j+1}+r-1\}$$

into the subsets $\{k_{j(r-1)+s}\colon 1\leqq s<r-1\}$ and $\{k_{n(r-1)+U_j+s}\colon 0\leqq s<u_{j+1}\}$ for each $j, 0\leqq j<n$. An alternative view is that $\mathbf{K}(-)$ and $\bar{u}$ determine $\mathbf{K}(+)$, in (iii) and (iv). □

LEMMA 3.6. *If $\bar{u}_{\mathbf{K}}$ is an assignment vector for $\mathbf{K}\in\Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$ and $u_{K,j}=s>0$, the jth dummy key and the $s-1$ dummy keys located "to its right" circularly are displaced in $\bar{y}=\text{PINSERT}(\mathbf{K})$.*

Proof. Suppose $j<n$; if $s=u_{K,j}>0$, then s keys in $\mathbf{K}(+)$ have values that are larger than $k_{\eta_{K,(j-1)(r-1)}}$ and smaller than $k_{\eta_{K,j(r-1)}}$. Since the insertion of a key from $\mathbf{K}(+)$ displaces the first dummy key "to the right" of the address determined by the (binary) search, the lemma is proved. The case $j=n$ only requires minor modification of this argument, which we leave for the reader. □

LEMMA 3.7. *If $\bar{u}_{\mathbf{K}}$ is an assignment vector for $\mathbf{K}\in\Pi_r(n, N)\{k_0, k_1, \cdots, k_{N-1}\}$ the jth dummy key is not displaced in $\bar{y}=\text{PINSERT}(\mathbf{K})$ if it satisfies:*

$$u_{K,s}+u_{K,s+1}+\cdots+u_{K,j}\leqq j-s, \qquad 0\leqq s\leqq j, \tag{3.4a}$$

$$u_{K,s}+u_{K,s+1}+\cdots+u_{K,n}+u_{K,0}+u_{K,1}+\cdots+u_{K,j}\leqq j+n-s, \qquad j<s\leqq n. \tag{3.4b}$$

Proof. The proof is the same as for Lemma 3.6. □

Remark 3.8. The cardinalities of the inverse images of $\bar{y}$ under INSERT and PINSERT

$$\text{INSERT}^{-1}(\bar{y})=\{k_0, k_1, \cdots, k_{N-1}\colon \bar{y}=\text{INSERT}(k_0, k_1, \cdots, k_{N-1})\},$$

$$\text{PINSERT}^{-1}(\bar{y})=\{\mathbf{K}\colon \bar{y}=\text{PINSERT}(\mathbf{K})\}$$

are related by

$$|\text{INSERT}^{-1}(\bar{y})|=(n(r-1))!\times(N-n(r-1))!\times|\text{PINSERT}^{-1}(\bar{y})|.$$

Now that we have associated insertion sequences with partitions, and partition counts with assignment vectors, we proceed to associate assignment vectors with table skeleton structures having specified dummy keys. The latter will appear in § 6 to be directly related to the number of moves required to insert a record.

Remark 3.9. To evaluate $|\text{PINSERT}^{-1}(\bar{y})|$ with $|\bar{y}|=nr$ and $\|\bar{y}\|=N$, we use the following "renewal-type" argument; if the skeleton $\bar{\sigma}$ of $\bar{y}$ satisfies

$$\sigma_t=\begin{cases}0 & \text{if } t=i,\\ 1 & \text{if } i<t<j,\\ 0 & \text{if } t=j,\end{cases}$$

then

(i) $j-i=Lr$ for some integer L;

(ii) The insertions of the keys into each of the two subtables

$$\bar{y}_1=(y_i, y_{i+1}, \cdots, y_{j-1}),$$

$$\bar{y}_2=(y_0, y_1, \cdots, y_{i-1}, y_j, y_{j+1}, \cdots, y_{nr-1})$$

are independent; meaning, if

$$\mathbf{K}_1=(\mathbf{K}_1(-), \mathbf{K}_1(+))\in\text{PINSERT}^{-1}(\bar{y}_1),$$

$$\mathbf{K}_2=(\mathbf{K}_2(-), \mathbf{K}_2(+))\in\text{PINSERT}^{-1}(\bar{y}_2)$$

then

$$\bar{y} = \text{PINSERT}(\mathbf{K}_1 \cup \mathbf{K}_2),$$

$$|\text{PINSERT}^{-1}(\bar{y})| = |\text{PINSERT}^{-1}(\bar{y}_1)| \times |\text{PINSERT}^{-1}(\bar{y}_2)|$$

and conversely, every partition $\mathbf{K} \in \text{PINSERT}^{-1}(\bar{y})$ may be described in such a fashion.

Note that

$$|\bar{y}_1| = Lr, \qquad \|\bar{y}_1\| = Lr - 1,$$

$$|\bar{y}_2| = (n-L)r, \qquad \|\bar{y}_2\| = N - Lr + 1.$$

Therefore we need only determine $|\text{PINSERT}^{-1}(\bar{y}_1)|$ for table states $\bar{y}_1$ which have skeletons of the form $((1)^{(t)}, 0, (1)^{(nr-t-1)})$, i.e., with a single specified dummy key, to build up from it the computation of $|\text{PINSERT}^{-1}(\bar{y})|$ for arbitrary table $\bar{y}$.

4. Tables with a single dummy key. Let $\Omega_r(n, t)$ be the set of partitions $\mathbf{K} \in \Pi_r(n, nr-1)\{0, 1, \cdots, nr-2\}$ which satisfy $\text{PINSERT}(\mathbf{K}) = (0, 1, \cdots, t-1, t, t, t+1, \cdots, nr-2)$. Note that the value of the pointer TO is left unspecified.

We define the decomposition $\Omega_r(n, t) = \bigcup_j \Omega_r(n, t, j)$ where $\Omega_r(n, t, j)$ consists of the partitions in $\Omega_r(n, t)$ for which $k_{\eta_{K,j(r-1)}} = t$. Let

$$\omega_r(n, t) = |\Omega_r(n, t)|, \qquad \omega_r(n, t, j) = |\Omega_r(n, t, j)|.$$

4.1. The single dummy key is 0. We begin by examining the special case $t = 0$, which implies $\text{TO} = 0$ and $\omega_r(n, 0) = \omega_r(n, 0, 0)$. With reference to Fig. 3.3, the subtable $\bar{y}''$ with parameters $|\bar{y}''| = kr$ and $\|\bar{y}''\| = kr - 1$ contains keys with values $i+m, i+m-1, \cdots, i+m-kr+2$. The only dummy key in the subtable $\bar{y}''$ has value $i+m-kr+2$. The number of ways of inserting these $kr-2$ keys in a table of size kr without displacing the dummy key $i+m-kr+2$ is the same as the number of ways of inserting keys with values $0, 1, \cdots, kr-2$ without displacing the dummy key with value 0.

LEMMA 4.1. *The assignment vector $\bar{u}_K$ of any $\mathbf{K} \in \Omega_r(n, 0)$ satisfies*

$$u_{K,0} + u_{K,1} + \cdots + u_{K,s} \begin{cases} = 0 & \text{if } s = 0, \\ \geqq s & \text{if } 1 \leqq s < n, \\ = n-1 & \text{if } s = n. \end{cases} \tag{4.1}$$

Proof. Lemma 3.6 states that $\bar{u}_K$ must satisfy $u_{K,0} = u_{K,n} = 0$, and

$$u_{K,s} + u_{K,s+1} + \cdots + u_{K,n-1} \leqq n - s, \qquad 0 < s < n,$$

so that (4.1) follows from (3.3). □

We proceed now to enumerate such vectors. Let $U^*(n+1, n-1)$ denote the subset of $U(n+1, n-1)$ consisting of vectors which satisfy (4.1). By Lemma 3.5 each $\bar{u} \in U^*(n+1, n-1)$ is an assignment vector for

$$\prod_{j=1}^{n} \binom{u_j + r - 2}{r-2}$$

partitions in $\Omega_r(n, 0)$.

Example 4.2. Let $r = 3$ and $n = 4$. The assignment vectors

$$\bar{u} = (u_0, u_1, u_2, u_3, u_4)$$

which yield the table $\bar{y} = (0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ are required to satisfy

$$u_0 = 0, \qquad u_0 + u_1 \geqq 1, \qquad u_0 + u_1 + u_2 \geqq 2,$$

$$u_0 + u_1 + u_2 + u_3 \geqq 3, \qquad u_0 + u_1 + u_2 + u_3 + u_4 = 3.$$

The five possible assignment vectors and their corresponding thirty partitions are given in Table 4.1.

LEMMA 4.3. $\omega_r(n,0)=\left|\text{PINSERT}^{-1}(0,0,1,2,\cdots,nr-2)\right|$ *is given by*

$$\omega_r(n,0)=\frac{1}{nr-1}\binom{nr-1}{n},\qquad 1\leqq n<\infty. \tag{4.2}$$

Proof. By Lemma 3.5,

$$\omega_r(n,0)=\sum_{\bar{u}\in U^*(n+1,n-1)}\prod_{j=1}^{n}\binom{u_j+r-2}{r-2}. \tag{4.3}$$

To evaluate the sum in (4.3) write $U^*(n+1,n-1)$ as the difference of two sets

$$U^*(n+1,n-1)=U'(n+1,n-1)-U''(n+1,n-1)$$

where

$$U'(n+1,n-1)=U(n+1,n-1)\cap\{\bar{u}\colon u_0=0\},$$

$$U''(n+1,n-1)=\bigcup_{k=1}^{n-1}U_k''(n+1,n-1)$$

TABLE 4.1

$\bar{u}$	$\mathbf{K}(-)$	$\mathbf{K}(+)$
(0, 3, 0, 0, 0)	{0, 1, 5, 6, 7, 8, 9, 10}	{2, 3, 4}
	{0, 2, 5, 6, 7, 8, 9, 10}	{1, 3, 4}
	{0, 3, 5, 6, 7, 8, 9, 10}	{1, 2, 4}
	{0, 4, 5, 6, 7, 8, 9, 10}	{1, 2, 3}
(0, 2, 0, 1, 0)	{0, 1, 4, 5, 6, 7, 9, 10}	{2, 3, 8}
	{0, 2, 4, 5, 6, 7, 9, 10}	{1, 3, 8}
	{0, 3, 4, 5, 6, 7, 9, 10}	{1, 2, 8}
	{0, 1, 4, 5, 6, 8, 9, 10}	{2, 3, 7}
	{0, 2, 4, 5, 6, 8, 9, 10}	{1, 3, 7}
	{0, 3, 4, 5, 6, 8, 9, 10}	{1, 2, 7}
(0, 2, 1, 0, 0)	{0, 1, 4, 5, 7, 8, 9, 10}	{2, 3, 6}
	{0, 2, 4, 5, 7, 8, 9, 10}	{1, 3, 6}
	{0, 3, 4, 5, 7, 8, 9, 10}	{1, 2, 6}
	{0, 1, 4, 6, 7, 8, 9, 10}	{2, 3, 5}
	{0, 2, 4, 6, 7, 8, 9, 10}	{1, 3, 5}
	{0, 3, 4, 6, 7, 8, 9, 10}	{1, 2, 5}
(0, 1, 2, 0, 0)	{0, 1, 3, 4, 7, 8, 9, 10}	{2, 5, 6}
	{0, 1, 3, 5, 7, 8, 9, 10}	{2, 4, 6}
	{0, 1, 3, 6, 7, 8, 9, 10}	{2, 4, 5}
	{0, 2, 3, 4, 7, 8, 9, 10}	{1, 5, 6}
	{0, 2, 3, 5, 7, 8, 9, 10}	{1, 4, 6}
	{0, 2, 3, 6, 7, 8, 9, 10}	{1, 4, 5}
(0, 1, 1, 1, 0)	{0, 1, 3, 4, 6, 7, 9, 10}	{2, 5, 8}
	{0, 1, 3, 4, 6, 8, 9, 10}	{2, 5, 7}
	{0, 1, 3, 5, 6, 7, 9, 10}	{2, 4, 8}
	{0, 1, 3, 5, 6, 8, 9, 10}	{2, 4, 7}
	{0, 2, 3, 4, 6, 7, 9, 10}	{1, 5, 8}
	{0, 2, 3, 4, 6, 8, 9, 10}	{1, 5, 7}
	{0, 2, 3, 5, 6, 7, 9, 10}	{1, 4, 8}
	{0, 2, 3, 5, 6, 8, 9, 10}	{1, 4, 7}

and

$$U_k''(n+1, n-1)$$
$$= \{\bar{u}\colon u_0 = 0,\ u_0 + \cdots + u_s \geqq s,\ 0 < s < k,\ u_0 + \cdots + u_k = k-1,\ u_{k+1} + \cdots + u_n = n-k\}$$

resulting in

$$\omega_r(n, 0) = \sum_{\bar{u} \in U'(n+1, n-1)} \prod_{j=1}^{n} \binom{u_j + r - 2}{r-2} - \sum_{\bar{u} \in U''(n+1, n-1)} \prod_{j=1}^{n} \binom{u_j + r - 2}{r-2}.$$

For the first summation, observe that since $\binom{k+r-2}{r-2}$ is the coefficient of z^k in the binomial development of $(1-z)^{-(r-1)}$, the summation over the set $U'(n+1, n-1)$ is the coefficient of z^{n-1} in $(1-z)^{-n(r-1)}$, namely $\binom{nr-2}{n-1}$. For the second summation, note that

$$U_k'' = \{\bar{u}\colon (u_0, u_1, \cdots, u_k) \in U^*(k+1, k-1),\ (u_{k+1}, u_{k+2}, \cdots, u_n) \in U(n-k, n-k)\}$$

and hence

$$\omega_r(n, 0) = \binom{nr-2}{n-1} - \sum_{k=1}^{n-1} \left[\sum_{(u_0, \cdots, u_k) \in U^*(k+1, k-1)} \prod_{j=0}^{k} \binom{u_j + r - 2}{r-2} - \sum_{(u_{k+1}, \cdots, u_n) \in U(n-k, n-k)} \prod_{j=k+1}^{n} \binom{u_j + r - 2}{r-2} \right]$$
$$= \binom{nr-2}{n-1} - \sum_{k=1}^{n-1} \omega_r(k, 0) \binom{(n-k)r-1}{n-k}$$

leading to the recursion

$$\binom{nr-2}{n-1} = \sum_{j=1}^{n} \omega_r(j, 0) \binom{(n-j)r-1}{n-j}. \tag{4.4}$$

The right-hand side of (4.4) is the nth-term in the convolution of the two sequences

$$\{\omega_r(n, 0)\colon 1 \leqq n < \infty\}, \qquad \left\{\binom{nr-1}{n}\colon 0 \leqq n < \infty\right\}.$$

This suggests introducing the generating functions

$$H_r(w) = \sum_{n=1}^{\infty} \omega_r(n, 0) w^n, \qquad Q_{r,s}(w) = \sum_{n=0}^{\infty} \binom{nr+s}{n} w^n.$$

Equation (4.4) translates into the relationship

$$wQ_{r,r-2}(w) = H_r(w) Q_{r,-1}(w). \tag{4.5}$$

We prove in Appendix 1 that

$$Q_{r,s}(w) = \frac{X^{s+1}(w)}{r - (r-1)X(w)} \tag{4.6}$$

where $X(w)$ is the unique analytic function which satisfies

$$wX^r(w) = X(w) - 1, \qquad |w| < \frac{(r-1)^{r-1}}{r^r}, \tag{4.7a}$$

$$X(w) \to 1 \quad \text{as } w \to 0. \tag{4.7b}$$

Furthermore, we show in Appendix 1 that

$$X^s(w) = \sum_{n=0}^{\infty} w^n \frac{s}{nr+s} \binom{nr+s}{n}. \tag{4.8}$$

Equations (4.5) and (4.6) yield the formula

$$H_r(w) = wX^{r-1}(w) \tag{4.9}$$

which with (4.8) completes the proof. □

4.2. The single dummy key is arbitrary. We now turn to the general case and determine $\omega_r(n, t)$.

4.2.1. Specifying the assignment vectors. Lemma 4.4 will establish conditions on assignment vectors $\bar{u}_K$ for which the dummy key t in the associated partition $\mathbf{K}$ is *not* displaced.

With reference to Fig. 3.4, we view the submemory state $\bar{z}''$ as containing two *blocks*; the first block begins with the dummy key indicated by $\varnothing$ and occupies all locations up to address $nr-1$. The second block begins with address 0 and ends at address $i+m-v$. During the building of the table $\bar{y}$ (memory $\bar{z}$), the table origin pointer has had to be adjusted and the keys with values $v, v+1, \cdots, i+m$ have been inserted at or "displaced" into the "beginning" of the table. As we observed in Remark 3.9, the two submemory states are independent and the number of ways they can arise is the product of the number of ways of building the two submemory states $\bar{z}'$ and $\bar{z}''$. In this and the following section, we enumerate for the submemory $\bar{z}''$. The number of ways keys with values from the dummy $\varnothing$ to $i+m$ can be inserted is given by the product of two enumerators $\psi_{r,R}$ and $\psi_{r,L}$ defined in § 4.2.2.

LEMMA 4.4. (a) $\Omega_r(n, t, j) = \varnothing$ *unless*

$$j(r-1) \leqq t \leqq jr. \tag{4.10}$$

(b) *The assignment vector* $\bar{u}_K$ of $\mathbf{K} \in \Omega_r(n, t, j)$ *satisfies the conditions*

$$u_{K,j+1} + u_{K,j+2} + \cdots + u_{K,j+s+1} \begin{cases} \geqq s+1 & \text{if } 0 \leqq s < n-j-1, \\ = n+j(r-1)-t-1 & \text{if } s = n-j-1, \end{cases} \tag{4.11a}$$

$$u_{K,0} + u_{K,1} + \cdots + u_{K,s} \begin{cases} \geqq 0 & \text{if } 0 \leqq s < jr-t, \\ \geqq s-jr+t+1 & \text{if } jr-t \leqq s < j, \\ = t-j(r-1) & \text{if } s = j. \end{cases} \tag{4.11b}$$

Note that (4.11b) *implies* $u_{K,j} = 0$.

Proof. Lemma 3.7 asserts that $\bar{u}_K$ must satisfy

$$u_{K,s} + u_{K,s+1} + \cdots + u_{K,j} \leqq j-s, \qquad 0 \leqq s \leqq j, \tag{4.12}$$

$$u_{K,s} + u_{K,s+1} + \cdots + u_{K,n} + u_{K,0} + u_{K,1} + \cdots + u_{K,j} \leqq j+n-s, \qquad j < s \leqq n, \tag{4.13}$$

in order that the jth dummy key not be displaced. Observe that

$$u_{K,0} + u_{K,1} + \cdots + u_{K,j} = t - j(r-1), \tag{4.14}$$

since

$$\begin{aligned} |\{k \in \mathbf{K}(+): k < t\}| &= |\{0, 1, \cdots, t-1\}| - |\{k_{\eta_{K,s}}: 0 \leqq s < j(r-1)\}| \\ &= t - j(r-1), \end{aligned}$$

so that (4.13) becomes

$$u_{K,s} + u_{K,s+1} + \cdots + u_{K,n} \leqq n - s - t + jr, \qquad j < s \leqq n. \tag{4.15}$$

Similarly, from (4.14)

$$u_{K,j+1} + u_{K,j+2} + \cdots + u_{K,n} = n + j(r-1) - t - 1. \tag{4.16}$$

Subtracting (4.15) from (4.16) yields (4.11a). Subtracting (4.12) from (4.14) yields (4.11b).

The inequality $j(r-1) \leqq t$ follows from the nonnegativity of $|\{k \in \mathbf{K}(+): k < t\}|$. If $|\{k \in \mathbf{K}(+): k < t\}|$ exceeds j (so that $t > jr$), then Lemma 3.6 implies that the jth dummy key is displaced, a contradiction. This proves (4.10). □

There does not seem to be a direct way to count the assignment vectors that satisfy (4.11a), (4.11b). Hence we proceed as follows:

4.2.2. Partial assignment vectors. The assignment vectors specified in Lemma 4.3 will be now partitioned according to the components that fall "to the right" and "to the left" of the dummy key t. Let $U_R(m, M)$ denote the subset of $U(m, M)$ satisfying the conditions:

$$u_0 + u_1 + \cdots + u_i \begin{cases} \geqq i+1 & \text{if } 0 \leqq i < m-1, \\ = M & \text{if } i = m-1, \end{cases} \qquad m-1 \leqq M < \infty. \tag{4.17}$$

Let $U_L(m, M)$ denote the subset of $U(m, M)$ satisfying the conditions:

$$u_0 + u_1 + \cdots + u_i \begin{cases} \geqq 0 & \text{if } 0 \leqq i < m-M-1, \\ \geqq i-(m-M-2) & \text{if } m-M-1 \leqq i < m-1, \\ = M & \text{if } i = m-1, \end{cases} \qquad 0 \leqq M < m. \tag{4.18}$$

The reader will perhaps find the following calculations easier to follow if these partial assignment vectors are interpreted in terms of the dummy keys they cause to be displaced.

Observation. Every vector $\bar{u} = (u_0, u_1, \cdots, u_n)$ with

$$(u_0, u_1, \cdots, u_j) \in U_L(j+1, t-j(r-1)),$$

$$(u_{j+1}, +u_{j+2}, \cdots, u_n) \in U_R(n-j, n+j(r-1)-t-1)$$

satisfies (4.1a), (4.1b) and by Lemma 3.5 is the assignment vector for

$$\prod_{i=0}^{j} \binom{u_i+r-2}{r-2} \times \prod_{i=j+1}^{n} \binom{u_i+r-2}{r-2}$$

partitions in $\Omega_r(n, t, j)$. Note that only $j=0$ satisfies (4.10) when $t=0$, and the first term in the product above is equal then to one.

Example 4.5. Let $r=3$, $n=4$, $t=7$ and $j=3$. The assignment vectors

$$\bar{u} = (u_0, u_1, u_2, u_3, u_4)$$

which yield the table $\bar{y} = (0, 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10)$ are required to satisfy

$$u_3 = 0, \quad u_4 = 2, \quad u_0 \geqq 0, \quad u_0 + u_1 \geqq 0, \quad u_0 + u_1 + u_2 \geqq 1.$$

The assignment vectors and their corresponding partitions are given in Table 4.2.

LEMMA 4.6. $\omega_r(n, t, j) = \psi_{r,R}(n-j, n+j(r-1)-t-1)\psi_{r,L}(j+1, t-j(r-1))$ *where*

$$\psi_{r,R}(m, M) = \sum_{\bar{u} \in U_R(m,M)} \prod_{i=0}^{m-1} \binom{u_i+r-2}{r-2} \tag{4.19}$$

for $m-1 \leqq M < \infty$, $1 \leqq m < \infty$ *and zero otherwise.*

$$\psi_{r,L}(m, M) = \sum_{\bar{u} \in U_L(m,M)} \prod_{i=0}^{m-1} \binom{u_i+r-2}{r-2} \tag{4.20}$$

for $0 \leqq M < m$, $1 \leqq m < \infty$ *and zero otherwise.*

Proof. The proof is the same as for Lemmas 3.5 and 4.3. □

TABLE 4.2

$\bar{u}$	K(−)	K(+)
(1, 0, 0, 0, 2)	{1, 2, 3, 4, 5, 6, 7, 8}	{0, 9, 10}
	{1, 2, 3, 4, 5, 6, 7, 9}	{0, 8, 10}
	{1, 2, 3, 4, 5, 6, 7, 10}	{0, 8, 9}
(0, 1, 0, 0, 2)	{0, 1, 3, 4, 5, 6, 7, 8}	{2, 9, 10}
	{0, 1, 3, 4, 5, 6, 7, 9}	{2, 8, 10}
	{0, 1, 3, 4, 5, 6, 7, 10}	{2, 8, 9}
	{0, 2, 3, 4, 5, 6, 7, 8}	{1, 9, 10}
	{0, 2, 3, 4, 5, 6, 7, 9}	{1, 8, 10}
	{0, 2, 3, 4, 5, 6, 7, 10}	{1, 8, 9}
(0, 0, 1, 0, 2)	{0, 1, 2, 3, 5, 6, 7, 8}	{4, 9, 10}
	{0, 1, 2, 3, 5, 6, 7, 9}	{4, 8, 10}
	{0, 1, 2, 3, 5, 6, 7, 10}	{4, 8, 9}
	{0, 1, 2, 4, 5, 6, 7, 8}	{3, 9, 10}
	{0, 1, 2, 4, 5, 6, 7, 9}	{3, 8, 10}
	{0, 1, 2, 4, 5, 6, 7, 10}	{3, 8, 9}

4.2.3. Explicit evaluation of $\psi_{r,L}(m, M)$ and $\psi_{r,R}(m, M)$. To evaluate the summation in (4.19), partition the vectors in $U_R(m, M)$ according to the value s assumed by u_{m-1};

$$u_0+u_1+\cdots+u_i\begin{cases}\geqq i+1 & \text{if } 0\leqq i<m-2,\\ =M-s & \text{if } i=m-2,\\ =M & \text{if } i=m-1.\end{cases}$$

The value of s can vary in the range $0\leqq s\leqq M-m+1$. This provides the recurrence relation

$$\psi_{r,R}(m, M)=\sum_{s=0}^{M-m+1}\binom{r+s-2}{s}\psi_{r,R}(m-1, M-s), \tag{4.21}$$

$$m-1\leqq M<\infty, \qquad 2\leqq m<\infty.$$

To obtain a similar recurrence for $\psi_{r,L}(m, M)$, write the right-hand side of (4.20) as the difference

$$\sum_{\bar{u}\in U_L(m,M)}\prod_{i=0}^{m-1}\binom{u_i+r-2}{r-2}$$

$$=\sum_{\bar{u}\in U(m,M)}\prod_{i=0}^{m-1}\binom{u_i+r-2}{r-2}-\sum_{\bar{u}\in U(m,M)-U_L(m,M)}\prod_{i=0}^{m-1}\binom{u_i+r-2}{r-2}.$$

Next note that $\bar{u}\in U(m, M)-U_L(m, M)$ if and only if there is an s, $0\leqq s<M$ such that

(i) $(u_0, u_1, \cdots, u_{m-M+s-1})\in U_L(m-M+s, s)$,

(ii) $(u_{m-M+s}, u_{m-M+s+1}, \cdots, u_{m-1})\in U(M-s, M-s)$.

That is $s+m-M-1$ is the smallest value of i that does not satisfy (4.18). Hence

$$\psi_{r,L}(m, M)=\binom{M+(m-1)(r-1)}{M}-\sum_{s=0}^{M-1}\psi_{r,L}(m-M+s, s)\binom{(M-s)r-1}{M-s}, \tag{4.22}$$

$$0\leqq M<m, \qquad 1\leqq m<\infty.$$

To solve the recurrences (4.21)-(4.22) introduce the following generating functions

$$H_{r,R}(w,z)=\sum_{m=1}^{\infty} w^m \sum_{M=m-1}^{\infty} \psi_{r,R}(m,M)z^M, \tag{4.23a}$$

$$H_{r,L}(w,z)=\sum_{m=1}^{\infty} w^m \sum_{M=0}^{m-1} \psi_{r,L}(m,M)z^M. \tag{4.23b}$$

Multiply (4.21) by $w^m z^M$ and sum over $m-1\leqq M<\infty, 2\leqq m<\infty$; then, for the left-hand side

$$\sum_{m=2}^{\infty} w^m \sum_{M=m-1}^{\infty} \psi_{r,R}(m,M)z^M = H_{r,R}(w,z)-w\sum_{M=0}^{\infty} z^M \binom{M+r-2}{M}$$

$$= H_{r,R}(w,z)-\frac{w}{(1-z)^{r-1}}$$

and for the right-hand side we have

$$\sum_{m=2}^{\infty} w^m \sum_{M=m-1}^{\infty} z^M \sum_{s=0}^{M-m+1} \binom{s+r-2}{s} \psi_{r,R}(m-1,M-s).$$

Interchange the order of summation over s and M, move the range of summation for M from $[m-1+s,\infty)$ to $[m-1,\infty)$ to obtain for the right-hand side

$$\left(\sum_{m=2}^{\infty} w^m \sum_{M=m-1}^{\infty} z^M \psi_{r,R}(m-1,M)\right)\frac{1}{(1-z)^{r-1}}$$

$$=\left(wH_{r,R}(w,z)-\frac{w}{z}H_r(wz)\right)\frac{1}{(1-z)^{r-1}}.$$

Combining both sides and using (4.9) yields

$$H_{r,R}(w,z)=w\frac{1-wX^{r-1}(wz)}{(1-z)^{r-1}-w}. \tag{4.24}$$

To evaluate $H_{r,L}(w,z)$ write (4.22) in the form

$$\binom{M+(m-1)(r-1)}{M}=\sum_{s=0}^{M}\psi_{r,L}(m-M+s,s)\binom{(M-s)r-1}{M-s}, \tag{4.25}$$

$$0\leqq M<m, \qquad 1\leqq m<\infty,$$

multiply by $w^m z^M$ and sum over m and M: $0\leqq M<m, 1\leqq m<\infty$ to obtain for the left-hand side

$$\sum_{m=1}^{\infty} w^m \sum_{M=0}^{m-1} z^M \binom{M+(m-1)(r-1)}{M}. \tag{4.26a}$$

Interchange the ranges of summation on both m and M (twice reversing the order of summation) and use (4.8) to obtain

$$\sum_{m=0}^{\infty} w^{m+1} \sum_{M=0}^{\infty} (wz)^M \binom{Mr+m(r-1)}{M} = \frac{1}{r-(r-1)X(wz)} \sum_{m=0}^{\infty} w^{m+1} X^{m(r-1)+1}(wz)$$

$$=\frac{1}{r-(r-1)X(wz)}\frac{wX(wz)}{1-wX^{r-1}(wz)}.$$

For the right-hand side of (4.25)

$$\sum_{m=1}^{\infty} w^m \sum_{M=0}^{m-1} z^M \sum_{s=0}^{M} \psi_{r,L}(m-M+s, s) \binom{(M-s)r-1}{M-s} \tag{4.26b}$$

interchange the order of summation on s and M and move the ranges of indices to obtain

$$\begin{aligned}
&\sum_{m=0}^{\infty} w^{m+1} \sum_{s=0}^{\infty} (wz)^s \psi_{r,L}(m+s+1, s) \sum_{M=0}^{\infty} (wz)^M \binom{Mr-1}{M} \\
&= \frac{1}{r-(r-1)X(wz)} \sum_{m=0}^{\infty} w^{m+1} \sum_{s=0}^{\infty} (wz)^s \psi_{r,L}(m+1+s, s) \\
&= \frac{1}{r-(r-1)X(wz)} H_{r,L}(w, z).
\end{aligned}$$

Combining (4.26a) and (4.26b) yields the formula

$$H_{r,L}(w, z) = \frac{wX(wz)}{1-wX^{r-1}(wz)}. \tag{4.27}$$

From (4.24) and (4.27) we obtain the formulae

LEMMA 4.7. *For* $m-1 \leqq M < \infty$ *and* $1 \leqq m < \infty$,

$$\psi_{r,R}(m, M) = \sum_{i=m-1}^{M} \frac{r-1}{ir+r-1} \binom{ir+r-1}{i} \binom{(m-i-1)(r-1)+M-i-1}{M-i}. \tag{4.28}$$

For $0 \leqq M < m$ *and* $1 \leqq m < \infty$,

$$\psi_{r,L}(m, M) = \frac{(m-M-1)(r-1)+1}{(m-1)(r-1)+M+1} \binom{(m-1)(r-1)+M+1}{M}. \tag{4.29}$$

Proof. The proof simply consists in converting the generating functions to the corresponding power series and extracting the suitable coefficients. Thus

$$\begin{aligned}
H_{r,R}(w, z) &= w \frac{1-wX^{r-1}(wz)}{(1-z)^{r-1}-w} \\
&= \frac{w}{(1-z)^{r-1}} (1-wX^{r-1}(wz)) \sum_{i=0}^{\infty} \frac{w^i}{(1-z)^{i(r-1)}}.
\end{aligned}$$

Using the identity

$$\sum_{k=0}^{\infty} \binom{k+a}{k} t^k = \frac{1}{(1-t)^{a+1}}$$

(a special case of (4.8)), and the power expansion of $X^{r-1}(wz)$, the above is written as

$$\begin{aligned}
&\sum_{i=0}^{\infty} w^{i+1} \sum_{k=0}^{\infty} z^k \binom{k+(i+1)(r-1)-1}{k} \\
&\quad - \sum_{i=0}^{\infty} \sum_{k=0}^{\infty} \sum_{j=0}^{\infty} \frac{r-1}{jr+r-1} \binom{jr+r-1}{j} \binom{k+(i+1)(r-1)-1}{k} w^{i+j+2} z^{k+j}
\end{aligned} \tag{4.30}$$

and comparison with (4.23a) yields (4.28).

Equation (4.29) is an immediate application of (4.28) to

$$H_{r,L}(w, z) = \frac{wX(wz)}{1-wX^{r-1}(wz)} = \sum_{i=0}^{\infty} w^{i+1} X^{i(r-1)+1}(wz). \qquad \square$$

5. Tables containing a specified dummy key. We begin with some new notation. Let $\Gamma_r(n, N, t)$ be the set of partitions $\mathbf{K}$ of the integers $0, 1, \cdots, N-1$ with $n(r-1) \leqq N < nr$ which yield a table in which t is a dummy key (although not necessarily the only dummy key). Partition $\Gamma_r(n, N, t)$

$$\Gamma_r(n, N, t) = \bigcup_j \Gamma_r(n, N, t, j)$$

where $\Gamma_r(n, N, t, j)$ consists of those partitions $\mathbf{K}$ in $\Gamma_r(n, N, t)$ for which $k_{\eta_{K,j(r-1)}} = t$. Let

$$\gamma_r(n, N, t) = |\Gamma_r(n, N, t)|, \qquad \gamma_r(n, N, t, j) = |\Gamma_r(n, N, t, j)|.$$

5.1. Specifying the dummy key 0. Just as in § 4, we begin by considering the special case $t = 0$.

LEMMA 5.1. *For* $1 \leqq n < \infty$, $n(r-1) \leqq N < nr$

$$\gamma_r(n, N, 0, j) = 0 \quad \text{for } j \neq 0$$

and $\gamma_r(n, N, 0) = \gamma_r(n, N, 0, 0)$ *satisfies the recurrence relation*

$$\gamma_r(n, N, 0) = \sum_{k=1}^{N-n(r-1)+1} \gamma_r(n-k, N-kr+1, 0)\omega_r(k, 0) \tag{5.1}$$

where the function $\gamma_r(n, N, 0)$ *is given the boundary values*

$$\gamma_r(0, N, 0) = \begin{cases} 1 & \text{if } N = 0, \\ 0 & \text{if } N > 0. \end{cases}$$

With reference to Fig. 3.4, $i+m$ is the smallest dummy key in the memory state $\bar{z}'$. If its size $|\bar{z}'\}$ is n, the number of ways this table can arise is related to the function $\gamma_r(n, N, 0, j)$. The corresponding calculation for Fig. 3.3 will be given in § 5.2.

Proof. If $\bar{y} = \text{PINSERT}(\mathbf{K})$ with $y_0 = y_1 = 0$, then either $N = nr-1$, in which case there is a single nonzero term in (5.1)

$$\gamma_r(n, N, 0) = \gamma_r(n, nr-1, 0) = \omega_r(n, 0)$$

or $N < nr-1$ in which case there exists a smallest integer k for which $y_{(n-k)r} = y_{(n-k)r+1}$. In this latter case $\bar{y}$ decomposes into the two subtables

$$\bar{y}' = (y_0, y_1, \cdots, y_{(n-k)r-1}), \qquad \bar{y}'' = (y_{(n-k)r}, y_{(n-k)r+1}, \cdots, y_{nr-1}),$$
$$|\bar{y}'| = (n-k)r, \quad \|\bar{y}'\| = N-kr+1, \quad |\bar{y}''| = kr, \quad \|\bar{y}''\| = kr-1.$$

By Remark 3.9,

$$|\text{PINSERT}^{-1}(\bar{y})| = |\text{PINSERT}^{-1}(y_0, y_1, \cdots, y_{(n-k)r-1})| \cdot |\text{PINSERT}^{-1}(y_{(n-k)r}, y_{(n-k)+1}, \cdots, y_{nr-1})|$$

which gives (5.1). □

LEMMA 5.2.

$$G_r(w, z) = 1 + \sum_{n=1}^{\infty} w^n \sum_{N=n(r-1)}^{nr-1} \gamma_r(n, N, 0) z^N = \frac{1}{1 - wz^{r-1}X^{r-1}(wz^r)}, \tag{5.2}$$

$$\gamma_r(n, N, 0) = \frac{(nr-N)(r-1)}{N} \binom{N}{n(r-1)}. \tag{5.3}$$

Proof. Multiply (5.1) by $w^n z^N$ and sum $n(r-1) \leqq N < nr$, $1 \leqq n < \infty$,

$$G_r(w, z) - 1 = \sum_{n=1}^{\infty} w^n \sum_{N=n(r-1)}^{nr-1} \gamma_r(n, N, 0) z^N$$
$$= \sum_{n=1}^{\infty} w^n \sum_{N=n(r-1)}^{nr-1} z^N \sum_{k=1}^{N-n(r-1)+1} \gamma_r(n-k, N-kr+1, 0)\omega_r(k, 0).$$

The transformation

$$(n, N, k) \to (n-k, N-kr+1, k)$$

maps the set of triples (n, N, k) which satisfy the bounds of the last 3-fold summation into the Cartesian product

$$\{(n, N): 0 \leqq n < \infty, n(r-1) \leqq N \leqq nr\} \times \{k: 1 \leqq k < \infty\}$$

so that the change of variables $n' = n-k$, $N' = N-kr+1$ yields, when the right-hand side is summed, $G_r(w, z) - 1 = G_r(w, z) H_r(wz^r)$. Substituting from (4.9) produces (5.2), and using (4.8) contributes (5.3). □

5.2. Specifying the dummy key to be t. Next we consider the enumeration of partitions $\mathbf{K} = (\mathbf{K}(-), \mathbf{K}(+))$ of sizes

$$|\mathbf{K}(-)| = n(r-1), \qquad |\mathbf{K}(+)| = N - n(r-1)$$

of the integers $0, 1, \cdots N-1$ which yield a table in which t is a dummy key. For $1 \leqq n < \infty$ and $n(r-1) \leqq N < nr$, $\Gamma_r(n, N, t)$ is the set of partitions $\mathbf{K} = (\mathbf{K}(-), \mathbf{K}(+))$

$$\mathbf{K}(-) = \{k_0, k_1, \cdots k_{n(r-1)-1}\}, \qquad \mathbf{K}(+) = \{k_{n(r-1)}, k_{n(r-1)+1}, \cdots, k_{n-1}\}$$

of the integers $0, 1, \cdots, N-1$ which yield a table

$$\bar{y} = (y_0, y_1, \cdots, y_{nr-1}) = \text{PINSERT}(\mathbf{K})$$

in which t is a dummy key. Following a now familiar pattern, we partition $\Gamma_r(n, N, t)$

$$\Gamma_r(n, N, t) = \bigcup_j \Gamma_r(n, N, t, j)$$

where $\Gamma_r(n, N, t, j)$ is the subset of $\Gamma_r(n, N, t)$ consisting of the partitions $\mathbf{K} = (\mathbf{K}(-), \mathbf{K}(+))$ for which the dummy key $k_{\eta_{Kj(r-1)}}$ is equal to t. The above definition of $\gamma_r(n, N, t)$ and of $\gamma_r(n, N, t, j)$, implies the following boundary conditions:

$$\gamma_r(0, N, t) = \begin{cases} 1 & \text{if } N = t = 0, \\ 0 & \text{otherwise,} \end{cases} \qquad \gamma_r(0, N, t, j) = \begin{cases} 1 & \text{if } N = t = j = 0, \\ 0 & \text{otherwise.} \end{cases}$$

The parameters n, N, t and j are constrained by the conditions

(5.4a) $$0 \leqq j < n < \infty,$$

(5.4b) $$n(r-1) \leqq N < nr,$$

(5.4c) $$j(r-1) \leqq t \leqq jr,$$

(5.4d) $$t \leqq N - (n-j)(r-1).$$

This region in the (n, N, t, j) space will be often used in the sequel and will be denoted by $\Delta_r(n, N, t, j)$.

Comment. The conditions imposed in (5.4a), (5.4b) are obvious; the condition of (5.4c) is just a restatement of (4.10). The final condition follows from the nonnegativity of $|\{k_i \in \mathbf{K}(+): k_i > t\}|$

$$\begin{aligned} |\{k_i \in \mathbf{K}(+): k_i > t\}t\}| &= |\{t+1, t+2, \cdots, N-1\}| - |\{k_{\eta_{K,s}}: j(r-1) < s < n(r-1)\}| \\ &= (N-t-1) - ((n-j)(r-1)-1 \\ &= N - (n-j)(r-1) - t. \end{aligned}$$

To evaluate $\gamma_r(n, N, j)$, partition $\Gamma_r(n, N, t, j)$ into three sets

$$\Gamma_r(n, N, t, j) = \bigcup_{i=1}^{3} \Gamma_{r,i}(n, N, t, j),$$

$$\gamma_{r,i}(n, N, t, j) = |\Gamma_{r,i}(n, N, t, j)|, \qquad i = 1, 2, 3$$

defined as follows.

1	$\Gamma_{r,1}(n, N, t, j)$
	t is the only dummy key

Note that $\Gamma_{r,1}(n, N, t, j) = \varnothing$ unless $N = nr - 1$. We have already obtained in Lemma 4.4 the value of $\gamma_{r,1}(n, N, t, j) = \omega_r(n, t, j)$.

2	$\Gamma_{r,2}(n, N, t, j)$
	At least one dummy key with value $< t$

Note that $\Gamma_{r,2}(n, N, t, j) \neq \varnothing$ requires $N < nr - 1$. The partition in $\Gamma_{r,2}(n, N, t, j)$ corresponds to the memory state $\bar{z} = (z_0, z_1, \cdots, z_{nr-1})$

$\cdots$	$t-kr+1$	$t-kr+1$	$t-kr+2$	$\cdots$	$t-1$	t	t	$\cdots$
	$(j-k)r$	$(j-k)r+1$	$(j-k)r+2$		$jr-1$	jr	$jr+1$	

which may be decomposed into the two substates

$$\bar{z}' = (z_{(j-k)r}, z_{(j-k)r+1}, \cdots, z_{jr-1}), \qquad \bar{z}'' = (z_0, z_1, \cdots, z_{(j-k)r-1}, z_{jr}, z_{jr+1}, \cdots, z_{nr-1})$$

of sizes kr and $(n-k)r$, for some value k. The state $\bar{z}'$ is characterized by

(i) $z_{(j-k)r}$ is a dummy key with value $t-kr+1$;

(ii) $z_{(j-k)r+1}, z_{(j-k)r+2}, \cdots, z_{jr-1}$ are genuine keys with values $t-kr+1, t-kr+2, \cdots, t-1$,

while the state $\bar{z}''$ satisfies

(i) z_{jr} is a dummy key with the value t;

(ii) z_{jr+1} is a genuine key with value t.

Furthermore, requiring $\bar{z}''$ to arise from partitions defined by values in $\Delta_r(n-k, N-kr+1, t-kr+1, j-k)$ implies the following constraints on admissible values of k, corresponding to (5.4a)-(5.4d):

(5.5a) $$1 \leqq k \leqq j-n,$$

(5.5b) $$(n-k)(r-1) \leqq N-kr+1 < (n-k)r,$$

(5.5c) $$(j-k)(r-1) \leqq t-kr+1 \leqq (j-k)r,$$

(5.5d) $$t-kr+1 \leqq N-kr+1-(n-j)(r-1).$$

We now use Remark 3.9 to conclude that

$$\gamma_{r,2}(n, N, t, j) = \sum_{\{k:\,(5.5a-d)\}} \omega_r(k, 0)\gamma_r(n-k, N-kr+1, t-kr+1, j-k)$$

where the range of summation is determined by the conditions in (5.5a)-(5.5d).

3	$\Gamma_{r,3}(n, N, t, j)$
At least one dummy key different from t All keys with values <t are genuine	

Note that $\Gamma_{r,3}(n, N, t, j) \neq \varnothing$ requires $N < nr-1$. The partition in $\Gamma_{r,3}(n, N, t, j)$ corresponds to the memory state

$$\bar{z}' = (z_0, z_1, \cdots, z_{nr-1})$$

$N-jr+t$	$\cdots$	$N-2$	$N-1$	0	$\cdots$	$t-2$	$t-1$	t	t	$\cdots$	$N-jr+t-1$
0		$jr-t-2$	$jr-t-1$	$jr-t$		$jr-2$	$jr-1$	jr	$jr+1$		$nr-1$

which decomposes into the two substates

$$\bar{z}' = (z_0, z_1, \cdots, z_{jr-1}, z_{(n-k)r}, z_{(n-k)r+1}, \cdots, z_{nr-1}),$$

$$\bar{z}'' = (z_{jr}, z_{jr+1}, \cdots, z_{(n-k)r-1})$$

of sizes jr and $(n-j)r$; these states are characterized by

(i) $z_{jr-1}, z_{jr-2}, \cdots, z_{jr-t}$ are genuine keys with values $t-1, t-2, \cdots, 0$.

(ii) $z_{jr-t-1}, z_{jr-t-2}, \cdots, z_0$ are genuine keys, with values $N-1, N-2, \cdots, N-jr+t$.

(iii) $z_{nr-1}, z_{nr-2}, \cdots, z_{(n-k)r+1}$ are genuine keys, with values $N-jr+t-1$, $N-jr+t-2, \cdots, N-jr+t-kr+1$.

(iv) $z_{(n-k)r}$ is a dummy key with value $N-(j+k)r+t+1$.

(v) $z_{jr} = z_{jr+1} = t$.

The admissible values for k are constrained by the conditions

$$1 \leqq k < n-j, \tag{5.6a}$$

$$k \leqq N-n(r-1)-j+1. \tag{5.6b}$$

Condition (5.6a) is clear, while condition (5.6b) follows from a count of the genuine keys in $\bar{z}$; in $\bar{z}'$ there are $(j+k)r-1$ genuine keys and in $\bar{z}''$ at least $(n-k-j)(r-1)$ keys, so that $N \geqq n(r-1)+k+j-1$. Note that (5.6b) implies (5.6a).

Using Remark 3.9 we observe that

$$\begin{aligned}\gamma_{r,3}(n, N, t, j) &= \sum_k \omega_r(k+j, t, j)\gamma_r(n-(k+j), N-(k+j)r+1, 0) \\ &= \sum_k \psi_{r,R}(k, k-1+jr-t)\psi_{r,L}(j+1, t-j(r-1)) \\ &\quad \cdot \gamma_r(n-(k+j), N-(k+j)r+1, 0)\end{aligned}$$

where the factor $\omega_r(k+j, t, j)$ counts

(i) the number of partitions yielding a table with memory state $\bar{z}'$ satisfying

$$|\bar{z}'| = (k+j)r, \qquad \|\bar{z}'\| = (k+j)r-1$$

while the second factor, $\gamma_r(n-(k+j), N-(k+j)r+1, 0)$, counts

(ii) the number of partitions yielding a table with memory state $\bar{z}''$

$$|\bar{z}''| = (n-k-j)r, \qquad \|\bar{z}'\| = N-(k+j)r+1$$

in which the "smallest" key is a dummy key.

When $k=n-j$, the term $\omega_r(n, t, j)\gamma_r(0, N-nr+1, 0)$ reduces to $\gamma_{r,1}(n, N, t, j)$. Thus, using Lemma 4.6, we have proved that $\gamma_r(n, N, t, j)$ satisfies the recurrence

$$\begin{aligned}\gamma_r(n, N, t, j) &= \sum_{k=1}^{j} \omega_r(k, 0)\gamma_r(n-k, N-kr+1, t-kr+1, j-k) \\ &\quad + \sum_{k=1}^{N-n(r-1)-j+1} \omega_r(k+j, t, j)\gamma_r(n-(k+j), N-(k+j)r+1, 0).\end{aligned} \tag{5.7}$$

Next, introduce the generating function

$$G_r(w, z, v, u) = 1 + \sum_{\Delta_r(n, N, t, j)} w^n z^N v^t u^j \gamma_r(n, N, t, j)$$

where Δ_r is the set of 4-tuples (n, N, t, j) that satisfy (5.4a)-(5.4d). Equation (5.7) translates into a relationship for the generating function $G_r(w, z, v, u)$; the details are given in Appendix 2.

From (5.7) we may also write

$$G_r(w, z, v, u) = 1 + G_r^{(1)}(w, z, v, u) + G_r^{(2)}(w, z, v, u),$$

where the first addend is shown in (A2.2) ((2) in Appendix 2) to be

$$G_r^{(1)}(w, z, v, u) = \frac{1}{zv} H_r(uw(zv)^r)(G_r(w, z, v, u) - 1).$$

Hence

$$G_r(w, z, v, u) = 1 + \frac{G_r^{(2)}(w, z, v, u)}{1-(zv)^{-1}H_r(uw(zv)^r)}.$$

Using now (A2.3) one can obtain an explicit expression for $\gamma_r(n, N, t, j)$ by "convolving" the parts of this last relation, though no closed form seems likely.

LEMMA 5.3.

$$\begin{aligned}G_r(w, z, v, u) &= 1 + \frac{X^{r-1}(wz^r) - uv^{r-1}X^{r-1}(uw(zv)^r)}{1-u[1+(v-1)X(uw(zv)^{r-1})]^{r-1}} \\ &\quad \cdot \frac{1}{1-wz^{r-1}X^{r-1}(wz^r)} \cdot \frac{wz^{r-1}X(uw(zv)^r)}{1-uw(vz)^{r-1}X^{r-1}(uw(zv)^r)}\end{aligned} \tag{5.8}$$

and

$$\begin{aligned}&\gamma_r(n, N, t, j) \\ &\quad = \sum_{l,k,i} \alpha(k, i)\psi_{r,R}(l, l-1+jr-k-t)\psi_{r,L}(j-i-k+1, t-i-j(r-1)) \\ &\qquad \cdot \gamma_r(n-j-l, N+k-(l+j)r+1, 0)\end{aligned} \tag{5.9}$$

with the ranges of summation implied by the vanishing of these functions, resulting in

$$1 \leqq l \leqq n-j-1, \qquad 0 \leqq i \leqq t-j(r-1),$$
$$n(r-1)+j+l-N-1 \leqq k \leqq \min\{jr-t, nr-N-2\}$$

and

$$\alpha(k, i) = \begin{cases} 1 & \text{if } k=0, \\ \dfrac{k(r-1)}{ir+k(r-1)}\binom{ir+k(r-1)}{i} & \text{if } k>0. \end{cases}$$

As we shall later show, only the "marginal" generating function $G_r(w, z, 1, 1)$ is needed to evaluate the cost of insertions and not $G_r(w, z, v, u)$. There does not seem

however to be any easy way of directly evaluating the marginal generating function and this has required the previous calculation.

6. The number of moves required to insert a key. We make the following suppositions:

(i) The table $\bar{Y}=(Y_0, Y_1, \cdots, Y_{nr-1})$ resulted from the insertion of N keys $X_0, X_1, \cdots, X_{N-1}$ with $N<nr$ using the padded table scheme defined in § 2.

(ii) An $(N+1)$st key X_N is to be inserted into $\bar{Y}$.

Let $\bar{\Pi}=(\Pi_0, \Pi_1, \cdots, \Pi_N)$ be the permutation of $0, 1, \cdots, N$ which sorts the insertion sequence $X_0, X_1, \cdots, X_N$

$$X_{\Pi_0}<X_{\Pi_1}<\cdots<X_{\Pi_N}.$$

According to Remark 2.1, we may assume the key values $X_0, X_1, \cdots, X_N$ to be the integers $0, 1, \cdots, N$. Our probabilistic model of key insertion postulates that

(6.1) *each of the* $(N+1)!$ *permutations* $\bar{\Pi}$ *is equally likely.*

Let $M_r(n, N)$ denote the (random) number of moves needed to insert X_N into the table

$$\bar{Y}=\text{INSERT}\,(X_0, X_1, \cdots, X_{N-1}).$$

To calculate the conditional probability

$$\Pr\{M_r(n, N)=m|X_N=i\}$$

we need to count the number of partitions $\mathbf{K}=(\mathbf{K}(-), \mathbf{K}(+))$ of the integers

$$0, 1, \cdots, i-1, i+1, \cdots, N$$

for which the state $\bar{y}=\text{PINSERT}\,(\mathbf{K})$ satisfies:

(i) The genuine keys with the values

$$i+1, i+2, \cdots (i+m) \text{ (modulo } N+1)$$

are contiguous in $\bar{y}$, and

(ii) A dummy key with value $(i+m+1)$ (modulo $N+1$) is contained in $\bar{y}$.

$\cdots$	$i+1$	$i+2$	$\cdots$	$i+m$	$i+m+1$	$i+m+1$	$\cdots$

Clearly this is the same as the number of partitions $\mathbf{K}=(\mathbf{K}(-), \mathbf{K}(+))$ of the integers $0, 1, \cdots, N-1$ for which PINSERT $(\mathbf{K})$ satisfies:

(i′) The genuine keys with the values

$$i, i+1, \cdots, (i+m-1) \text{ (modulo } N)$$

are contiguous in $\bar{y}$, and

(ii′) A dummy key with value $(i+m)$ (modulo N) is contained in $\bar{y}$.

$\cdots$	i	$i+1$	$\cdots$	$i+m-1$	$i+m$	$i+m$	$\cdots$

We already have counted such partitions in § 5; denoting their number by $\lambda_r(n, N, m, i)$, we can conclude that

$$\begin{aligned}\lambda_r(n, N, m, i)=&\sum_{\{k,j:\ kr>m+1\}}\omega_r(k, 0)\gamma_r(n-k, N-kr+1, t-kr+1, j-k)\\ &+\sum_{\{k,j:\ (j+k)r>m+1\}}\psi_{r,R}(k, k-1+jr-t)\psi_{r,L}(j+1, t-j(r-1))\\ &\cdot\gamma_r(m-(k+j), N-(k+j)r+1, 0)\end{aligned}\tag{6.2}$$

where $t=(i+m)$ (modulo N). We note that $\lambda_r(n, N, m, i)$ is not independent of i, and $\lambda_r(n, N, m, 0)=\lambda_r(n, N, m, N)$.

We remark that the first of these observations is the source of most of the intricacy of the required analysis. Removing the conditioning yields the probability distribution for the number of moves

$$\Pr\{M_r(n, N)=m\}=\frac{1}{N+1}\frac{1}{\binom{N}{n(r-1)}}\sum_{i=0}^{N}\lambda_r(n, N, m, i). \tag{6.3}$$

This suggests that we consider the counting function

$$\Lambda_r(n, N, m)=\sum_{i=0}^{N}\lambda_r(n, N, m, i) \tag{6.4}$$

which apart from the normalization factor of $(N+1)\binom{N}{n(r-1)}$ is the probability of the event $\{M_r(n, N)=m\}$.

Introduce the generating function

$$L_r(w, z, y)=\sum_{n=1}^{\infty}w^n\sum_{N=n(r-1)}^{nr-1}z^N\sum_{m=0}^{N-1}y^m\Lambda_r(n, N, m). \tag{6.5}$$

The expected number of moves needed to insert the $(N+1)$st key has the generating function

$$\frac{1}{N+1}\frac{1}{\binom{N}{n(r-1)}}\frac{\partial}{\partial y}L_r(w, z, y)\Big|_{y=1}.$$

The computation of this generating function and its derivative is laborious, albeit straightforward, and is carried out in Appendix 3. The result is

THEOREM 6.1.

$$\begin{aligned}E\{M_r(n, N)\}=&\frac{1}{2}\chi_{\{N=nr-1\}}\left[N-1-\frac{r-2}{N-n}\right]\\&+\frac{(r-1)(nr-N-1)}{2(N+1)}\sum_{i\geq 0}\frac{1}{N-ir-r+1}\binom{N-ir-r+1}{N-n(r-1)-i}\\&\cdot\frac{1}{\binom{N}{n(r-1)}}\left[(2-r)\binom{ir+r}{i}+(N+1)(ir+r-2)\binom{ir+r-1}{i}\right]\end{aligned} \tag{6.6}$$

where $\chi_{\{E\}}$ is the indicator function of the event E. Note that the range of the sum in (6.6) *is finite (up to $(n-1)(r-1)$ at most). See Appendix* 3, (A3.6b), (A3.18) *and* (A3.22) *for the distribution of $M_r(n, N)$.*

THEOREM 6.2. *When $(r-1)/r\leqq\rho<1$*

$$\lim_{n\to\infty}E\{M_r(n, n\rho r)\}=\frac{\rho}{2(r-1)(1-\rho)^2}-1. \tag{6.7}$$

THEOREM 6.3. *When $(r-1)/r\leqq\rho<1$*

$$\begin{aligned}&\lim_{n\to\infty}\Pr\{M_r(n, n\rho r)=m\}\\&\quad=\frac{1-\rho}{\rho}-\frac{(1-\rho)r}{1-r+\rho r}\sum_{i=1}^{\lfloor(m+1)/r\rfloor}\frac{1}{ir-1}\binom{ir-1}{i}\left(\frac{(r-1^{r-1}(1-r+\rho r)}{(\rho r)^r}\right)^i\end{aligned} \tag{6.8}$$

and a value for the tail of the distribution can also be produced:

$$\lim_{n\to\infty} \Pr\{M_r(n, n\rho r) \geqq s\} = \frac{1+sr(r-1)(1-\rho)}{1-r+\rho r} - \frac{(1-\rho)r}{1-r+\rho r} \sum_{i=0}^{\lfloor (s+1)/r \rfloor} \frac{ir-s-1}{ir-1} \binom{ir-1}{i} \cdot \left(\frac{(r-1)^{r-1}(1-r+\rho r)}{(\rho r)^r} \right)^i. \tag{6.9}$$

Remark 6.4. The padded list with $r_k = 2$ reduces to the sparse table with $m_k = 2$ discussed in [4].

6.1. A regularity phenomenon. Equation (6.8) shows that the limiting probability

$$\pi_r(m) = \lim_{n\to\infty} \Pr\{M_r(n, n\rho r) = m\}$$

enjoys the following properties:

$$\pi_r(0) = \pi_r(1) = \cdots = \pi_r(r-2), \tag{6.10a}$$

$$\pi_r(kr-1) = \pi_r(kr) = \cdots = \pi_r((k+1)r-2), \qquad 1 \leqq k < \infty. \tag{6.10b}$$

This property may be explained by the following observation: Suppose the keys $X_1, X_2, \cdots, X_{N-1}$ which are inserted into a padded list $\bar{Y} = \text{INSERT}(X_1, X_2, \cdots, X_{N-1})$ are allowed to have rational values. The probability of each of the $N+1$ events

$$E_0: \quad X_N < X_0 < X_1 < \cdots < X_{N-1},$$

$$E_i: \quad X_0 < X_1 < \cdots < X_{i-1} < X_N < X_i < \cdots < X_{N-1}, \qquad 0 < i < N,$$

$$E_N: \quad X_0 < X_1 < \cdots < X_{N-1} < X_N$$

is $1/(N+1)$. The probability of the event $\{M_r(n, N) = m\}$ can be written as

$$\Pr\{M_r(n, N) = m\} = \frac{1}{N+1} \sum_{i,\{\bar{z}\}} \Pr\{M_r(n, N) = m \quad \text{and} \quad E_i | \bar{z}\} \times \Pr\{\bar{z}\}$$

where $\bar{z}$ denotes the memory state corresponding to $\text{INSERT}(X_0, X_1, \cdots, X_{N-1})$. The skeleton σ of the memory state $\bar{z}$ is of one of two types:

Case 1: $\sigma = (0, (1)^{m_1 r-1}, \cdots, 0, (1)^{m_{s-1} r-1}), \qquad \text{TO} = 0,$

Case 2: $\sigma = ((1)^{m_{0.+}+m_{0.-}}, 0, (1)^{m_1 r-1}, \cdots, 0, (1)^{m_{s-1} r-1}), \qquad \text{TO} = m_{0.+} > 0.$

In Case 1, let $L_i (i \geqq 1)$ equal the number of blocks of genuine keys of length $ir-1$. Note that

$$N = \sum_i (ir-1) L_i, \qquad \sum_i L_i = nr - N.$$

It is simple to see that

$$\Pr\{M_r(n, N) = m | \bar{z}\} = \begin{cases} \dfrac{1+\sum_i L_i}{N+1} & \text{if } m = 0, \\ \dfrac{\sum_{\{i:\, ir-1 > m\}} L_i}{N+1} & \text{if } m > 0. \end{cases}$$

Thus $\lim_{n\to\infty} \Pr\{M_r(n, n\rho r) = m | \bar{z}\}$ satisfies (6.10a)-(6.10b) for state $\bar{z}$ corresponding to Case 1.

In Case 2, let L_i denote the number of blocks of genuine keys with the skeleton $(0, (1)^{m_j r-1})$ of length $ir-1$ with $1 \leqq j < s-1$. Then

$$\Pr\{M_r(n, N) = m|\bar{z}\} = \begin{cases} \dfrac{1+\sum_i L_i}{N+1} & \text{if } m=0, \\ \dfrac{\chi_{\{0<m\leqq m_{0,-}+m_{0,+}+m_{s-1}(r-1)\}} + \sum_{\{i:\, ir-1>m\}}^{L_i}}{N+1} & \text{if } m>0 \end{cases}$$

where $\chi_{\{E\}}$ is the indicator function of the event E. Thus

$$\lim_{n\to\infty} \Pr\{M_r(n, n\rho r) = m|\bar{z}\}$$

satisfies (6.10a), (6.10b) for states $\bar{z}$ belonging to Case 2 as well.

7. Numerical results. To illustrate convergence to the limiting value, we plot $E\{M_r(n, n\rho r)\}$ versus n for $r=2(1)5$ for fixed utilization ρ in Figs. 7.1-7.4.

In Fig. 7.5, we plot $\lim_{n\to\infty} E\{M_r(n, n\rho r)\}$ versus the utilization ρ for $r=2(1)5$.

In Figs. 7.6-7.9, we plot $\lim_{n\to\infty} \Pr\{M_r(n, n\rho r) \leqq m\}$ as a function of the utilization ρ for $r=2(1)5$.

Appendix 1.

LEMMA A1.1. *For an integer $r \geqq 2$ and $|z| < (r-1)^{r-1}/r^r$, the equation*

$$(*) \qquad zx^r = x-1$$

has a unique root $X(z)$ which is

- *analytic in z in the open disk*

$$D(0, (r-1)^{r-1}) = \left\{z: |z-0| < \frac{(r-1)^{r-1}}{r^r}\right\},$$

- *takes the value 1 at $z=0$.*

Moreover, if $f(z)$ is analytic in the open disk $D(1, 1/(r-1))$, then

$$f(X(z)) = f(1) + \sum_{i=1}^{\infty} \frac{z^i}{i!} \frac{d^{i-1}}{du^{i-1}} \left[u^{ir} \frac{d}{du} f(u)\right]\Big|_{u=1}. \tag{A1.1}$$

Proof. For each z there are r roots of the equation $zx^r = x-1$. When $|z| < (r-1)^{r-1}/r^r$, the inequality $|zx^r| < |x-1|$ is satisfied for all x on the boundary of the disk $D(1, 1/(r-1))$. Rouché's theorem [1] then implies that equation $(*)$ has a unique root $X(z)$ in the interior of this disk. Since z^r is analytic in and on $D(1, 1/(r-1))$, (A1.1) follows from Lagrange's theorem [8].

Setting $f(z) = z^s$ (with s real), we obtain the useful expansion

$$X^s(z) = \sum_{i=0}^{\infty} z^i \frac{s}{ir+s} \binom{ir+s}{i}. \tag{A1.2}$$

Replacing z by z^r in (A1.2) and differentiating, we obtain

$$X^s(z^r) + rz^r X^{s-1}(z^r) \frac{d}{d\zeta} X(\zeta)\Big|_{\zeta=z^r} = \sum_{i=0}^{\infty} z^{ir} \binom{ir+s}{i}.$$

The defining relation $z^r X^r(z^r) = X(z^r) - 1$ yields upon differentiation

$$\frac{d}{d\zeta} X(\zeta)\Big|_{\zeta=z^r} = \frac{X^{r+1}(z^r)}{r-(r-1)X(z^r)} \tag{A1.3}$$

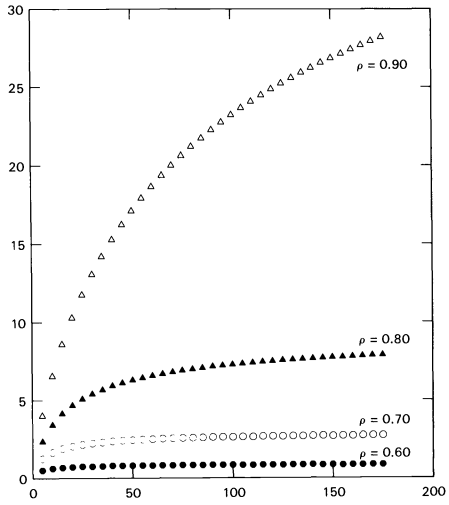


FIG. 7.1. *Expected number of moves to insert a record into a table of density* ρ: $r = 2$.

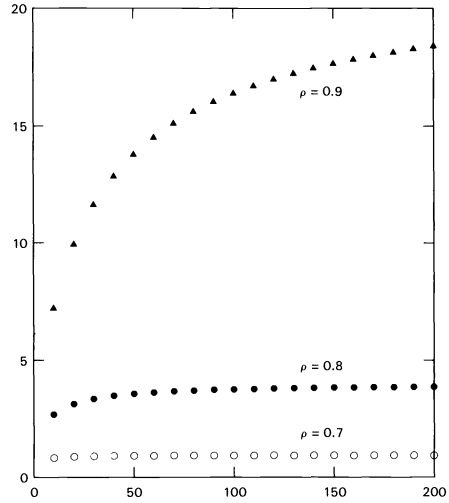


FIG. 7.2. *Expected number of moves to insert a record into a table of density* ρ: $r = 3$.

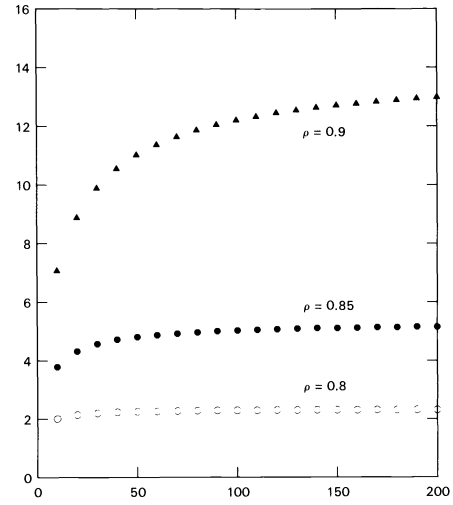


FIG. 7.3. *Expected number of moves to insert a record into a table of density* ρ: $r = 4$.

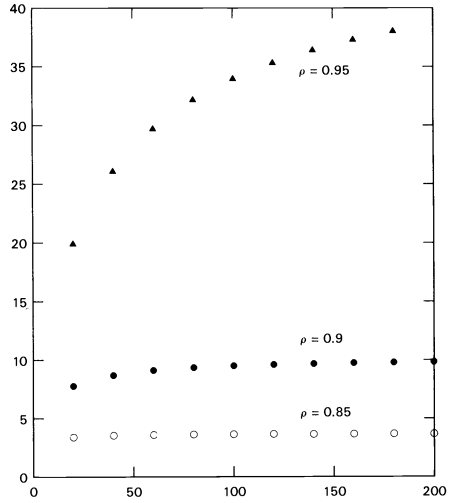


FIG. 7.4. *Expected number of moves to insert a record into a table of density* ρ: $r = 5$.

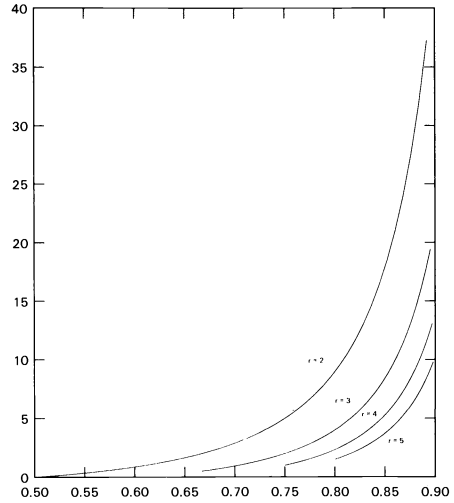


FIG. 7.5. *Limiting value for the expected number of moves as a function of density.*

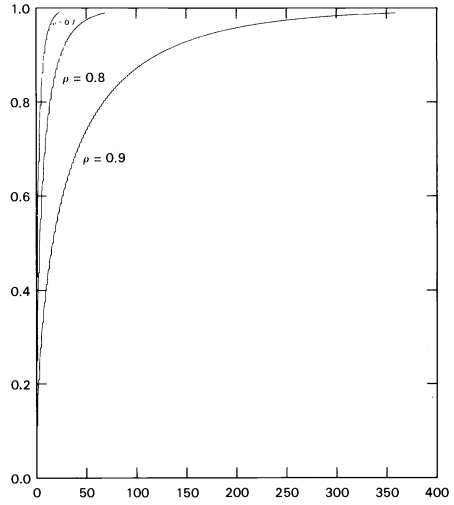


FIG. 7.6. *Limiting value of cumulative distribution function*: $r = 2$.

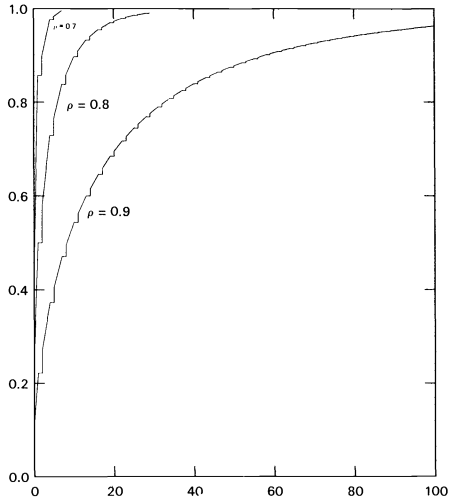


FIG. 7.7. *Limiting value of cumulative distribution function*: $r = 3$.

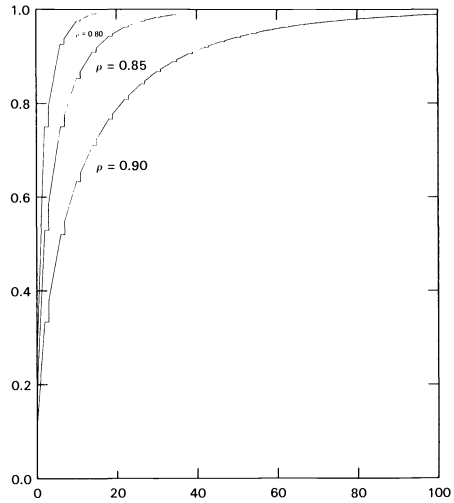


FIG. 7.8. *Limiting value of cumulative distribution function*: $r = 4$.

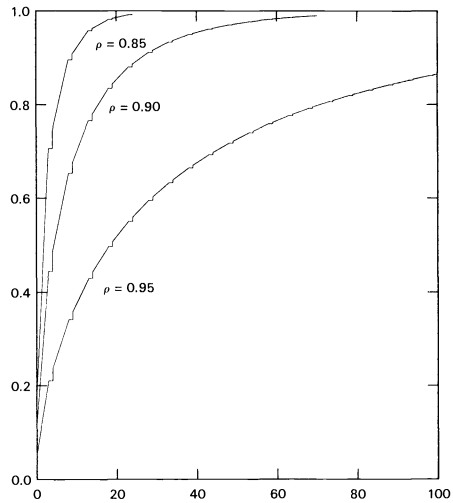


FIG. 7.9. *Limiting value of cumulative distribution function*: $r = 5$.

which together with the previous expansion gives

$$Q_{r,s}(z)=\sum_{i=0}^{\infty} z^i \binom{ir+s}{i}=\frac{X^{s+1}(z)}{r-(r-1)X(z)}. \tag{A1.4}$$

We note that (A1.2) and (A1.4) are also valid for $s=-1$.

Appendix 2. We start with the recurrence relation which is equivalent to (5.7).

$$\begin{aligned}\gamma_r(n, N, t, j)&=\sum_{k=1}^{j}\omega_r(k,0)\gamma_r(n-k, N-kr+1, t-kr+1, j-k)\\ &\quad+\sum_{k=1}^{n-j}\psi_{r,R}(k, k-1+jr-t)\psi_{r,L}(j+1, t-j(r-1))\\ &\qquad\times\gamma_r(n-(k+j), N-(k+j)r+1, 0)\end{aligned} \tag{A2.1}$$

for $1\leqq n<\infty$. Multiply each of the terms of (A2.1) by $w^n z^N v^t u^j$ and sum over Δ_r, the set of 4-tuples which satisfy (5.4a)-(5.4d). The left-hand side is $G_r(w, z, v, u)-1$ while the right-hand side is the sum of two terms, the evaluation of which we now turn to.

The First Term in (A2.1)

$$G_{r,1}(w, z, v, u)=\sum_{(n,N,t,j)\in\Delta_r}\sum_{k} z^N w^n v^t u^j \omega_r(k,0)\gamma_r(n-k, N-kr+1, t-kr+1, j-k).$$

Now make the change of variables

$$n'=n-k,\quad N'=N-kr+1,\quad j'=j-k,\quad t'=t-kr+1$$

to obtain

$$\begin{aligned}G_{r,1}=G_{r,1}(w, z, v, u)&=\frac{1}{zv}\sum_{k}((zv)^r wu)^k\omega_r(k,0)\\ &\quad\cdot\sum_{(n,N,t,j)\in\Delta_r} z^N w^n v^t u^j \gamma_r(n, N, t, j)\\ &=\frac{1}{zv}H_r(uw(zv)^r)(G_r(w, z, v, u)-1).\end{aligned} \tag{A2.2}$$

The Second Term in (A2.1)

$$G_{r,2}=G_{r,2}(w, z, v, u)=\sum_{n=1} w^n \sum_{N=n(r-1)}^{nr-1} z^N \sum_{j=0}^{n-1} u^j \sum_{t=j(r-1)}^{ir} v^t$$

$$\cdot\left[\sum_{k=1}^{n-j}\psi_{r,R}(k, k-1+jr-t)\psi_{r,L}(j+1, t-j(r-1))\gamma_r(n-(k+j), N-(k+j)r+1, 0)\right].$$

Make the change of variables

$$n'=n-k-j,\qquad N'=N-(j+k)r+1$$

and use (5.2)

$$G_{r,2}=\frac{1}{z}G_r(w,z)\sum_{k=1}^{\infty}(wz^r)^k\sum_{j=0}^{\infty}(uwz^r)^j \times\sum_{t=j(r-1)}^{ir}v^t\psi_{r,R}(k,k-1+jr-t)\psi_{r,L}(j+1,t-j(r-1)).$$

Now make the change of variables $t'=t-j(r-1)$ to get

$$G_{r,2}=\frac{1}{z}G_r(w,z)\sum_{k=1}^{\infty}(wz^r)^k\sum_{j=0}^{\infty}(uwz^rv^{r-1})^j \times\sum_{t=0}^{j}v^t\psi_{r,R}(k,k-1+jr-t)\psi_{r,L}(j+1,t).$$

Interchanging the order of summation over j and t, and then substituting $j'=j-t$, we obtain

$$G_{r,2}=\frac{1}{z}G_r(w,z)\sum_{k=1}^{\infty}(wz^r)^k\sum_{t=0}^{\infty}(uw(zv)^r)^t\sum_{j=0}^{\infty}(uwz^rv^{r-1})^j \times\psi_{r,R}(k,k-1+j)\psi_{r,L}(j+t+1,t).$$

This multiple summation may be evaluated in terms of the generating functions $H_{r,R}(w,z)$ and $H_{r,L}(w,z)$ as follows: First

$$H_{r,L}(w,z)=\sum_{j=1}^{\infty}w^j\sum_{t=0}^{j-1}z^t\psi_{r,L}(j,t)$$

change the order of summation and modify the range of j to obtain

$$H_{r,L}(w,z)=\sum_{j=0}^{\infty}w^{j+1}\sum_{t=0}^{\infty}(wz)^t\psi_{r,L}(j+t+1,t).$$

Writing (4.27) as

$$H_{r,L}(z,w)=\sum_{j=0}^{\infty}w^{j+1}X^{1+j(r-1)}(wz)$$

implies

$$X^{1+j(r-1)}(wz)=\sum_{t=0}^{\infty}(wz)^t\psi_{r,L}(j+t+1,t)$$

so that

$$G_{r,2}=\frac{1}{z}G_r(w,z)X(uw(zv)^r)\sum_{k=1}^{\infty}(wz^r)^k\sum_{j=0}^{\infty}[wuz^rv^{r-1}X^{r-1}(wu(zv)^r)]^j\psi_{r,R}(k,k-1+j).$$

Next

$$H_{r,R}(w,z)=\sum_{k=1}^{\infty}w^k\sum_{j=k-1}^{\infty}z^j\psi_{r,R}(k,j) =\frac{1}{z}\sum_{k=1}^{\infty}(wz)^k\sum_{j=0}^{\infty}z^j\psi_{r,R}(k,k-1+j)$$

so that

$$G_{r,2}(w,z,v,u)=\frac{1}{z}X(uw(zv)^r)G_r(w,z)\beta H_{r,R}(\alpha,\beta)$$

where

$$\alpha\beta = wz^r, \qquad \beta = uwz^r v^{r-1} X^{r-1}(uw(zv)^r).$$

Using (4.24) and (4.7a) we obtain

$$\begin{aligned} G_{r,2}(w, z, v, u) &= wz^{r-1} X(uw(zv)^r) G_r(w, z) \\ &\quad \cdot \frac{X^{r-1}(wz^r) - uv^{r-1} X^{r-1}(uw(zv)^r)}{1 - uv^{r-1} X^{r-1}(uw(zv)^r)[1 - uwz^r v^{r-1} X^{r-1}(uw(zv)^r)]^{r-1}} \\ &= wr^{r-1} G_r(w, z) X(uw(zv)^r) \\ &\quad \cdot \frac{X^{r-1}(wz^r) - uv^{r-1} X^{r-1}(uw(zv)^r)}{1 - u[1 + (v-1)X(uw(zv)^r)]^{r-1}}. \end{aligned} \tag{A2.3}$$

Combining the left-hand side and the two partial functions $G_{r,i}$ we obtain (5.8).

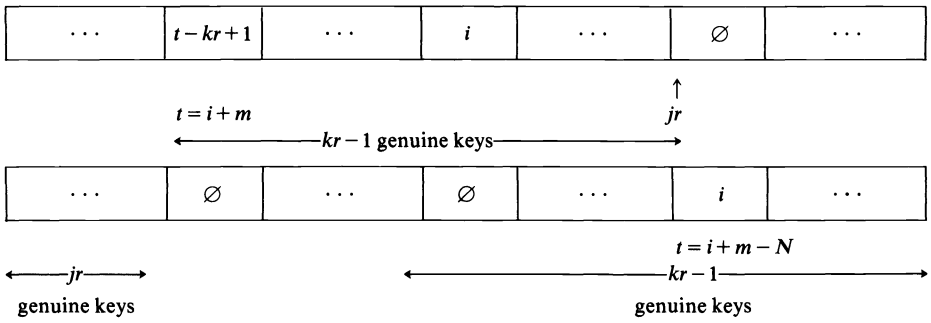
Appendix 3. The key observation in enumerating moves caused by the insertion of a key with the value i is the following:

- each of the values $0, 1, \cdots, m-1, m+1, \cdots, N-1$ in the range of the function

$$i \to (i+m) \text{ (modulo } N), \qquad i = 0, 1, \cdots, N$$

  is taken with multiplicity one;
- the value m is taken with multiplicity two.

The following two diagrams represent a "simple" and a "wraparound" move, respectively. The key values are later to be translated, as explained preceding (6.1):



Write

$$\Lambda_r(n, N, M) = \sum_{a=1}^{4} \Lambda_{r,a}(n, N, m)$$

where

$$\Lambda_{r,1}(n, N, m) = \sum_{k,j,t} \chi_{\{kr > m+1\}} \omega_r(k, 0) \gamma_r(n-k, N-kr+1, t-kr+1, j-k), \tag{A3.1a}$$

$$\Lambda_{r,2}(n, N, m) = \sum_{k,j} \chi_{\{kr > m+1\}} \omega_r(k, 0) \gamma_r(n-k, N-kr+1, m-kr+1, j-k), \tag{A3.1b}$$

$$\begin{aligned} \Lambda_{r,3}(n, N, m) &= \sum_{k,j,t} \chi_{\{(j+k)r > m+1\}} \psi_{r,R}(k, k+jr-1-t) \psi_{r,L}(j+1, t-j(r-1)) \\ &\quad \cdot \gamma_r(n-(k+j), N-(k+j)r+1, 0), \end{aligned} \tag{A3.1c}$$

$$\begin{aligned} \Lambda_{r,4}(n, N, m) &= \sum_{k,j} \chi_{\{(j+k)r > m+1\}} \psi_{r,R}(k, k+jr-1-m) \psi_{r,L}(j+1, m-j(r-1)) \\ &\quad \cdot \gamma_r(n-(k+j), N-(k+j)r+1, 0) \end{aligned} \tag{A3.1d}$$

and define

$$\Psi_{r,i}(\rho, m) = \lim_{n\to\infty} \frac{\Lambda_{r,i}(n, n\rho r, m)}{(1+n\rho r)\binom{n\rho r}{n(r-1)}}, \qquad 1 \leqq i \leqq 4.$$

Introduce the generating functions

$$L_{r,a}(w, z, y) = \sum_{n=1}^{\infty} w^n \sum_{N=n(r-1)}^{nr-1} z^N \sum_{m=0}^{\infty} y^m \Lambda_{r,a}(n, N, m), \qquad 1 \leqq a \leqq 4$$

so that

$$L_r(w, z, y) = \sum_{a=1}^{4} L_{r,a}(w, z, y).$$

If $F(\zeta_1, \cdots)$ is analytic in the vector variable $(\zeta_1, \cdots)$ in a neighborhood of the origin $(0, \cdots)$, then $\mathbf{C}_{n_1,n_2,\ldots} F(\zeta_1, \zeta_2, \cdots)$ denotes the coefficient of $\zeta_1^{n_1}\zeta_2^{n_2}\cdots$ in the power series expansion about the origin.

From explicit expressions for the generating functions $\{L_{r,a}(w, z, y)\}$ to be derived next, we want to identify the coefficient of $w^n z^N y^m$ for $n(r-1) \leqq N < nr$. These expressions will contain two types of terms; the first type is

$$\frac{1}{(1 - wz^{r-1}X^{r-1}(wz^r))^A}.$$

By the binomial theorem

$$\text{(A3.2)} \qquad \frac{1}{(1 - wz^{r-1}X^{r-1}(wz^r))^A} = \sum_{i=0}^{\infty} \binom{i+A-1}{i} (wz^{r-1}X^{r-1}(wz^r))^i$$

and then by (A1.2)

$$= \sum_{i=0}^{\infty} (wz^{r-1})^i \binom{i+A-1}{i} \cdot \sum_{j=0}^{\infty} \frac{i(r-1)}{jr+i(r-1)} \binom{jr+i(r-1)}{j} (wz^r)^j.$$

The coefficient of the term $w^n z^N$ is a sum of products

$$\binom{i+A-1}{i} \frac{i(r-1)}{jr+i(r-1)} \binom{jr+i(r-1)}{j}$$

where i and j satisfy

$$i+j=n, \qquad i(r-1)+jr=N$$

yielding

$$\text{(A3.3)} \qquad \mathbf{C}_{n,N} \frac{1}{(1 - wz^{r-1}X^{r-1}(wz^r))^A} = \binom{N}{N-n(r-1)} \binom{nr-N+A-1}{A-1},$$

$$n(r-1) \leqq N \leqq nr, \qquad 0 \leqq n < \infty.$$

The second type of term is

$$\frac{X^{s+1}(wz^r)}{r-(r-1)X(wz^r)}.$$

From (A1.4) we have

$$\text{(A3.4)}\quad \mathbf{C}_{n,N}\frac{X^{s+1}(wz^r)}{r-(r-1)X(wz^r)}=\begin{cases}\binom{nr+s}{n} & \text{if } N=nr,\quad 0\leqq n<\infty,\\ 0 & \text{otherwise.}\end{cases}$$

Evaluation of $L_{r,1}(w, z, y)$

Multiply (A3.1a) by $w^n z^N y^m$ and sum over the appropriate (n, N, m) to obtain

$$\text{(A3.5)}\quad \begin{aligned}L_{r,1}(w,z,y)=\sum_{n=1}^{\infty} w^n \sum_{N=n(r-1)}^{nr-1} z^N \sum_{m=0}^{\infty} y^m \sum_{k,j,t} \chi_{\{kr>m+1\}}\\ \cdot[\omega_r(k,0)\gamma_r(n-k,N-kr+1,t-kr+1,j-k)].\end{aligned}$$

Next, make the changes of variables

$$n'=n-k,\qquad N'=N-kr+1,\qquad j'=j-k,\qquad t'=t-kr+1$$

in (A3.5) to obtain

$$\text{(A3.6)}\quad \begin{aligned}L_{r,1}(w,z,y)&=\frac{1}{z}[G_r(w,z,1,1)-1]\sum_{m=0}^{\infty} y^m \sum_{k=1}^{\infty}\chi_{\{kr>m+1\}}\omega_r(k,0)(wz^r)^k\\ &=\frac{1}{z}[G_r(w,z,1,1)-1]\sum_{k=1}^{\infty}\omega_r(k,0)(wz^r)^k\sum_{m=0}^{kr-2} y^m\\ &=\frac{1}{z}[G_r(w,z,1,1)-1]\sum_{k=1}^{\infty}\omega_r(k,0)(wz^r)^k\frac{1-y^{kr-1}}{1-y}\\ &=\frac{1}{z}[G_r(w,z,1,1)-1]\frac{H_r(wz^r)-y^{-1}H_r(w(zy)^r)}{1-y}.\end{aligned}$$

To determine $\mathbf{C}_{n,N,m}L_{r,1}(w, z, y)$ we need to examine the two factors in (A3.6). We begin with $G_r(w, z, 1, 1)$. For this evaluation we set $v=1$ in (5.8) to obtain

$$\text{(A3.7)}\quad \begin{aligned}G_r(w,z,1,u)-1&=wz^{r-1}X(uwz^r)\frac{X^{r-1}(wz^r)-uX^{r-1}(uwz^r)}{1-u}\\ &\quad\cdot\frac{1}{1-wz^{r-1}X^{r-1}(wz^r)}\frac{1}{1-uwz^{r-1}X^{r-1}(uwz^r)},\\ &=wz^{r-1}X(uwz^r)\sum_{i=0}^{\infty}\frac{r-1}{ir+r-1}\binom{ir+r-1}{i}(wz^r)^i\frac{1-u^{i+1}}{1-u}\\ &\quad\cdot\frac{1}{1-wz^{r-1}X^{r-1}(wz^r)}\frac{1}{1-uwz^{r-1}X^{r-1}(uwz^r)}.\end{aligned}$$

Allowing $u\to 1$, we obtain

$$\text{(A3.8)}\quad \begin{aligned}G_r(w,z,1,1)-1&=z^{-1}\frac{wz^rX(wz^r)}{(1-wz^{r-1}X^{r-1}(wz^r))^2}\sum_{i=0}^{\infty}\binom{ir+r-2}{i}(wz^r)^i\\ &=z^{-1}\frac{wz^rX^r(wz^r)}{r-(r-1)X(wz^r)}\frac{1}{(1-wz^{r-1}X^{r-1}(wz^r))^2}\end{aligned}$$

the last equality from (A1.4). Next, using (4.9), (A1.2) and the defining relationship for $X(\zeta)$, we obtain

$$\frac{H_r(wz^r)-y^{1-}H_r(w(zy)^r)}{1-y}=\sum_{k=1}\frac{1}{ir-1}\binom{kr-1}{k}\frac{1-y^{kr-1}}{1-y}(wz^r)^k. \tag{A3.9}$$

Finally, combining (A3.2), (A3.8) and (A3.9) we have

$$\begin{aligned}
L_{r,1}(w,z,y)&=\frac{1}{z^2}(wz^r)\sum_{j=0}^{\infty}(j+1)(wz^{r-1})^j\frac{X^{j(r-1)+r}(wz^r)}{r-(r-1)X(wz^r)}\\
&\quad\cdot\left[\sum_{k=1}^{\infty}\frac{1}{kr-1}\binom{kr-1}{k}\frac{1-y^{kr-1}}{1-y}(wz^r)^k\right]\\
&=\frac{1}{z^2}\sum_{j=0}^{\infty}(j+1)(wz^{r-1})^j\sum_{i=0}^{\infty}\binom{ir+j(r-1)+r-1}{i}(wz^r)^{i+1}\\
&\quad\cdot\left[\sum_{k=1}^{\infty}\frac{1}{kr-1}\binom{kr-1}{k}\frac{1-y^{kr-1}}{1-y}(wz^r)^k\right]\\
&=\sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\sum_{k=1}^{\infty}w^{i+j+k+1}z^{ir+j(r-1)+kr+r-2}\frac{j+1}{kr-1}\\
&\quad\cdot\binom{ir+j(r-1)+r-1}{i}\binom{kr-1}{k}\frac{1-y^{kr-1}}{1-y}.
\end{aligned} \tag{A3.10}$$

The coefficient of $w^n z^N y^m$ is a sum of the terms

$$\frac{j+1}{kr-1}\binom{ir+j(r-1)+r-1}{i}\binom{kr-1}{k}$$

over index triples (i,j,k) which satisfy

$$i+j+k+1=n,\quad (i+k+1)r+j(r-1)-2=N,\quad m<kr$$

so that we obtain

$$\begin{aligned}
L_{r,1}(w,z,y)&=\sum_{n=0}^{\infty}w^n\sum_{N=n(r-1)}^{nr-1}z^N\sum_{k=1}^{N-n(r-1)+1}\frac{nr-N-1}{kr-1}\\
&\quad\cdot\binom{kr-1}{k}\binom{N-kr+1}{N-n(r-1)-k+1}\sum_{m=0}^{kr-2}y^m
\end{aligned} \tag{A3.11}$$

and

$$\Lambda_{r,1}(n,N,m)=\sum_{\substack{1\le k<\infty\\ \{kr\ge m+1\}}}\frac{nr-N-1}{kr-1}\binom{kr-1}{k}\binom{N-kr+1}{N-n(r-1)-k+1}. \tag{A3.12}$$

From (A3.12) we obtain

$$\begin{aligned}
\Psi_{r,1}(\rho,m)&=\frac{(1-\rho)r}{1-r+\rho r}\sum_{\substack{1\le k<\infty\\ \{kr\ge m+2\}}}\frac{1}{kr-1}\\
&\quad\cdot\binom{kr-1}{k}\left(\frac{(r-1)^{r-1}(1-r+\rho r)}{(\rho r)^r}\right)^k.
\end{aligned} \tag{A3.13}$$

To calculate the expected number of moves, we need to evaluate

$$\mathbf{C}_{n,N}\frac{\partial}{\partial y}\Lambda_r(w,z,y)|_{y=1}.$$

From (A3.6) we observe that we need only differentiate the term in (A3.9) to obtain

$$\begin{aligned}&\frac{\partial}{\partial y}\frac{H_r(wz^r)-y^{-1}H_r(w(zy)^r)}{1-y}\\ &\quad=\sum_{k=1}^{\infty}\frac{1}{ir-1}\binom{kr-1}{k}(1+2y+3y^2+\cdots+(kr-2)y^{kr-3})(wz^r)^k\end{aligned}\tag{A3.14}$$

and then make the obvious modifications in (A3.12) to obtain

$$\begin{aligned}\frac{\partial}{\partial y}L_{r,1}(w,z,y)|_{y=1}=\sum_{n=0}^{\infty}w^n\sum_{N=n(r-1)}^{nr-1}z^N\sum_{k=1}^{N-n(r-1)+1}&\\ \cdot\frac{1}{2}(nr-N-1)(kr-2)\binom{kr-1}{k}&\\ \cdot\binom{N-kr+1}{N-n(r-1)-k+1}.&\end{aligned}\tag{A3.15}$$

Finally

$$\begin{aligned}&\lim_{n\to\infty}\sum_m m\frac{\Lambda_{r,1}(n,n\rho r,m)}{(1+n\rho r)\binom{n\rho r}{n(r-1)}}\\ &\quad=\frac{1}{2}\frac{(1-\rho)r}{1-r+\rho r}\sum_{\substack{k:\,1\le k<\infty\\ \{kr\ge m+2\}}}(kr-2)\binom{kr-1}{k}\left(\frac{(r-1)^{r-1}(1-r+\rho r)}{(\rho r)^r}\right)^k.\end{aligned}\tag{A3.16}$$

Evaluation of $L_{r,2}(w,z,y)$

Since $\gamma_r(n-k,N-kr+1,m-kr+1,j-k)=0$ whenever $kr>m+1$, we have $L_{r,2}(w,z,y)=0$.

Evaluation of $L_{r,3}(w,z,y)$

Multiply (A3.1c) by $w^nz^Ny^m$ and sum over all m and $(n,N)\in\Delta_r$. This results in the following:

$$\begin{aligned}L_{r,3}(w,z,y)=\sum_m y^m\sum_{(n,N)\in\Delta_r}w^nz^N\sum_{k,j,t}\chi_{\{(j+k)r>m+1\}}&\\ \cdot\psi_{r,R}(k,k+jr-1-t)\psi_{r,L}(j+1,t-j(r-1))&\\ \cdot\gamma_r(n-(k+j),N-(k+j)r+1,0).&\end{aligned}\tag{A3.17}$$

To evaluate the right-hand side of (A3.17), make the change of variables

$$n'=n-k-j,\qquad N'=N-(j+k)r+1$$

to obtain

$$L_{r,3}(w,z,y)=\frac{1}{z}G_r(w,z)\sum_{m\geqq 0}y^m\sum_{k\geqq 1}(wz^r)^k\sum_{j\geqq 0}\chi_{\{(j+k)r>m+1\}}$$
$$\cdot\sum_{t=j(r-1)}^{jr}\psi_{r,R}(k,k+jr-t-1)\psi_{r,L}(j+1,t-j(r-1)).$$

Next, interchange the order of the m- and k-summations to get

$$L_{r,3}(w,z,y)=\frac{1}{z}G_r(w,z)\sum_{k\geqq 1}(wz^r)^k\sum_{j\geqq 0}\sum_{t=j(r-1)}^{jr}(wz^r)^j \tag{A3.18}$$
$$\cdot\psi_{r,R}(k,k+jr-t-1)\psi_{r,L}(j+1,t-j(r-1))\frac{1-y^{(j+k)r-1}}{1-y},$$

and make the substitutions

$$l=t-j(r-1),\qquad i=jr-t$$

to obtain

$$L_{r,3}(w,z,y)=\frac{G_r(w,z)}{z(1-y)}\sum_{k\geqq 1}(wz^r)^k\sum_{i\geqq 0}\psi_{r,R}(k,k+i-1)$$
$$\cdot\sum_{l\geqq 0}(wz^r)^{i+l}(1-y^{(i+l+k)(r-1)})\psi_{r,L}(i+l+1,l).$$

Introducing the explicit expression in (4.29) for $\psi_{r,L}(\cdots)$ allows us to sum over l and find

$$L_{r,3}(w,z,y)=\frac{G_r(w,z)}{z(1-y)}\sum_{k\geqq 1}(wz^r)^k\sum_{i\geqq 0}(wz^r)^i \tag{A3.19}$$
$$\cdot\psi_{r,R}(k,k+i-1)[X^{i(r-1)+1}(wz^r)-y^{(i+k)r-1}X^{i(r-1)+1}(wz^ry^r)].$$

Now using (4.28) for the value of $\psi_{r,R}(\cdots)$, we obtain

$$L_{r,3}(w,z,y)=\frac{G_r(w,z)}{z(1-y)}\sum_{k\geqq 1}(wz^r)^k\sum_{i\geqq 0}(wz^r)^i$$
$$\cdot\sum_{j=k-1}^{k+i-1}\frac{r-1}{jr+r-1}\binom{jr+r-1}{j}\binom{(k-1)(r-1)-jr+k+i-2}{k+i-j-1}$$
$$\cdot[X^{i(r-1)+1}(wz^r)-y^{(i+k)r-1}X^{i(r-1)+1}(wz^ry^r)].$$

Another change of variables

$$t=j-k+1,\qquad l=i+k-j-1$$

and summation over l, using the relation $wz^rX^r(wz^r)=X(wz^r)-1$ in the result and similarly for $X(wz^ry^r)$ simplifies to

$$L_{r,3}(w,z,y)=\frac{G_r(w,z)}{z(1-y)}\sum_{k\geqq 1}(wz^r)^k\sum_{t\geqq 0}\frac{r-1}{(t+k)r-1} \tag{A3.20}$$
$$\cdot\binom{(t+k)r-1}{t+k-1}[X(wz^r)-y^{(t+k)r-1}X(wz^ry^r)].$$

Let $i = k + t$, eliminate t and again change the order of summation to obtain

$$L_{r,3}(w, z, y) = \frac{G_r(w, z)}{z(1-y)} \sum_{i \geqq 0} \frac{(i+1)(r-1)}{ir+r-1} \binom{ir+r-1}{i} [X(wz^r) - y^{ir+r-1} X(wz^r y^r)]$$

$$= \frac{wz^r G_r(w, z)}{z(1-y)} \frac{\partial}{\partial a} \left[a \sum_{i \geqq 0} a^i \frac{r-1}{ir+r-1} \binom{ir+r-1}{i} \times [X(wz^r) - y^{ir+r-1} X(wz^r y^r)] \right]_{a = wz^r}$$

and using (A1.3) this readily yields

$$L_{r,3}(w, z, y) = \frac{wz^r G_r(w, z)}{z(1-y)} \left[\frac{X^r(wz^r)}{r + (1-r) X(wz^r)} - y^{r-1} \frac{X^r(wz^r y^r)}{r + (1-r) X(wz^r y^r)} \right]. \tag{A3.21}$$

Now use Lemma 5.2 for the coefficients in $G_r(w, z)$, and expand the brackets in terms of the familiar series to directly obtain the coefficient of $w^n z^N$:

$$L_{r,3}(n, N, y) = \binom{nr-1}{n-1} \frac{1-y^{nr-1}}{1-y} \chi_{\{N = nr-1\}} + \sum_{i \geqq 0} \frac{(nr-N-1)(r-1)}{N-ir-r+1} \binom{N-ir-r+1}{N-n(r-1)-i} \times \binom{ir+r-1}{i} \frac{1-y^{ir+r-1}}{1-y}. \tag{A3.22}$$

The contribution of this toward the average value is immediate:

$$\frac{d}{dy} L_{r,3}(n, N, y)|_{y=1} = \binom{nr-1}{n-1} \frac{N(nr-2)}{2} \chi_{\{N = nr-1\}} + \frac{(r-1)(nr-N-1)}{2} \sum_{i \geqq 0} \frac{(ir+r-1)(ir+r-2)}{N-ir-r+1} \cdot \binom{N-ir-r+1}{N-n(r-1)-i} \binom{ir+r-1}{i}. \tag{A3.23}$$

Upon dividing by the normalization factor

$$(N+1) \binom{N}{n(r-1)}$$

we find that as $n \to \infty$, this contribution to the expected value vanishes; this is intuitively obvious, since $L_{r,3}$ counts the contributions of such insertions where the displaced dummy key is the minimal one. There are infinitely many dummy keys in the limit as $n \to \infty$ for any $\rho < 1$.

Evaluation of $L_{r,4}(w, z, y)$

For the contribution of the fourth part of $\Pr\{M_r(n, N) = m\}$ no explicit form was found; we can write one, though, for its generation function.

Multiply (A3.1d) by $w^n z^N y^m$ and sum over all m and $(n, N) \in \Delta_r$; this results in

$$L_{r,4}(w, z, y) = \sum_m y^m \sum_{(n,N)\in\Delta_r} w^n z^N \sum_{k,j} \chi_{\{(j+k)r>m+1\}} \psi_{r,R}(k, k+jr-1-m) \times \psi_{r,L}(j+1, m-j(r-1)) \gamma_r(n-(k+j), N-(k+j)r+1, 0).$$

To evaluate the right-hand side make the change of variables

$$n' = n-k-j, \quad N' = N-(j+k)r+1, \quad m' = m-j(r-1)$$

to obtain

$$L_{r,4}(w, z, y) = \frac{1}{z} G_r(w, z) \sum_{k\geq 1} (wz^r)^k \sum_{j\geq 0} (wz^r y^{r-1})^j \cdot \sum_{m=0}^{j} y^m \chi_{\{j+kr>m+1\}} \psi_{r,R}(k, k+j-m-1) \psi_{r,L}(j+1, m). \tag{A3.24}$$

The conditions $k \geq 1$, $r \geq 2$ and $m \leq j$ on the ranges of summation in the last equation guarantee that $\chi_{\{j+kr>m+1\}} = 1$ so that

$$L_{r,4}(w, z, y) = \frac{1}{z} G_r(w, z) \sum_{k\geq 1} (wz^r)^k \sum_{j\geq 0} (wz^r y^{r-1})^j \cdot \sum_{m=0}^{j} y^m \psi_{r,R}(k, k+j-m-1) \psi_{r,L}(j+1, m)$$

$$= \frac{1}{z} G_r(w, z) X(wz^r y^r) \beta H_{r,R}(\alpha, \beta)$$

where

$$\alpha\beta = wz^r, \qquad \beta = wz^r y^{r-1} X^{r-1}(wz^r y^r).$$

This gives the result

$$L_{r,4}(w, z, y) = wz^{r-1} G_r(w, z) X(wz^r y^r) \frac{X^{r-1}(wz^r) - y^{r-1} X^{r-1}(wz^r y^r)}{1-[1+(y-1)X(wz^r y^r)]^{r-1}}. \tag{A3.25}$$

However, no expression for the coefficient of $w^n z^N y^m$ seems to be available from (A3.25) that is simpler than the original one that can be read off (A3.1d):

$$L_{r,4}(n, N, y) = \sum_{k=1}^{n-1} \sum_{j=0}^{n-k} \sum_{m=j(r-1)}^{jr} y^m \psi_{r,R}(k, k+jr-m-1) \cdot \psi_{r,L}(j+1, m-j(r-1)) \gamma_r(n-k-j, N-(k+j)r+1, 0). \tag{A3.26}$$

From the remark following (A3.23) we conclude that the contribution of the above term to $\Pr\{M_r(n, N) = m\}$ vanishes in the limit as $n \to \infty$.

To evaluate

$$\frac{\partial}{\partial y} L_{r,4}(w, z, y)\big|_{y=1}$$

expand the numerator and denominator of (A3.25) in a power series about the point $y = 1$:

$$X(wz^r y^r) = X(wz^r) + (y-1) rwz^r \frac{X^{r+1}(wz^r)}{r+(1-r)X(wz^r)} + O((1-y)^2),$$

$$X^{r-1}(wz^r)-y^{r-1}X^{r-1}(wz^ry^r)$$
$$=-(r-1)(y-1)$$
$$\cdot\left[\frac{X^r(wz^r)}{r+(1-r)X(wz^r)}+\frac{1}{2}(y-1)\sum_{i\geq 0}(ir+r-2)\binom{ir+r-1}{i}(wz^r)^i\right]+O((1-y)^3),$$
$$1-[1+(y-1)X(wz^ry^r)]^{r-1}$$
$$=-(r-1)(y-1)$$
$$\cdot\left[X(wz^r)+\frac{1}{2}(y-1)[(r-2)X^2(wz^r)+2rX'(wz^r)wz^r]\right]+O((1-y)^3)$$

and immediately

$$\frac{\partial}{\partial y}L_{r,4}(w,z,y)\big|_{y=1}$$
$$=wz^{r-1}\frac{G_r(w,z)}{(1-r)X(wz^r)}\left[X(wz^r)\frac{1-r}{2}\sum_{i\geq 0}(ir+r-2)\binom{ir+r-1}{i}(wz^r)^i\right.$$
$$+rwz^r\frac{(1-r)X^{2r+1}(wz^r)}{[r+(1-r)X(wz^r)]^2}$$
$$\left.-\frac{(1-r)^2X^{r+1}(wz^r)[(r-2)X^2(wz^r)+2rX'(wz^r)wz^r]}{2[r+(1-r)X(wz^r)]}\right]$$

which upon cancellation yields

$$=\frac{1}{2}\sum_{i\geq 0}(wz^r)^i\left[(ir+r-2)\binom{ir+r-1}{i}+(2-r)\binom{ir+r}{i}\right]. \tag{A3.27}$$

Proceeding as we did from (A3.21) we finally obtain

$$\frac{\partial}{\partial y}L_{r,4}(n,N,y)\big|_{y=1}$$
$$=\frac{1}{2}\left[(nr-2)\binom{nr-1}{n-1}+(2-r)\binom{nr}{n-1}\right]\chi_{\{N=nr-1\}}$$
$$+\frac{(nr-N-1)(r-1)}{2}\sum_{i\geq 0}\frac{1}{N-ir-r+1}\binom{N-ir-r+1}{(n-i-1)(r-1)}$$
$$\cdot(ir+r-2)\binom{ir+r-1}{i}+(2-r)\binom{ir+r}{i}. \tag{A3.28}$$

Collecting the contributions from (A3.7), (A3.23) and (A3.28), and dividing by the normalization factor of (6.3) we obtain Theorem 6.1. The value given in (A3.12) is the only component of $E\{M_r(n,N)\}$ that survives in the limit as $n\to\infty$, yielding then Theorem 6.2, and likewise for (A3.11) and Theorem 6.3.

REFERENCES

[1] LARS V. AHLFORS, *Complex Analysis*, McGraw-Hill, New York, 1953.

[2] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[3] W. R. FRANKLIN, *Padded lists: Set operations in expected $O(\log\log N)$ time*, Inform. Process. Lett., 9 (1979), pp. 161-166.

[4] ALON ITAI, ALAN G. KONHEIM AND MICKY RODEH, *A sparse table implementation of sorted lists*, IBM Research Report 8550, 1980.

[5] R. MELVILLE AND D. GRIES, *Sorting and searching using controlled density arrays*, Technical Report # 78-362, Cornell University, Ithaca, NY.

[6] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[7] V. VUILLEMIN, *A data structure for manipulating priority queues*, Comm. ACM, 21 (1978), pp. 309–315.

[8] E. T. WHITTAKER AND G. N. WATSON, *A Course of Modern Analysis*, Cambridge University Press, London, 1952.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
009

HEIGHT RESTRICTED OPTIMAL BINARY TREES*

LAWRENCE L. LARMORE†

Abstract. An algorithm that constructs an optimal height-restricted binary tree for a set of n weights in $O(n^{3/2}L\log^{1/2} n)$ time, where L is the maximum permitted height, is presented. This is an improvement over the fastest previously known algorithm, which requires $O(n^2L)$ time. The algorithm is a hybrid, combining a technique by Hu and Tan with a technique by Michael Garey.

Key words. Huffman coding, binary trees

AMS(MOS) subject classifications. 68P20, 68P25

1. Introduction. *Organization of this paper.* In this section, we describe the problem of finding an optimal height-restricted binary tree, and summarize past and present results. In § 2, we give some needed preliminary definitions. In § 3, we explain Hu and Tan's algorithm [3] and show how it can be sped up to $O(n^2L)$ time, while in § 4, we present Garey's $O(n^2L)$-time algorithm [1], using an improved version of his monotonicity lemma that will be needed later. In § 5, we present our main result, an $O(n^{3/2}L\log^{1/2}n)$-time "hybrid" algorithm that combines the methods of [3] and [1].

Given a list $(w_1, \cdots, w_n)$ of nonnegative weights, an *alphabetic binary tree* for that list is defined to be a full[1] binary tree with n leaves, called $A_1, \cdots, A_n$, where $\text{weight}(A_i) = w_i$. The *weighted path length* of such a tree is defined to be $P = \sum_{1 \leqq i \leqq n} w_i l_i$, where l_i is the depth of A_i, and we say that an alphabetic binary tree is *optimal* if it has minimum weighted path length. We define a *binary tree* for the list $(w_1, \cdots, w_n)$ to be an alphabetic binary tree for an arbitrary permutation of the list. The problem of finding an optimal binary tree (i.e., one with minimum weighted path length) for a given weight list is actually a special case of the problem of finding an optimal alphabetic binary tree. (Given an arbitrary list of weights, simply sort the weights. The optimal alphabetic tree for the sorted list is an optimal binary tree for the unsorted list. See [3], [1], also Lemma 2.3 below.)

Huffman [5] presents on $O(n \log n)$-time algorithm for constructing an optimal binary tree for a list of weights of length n.[2] Hu and Tucker [4] present an $O(n \log n)$-time algorithm for finding an optimal alphabetic binary tree for a given list of weights. In a later paper, Hu and Tan [3] present an algorithm for constructing an optimal height-restricted binary tree, that is, a binary tree for a given list of weights, with the restriction that its height cannot exceed a given integer L. (Note that such a tree can exist if and only if $L \geqq \log_2 n$.) The time complexity of their algorithm is exponential in L. Garey [1] presents a different algorithm for the same problem, which requires $O(n^2L)$ time. Garey's method is substantially different from that of Hu and Tan who

* Received by the editors August 29, 1986; accepted for publication (in revised form) February 10, 1987.

† California State University, Dominguez Hills, Carson, California 90747 and Department of Information and Computer Science, University of California, Irvine, California 92717.

[1] We define a binary tree to be *full* if every node is either a leaf or has exactly two children.

[2] It is not generally known that Huffman's algorithm runs in linear time if the items are already sorted. Place the items in a sorted stack S, smallest on top, and initialize a queue Q to empty. Each step examines the top two (or fewer, if there are fewer) items in S and the front two (or fewer, if there are fewer) items in Q. The smallest two of these four (or fewer) items are then popped or dequeued, and then combined to form a single node which is enqueued. By induction, Q will remain in sorted order, smallest to the front. The algorithm halts when S is empty and Q has one item, which will be the finished tree.

use left-to-right dynamic programming, finding optimal partial trees on prefixes of the weight list. Garey, on the other hand, uses a bottom-up approach, finding optimal subtrees on sublists of increasing size. His algorithm requires $O(n^3L)$ time if it is applied in a traditional way, but he proves a monotonicity lemma which cuts the time by an order of magnitude. The lemma proved by Garey [1, Cor. 2] is in the same spirit as Knuth's monotonicity lemma which reduces the time needed to construct an optimal binary search tree by an order of magnitude [7, p. 18].

Garey also addresses the problem of constructing an optimal height-restricted alphabetic binary tree. He points out that his algorithm can be applied in that case, but his proof of the monotonicity lemma requires monotonicity of the weights, and thus does not apply in the alphabetic case. His version of the algorithm in this case takes $O(n^3L)$ time. Itai [6] and Wessner [8] show that this time can be improved to $O(n^2L)$.

In this paper we show that Hu and Tan's algorithm can be sped up to $O(n^2L)$ time by pruning the set of partial solutions computed. The key to this reduction is the pruning lemma (Lemma 3.2) which greatly restricts the number of partial solutions that could possibly lead to an optimal solution.

Our hybrid algorithm for the nonalphabetic case combines the two methods. The algorithm uses the bottom-up approach to Garey to compute optimal height-restricted subtrees up to some length, then uses a left-to-right dynamic programming technique, reminiscent of Hu and Tan's, to finish the job. Another pruning lemma is used to prune the set of partial solutions. The time required is $O(n^{3/2}L\log^{1/2} n)$ for each of the two parts of the hybrid algorithm.

2. Tree lists. We define the *width* of a list $(l_1, \cdots, l_m)$ of nonnegative integers to be $\sum_{1\le i\le m} 2^{-l_i}$, and we say that the list is a *tree list* if it is the list of depths of the leaves of some full binary tree in left-to-right order. A tree list must have width 1, but that condition is insufficient. For example, (0), (1, 1), (1, 2, 2), and (2, 3, 3, 1) are tree lists, while (2, 1, 2) is not a tree list. We define a *subtree list of depth d* to be the list of depths of the leaves of some subtree, whose root is at depth d, of some full binary tree, and we define a *partial tree list* to be a prefix of a tree list. A subtree list is a partial tree list, a partial tree list is a subtree list of depth d if its width is 2^{-d}, and a subtree list of depth 0 is a tree list. For example, (5, 6, 6) is a subtree list of depth 4, (2, 3) is a partial tree list of width 3/8, and hence is not a subtree list, while (3, 2) is not a partial tree list.

The following two lemmas give recursive characterizations of subtree lists and partial tree lists.

LEMMA 2.1. *A list of nonnegative integers w is a subtree list of depth d if and only if one of the following holds*:

(i) $w = (d)$;

(ii) $w = uv$ (*the concatenation*), *where u and v are subtree lists of depth* $d+1$.

LEMMA 2.2. *A list of nonnegative integers* 2 *is a partial tree list if and only if one of the following holds*:

(i) $w = (\ )$, *the empty list*;

(ii) *w is a subtree list*;

(iii) *width* $(w) < 1$ *and* $w = uv$, *where v is a subtree list of depth d for some* $d > 1$, *and u is a partial tree list whose width is a positive even integral multiple of* 2^{-d}.

Lemma 2.1 follows immediately from the definitions, while Lemma 2.2 is slightly trickier. We leave the proof as an exercise for the reader. (Hint: If w is a prefix of the list of depths of the leaves of a full binary tree T, let S be the largest subtree of T

whose rightmost leaf is the leaf corresponding to the last item of w. Then the leaves of S correspond to v.)

The following lemma is needed to reduce the nonalphabetic optimal binary tree problem to the alphabetic binary tree problem. (See also [1], [3].)

LEMMA 2.3. *If $w=(l_1,\cdots,l_m)$ is a monotone increasing list of nonnegative integers whose width does not exceed* 1, *then w is a partial tree list.*

Proof. By induction on m. If $m=0$, the result is immediate. If width $(w)=2^{-d}$ for some d, we are done by Lemma 2.2. Otherwise, $m\geqq 2$, and we can choose d such that width $(w)=b2^{-d}$ for some odd integer b. Choose the maximum k such that width $(l_1,\cdots,l_k)=\sum_{1\leqq i\leqq k}2^{-l_i}\leqq(b-1)2^{-d}$. Since $l_i\leqq l_{k+1}$ for all $i\leqq k$, width $(l_1,\cdots,l_k)=a2^{-l_{k+1}}\leqq(b-1)2^{-d}$ for some positive integer a, and width $(l_1,\cdots,l_{k+1})=(a+1)^{-l_{k+1}}>(b-1)2^{-d}$. It follows that width $(l_1,\cdots,l_k)=(b-1)2^{-d}$. By the inductive step, $u=(l_1,\cdots,l_k)$ is a partial tree list, and $v=(l_{k+1},\cdots,l_m)$ is a subtree list of depth d. By Lemma 2.2 we are done.

Rephrasing the problems. We can now express the optimal alphabetic binary tree problem as follows. Given a list $(w_1,\cdots,w_n)$ of nonnegative weights, find a tree list $(l_1,\cdots,l_n)$ such that $P=\sum w_il_i$ is minimized. In the height restricted case, simply minimize P subject to the requirement that $l_i\leqq L$ for all i. As pointed out by Hu and Tan, and also by Garey, the optimal binary tree problem is simply the special case of the optimal alphabetic binary tree problem in the case that the weights are monotone decreasing. For if T is an optimal binary tree for a given list of weights, Lemma 2.3 enables us to assume that the depths of the leaves of T are increasing from left to right. Suppose $w_i>w_j$ for $i<j$. Then the values of w_i and w_j can be exchanged without increasing P, since $l_i\leqq l_j$. Thus, we can assume, without loss of generality, that the weights are monotone decreasing.

3. Hu and Tan's algorithm. In this section we present Hu and Tan's original algorithm for constructing an optimal binary tree for a list of wieghts $(w_1,\cdots,w_n)$, and show how that algorithm can be sped up to require $O(n^2L)$ time.

We may assume that $w_i\geqq w_j$ for $i<j$. For any diadic rational number[3] q, and for any $0\leqq k\leqq n$, define

$$\psi_k(q)=\min\left\{\sum_{1\leqq i\leqq k}w_il_i\,\middle|\,(l_1,\cdots,l_k)\text{ is a partial tree list of width }q\right\}.$$

If there is no partial tree list of length k and width q, we define $\psi_k(q)$ to be ∞. Clearly $\psi_n(1)$ is the weighted path length of the optimal binary tree. Hu and Tan's method is to dynamically compute, for successively larger k, $\psi_k(q)$ for $q=a2^{-L}$ for all integral choices of a from 0 to 2^L.

From [3], we have the following.

LEMMA 3.1. *For any integer $0\leqq k\leqq n$ and any diadic rational $0\leqq q\leqq 1$ which is a multiple of 2^{-L},*

$$\psi_k(q)=\min\left\{\sum_{1\leqq i\leqq k}w_il_i\,\middle|\,(l_1,\cdots,l_k)\text{ is a list of nonnegative integers of width }q\right\}.$$

Proof. By definition, $\psi_k(q)$ is no less than that minimum. Suppose $(l_1,\cdots,l_k)$ is a list of nonnegative integers of width q. Let $(l_1',\cdots,l_k')$ be the list consisting of the

[3] We define a rational number to be *diadic* if it can be expressed as a fraction whose denominator is a power of 2.

same integers in ascending order. By Lemma 2.3, $(l'_1, \cdots, l'_k)$ is a partial tree list. Since the w_i are monotone decreasing, $\sum_{1\leq i\leq k} w_i l'_i \leq \sum_{1\leq i\leq k} w_i l_i$. By definition, $\psi_k(q) \leq \sum_{1\leq i\leq k} w_i l'_i$, and we are done.

The algorithm of [3] is based on the following recursion relationship which depends on Lemma 3.1:

$$\psi_0(0) = 0,$$

$$\psi_0(q) = \infty \quad \text{for all } q > 0,$$

$$\text{For } k > 0, \quad \psi_k(q) = \min\{\psi_{k-1}(q - 2^{-d}) + w_k d \mid 0 \leq d \leq L, 2^{-d} \leq q\}.$$

The Hu-Tan graph. We define a weighted directed acyclic graph $G = (V, E)$, called the *Hu-Tan graph*, as follows. V is the set of ordered pairs (k, q) such that k is an integer from 0 to n, and q is an integral multiple of 2^{-L} in the range 0 to 1. For each (k, q) where $k > 0$ and for each integer d in the range $\lceil \log_2(1/q) \rceil$ to L, there is an edge $\varepsilon_k[q, d] \in E$ from $(k-1, q-2^{-d})$ to (k, q). The weight of that edge is $w_k d$. Then Hu and Tan's algorithm is simply the classical dynamic programming algorithm for computing the minimum weighted path in a weighted directed acyclic graph. We call $(0, 0)$ the *source* of G. There is a one-to-one correspondence between source-based paths in G and nonnegative integer lists of width no greater than 1, where a list $(l_1, \cdots, l_k)$ corresponds to the path $\{\varepsilon_j[q_j, l_j]\}_{1\leq j\leq k}$, where $q_j = \text{width}(w_1, \cdots, w_j)$. The minimum weighted path length of a binary tree for the given weight list is the minimum weight of a directed path in G from $(0, 0)$ to $(n, 1)$. Furthermore, the optimal binary tree can be recovered easily from the minimum weight path.

The size of G is exponential in L. Hu and Tan's algorithm can be sped up considerably by eliminating from consideration nodes and edges which cannot be part of a minimum weight path. We define an edge $\varepsilon_k[q, d] \in E$ to be *useful* if $1 - q = a2^{-d}$ for some integer a such that $0 \leq a \leq n - k$. We define a vertex of G to be *useful* if it is at an end of a useful edge. Since there are at most $L+1$ choices of d, $n+1$ choices of k, $n+1$ choices of k, and for each pair of choices of k and d, at most $n+1$ choices of q, there are $O(n^2 L)$ useful edges and vertices in G. The speed-up of Hu and Tan's algorithm is to consider only useful edges in the procedure; since the traditional dynamic programming algorithm for finding the minimum weight path in a directed acyclic graph looks at each edge exactly once, the time required for the algorithm is $O(n^2 L)$. Correctness of the method is guaranteed by the pruning lemma, below.

LEMMA 3.2 (Pruning Lemma). *If $L \leq n$, there is a path in G which uses only useful edges whose weight is equal to the weighted path length of the optimal binary tree.*

Proof. Suppose that $(l_1, \cdots, l_n)$ is the tree list for an optimal binary tree. Without loss of generality, that list is monotone increasing. For each k, let $q_k = \text{width}(l_1, \cdots, l_k)$. Then the edges of the Hu-Tan graph in the path corresponding to that optimal binary tree are $\{\varepsilon_k[q_k, l_k] \mid 1 \leq k \leq n\}$. We need to show that all those edges are useful. Fix k. Since $l_i \leq l_k$ for all $i \leq k$, width $(w_1, \cdots, w_k)$ must be an integral multiple of 2^{-l_k}; hence width $(l_{k+1}, \cdots, w_n)$ must also be an integral multiple of 2^{-l_k}. Since $l_i \geq l_k$ for all $i > k$, width $(l_{k+1}, \cdots, l_n)$ cannot exceed $(n-k)2^{-l_k}$. By definition $\varepsilon_k[q_k, l_k]$ is useful.

4. Optimizing subtrees. In this section, we present Garey's $O(n^2 L)$-time algorithm for constructing an optimal height-restricted binary tree. In [1], the time given for constructing an optimal alphabetic tree is $O(n^3 L)$. The monotonicity lemma in [1] can be generalized to cover the alphabetic case, achieving $O(n^2 L)$-time in that case [6], [8].

Throughout this section, let $(w_1, \cdots, w_n)$ be a list of nonnegative weights. (We do not assume that the weights are sorted.) For all integers $1 \leq i, j \leq n$ and $0 \leq d \leq L$,

we define $\chi_{ij}(d)$ as follows:

$$\chi_{ij}(d) = \min\left\{ \sum_{i \leqq k \leqq j} w_k l_k \,\middle|\, (l_i, \cdots, l_j) \text{ is a subtree list of depth } d \text{ and } l_k \leqq L \text{ for all } k \right\} \quad \text{if } j-i+1 \leqq 2^{L-d},$$

$$\chi_{ij}(d) = \infty \quad \text{if } j-i+1 > 2^{L-d}.$$

If $i<j$, then by Lemma 2.1 we have that $\chi_{ij}(d) \leqq \chi_{i,r-1}(d+1)+\chi_{rj}(d+1)$ for all $i<r \leqq j$, and there is some value of r, which we call $\rho_{i,j}(d)$, for which the inequality becomes an equality. To eliminate ambiguity and simplify proofs, we require $\rho_{i,j}(d)$ be assigned the minimum such choice of r, although the ambiguity is actually harmless.

Note that $\chi_{ii}(d) = w_i d$. Garey's method is to compute the values of χ_{ij} dynamically for all choices of i, j, and d. Then $\chi_{1n}(0)$ will be the weighted path length of the optimal binary tree, and the actual tree can be recovered from the values of ρ.

THE BASIC SUBTREE ALGORITHM.
For all i and all d do
    $\chi_{ii}(d) \leftarrow w_i d$
For d from L down to 0 do
    For all pairs $1 \leqq i < j \leqq n$ do
        Compute $\chi_{ij}(d)$ and $\rho_{ij}(d)$

The innermost step of the basic algorithm is to assign to $\rho_{ij}(d)$ that value of r in the range $i<r \leqq j$ which minimizes the sum $\chi_{i,r-1}(d+1)+\chi_{rj}(d+1)$. The time required for each execution of that innermost step is $O(n)$, since every r in the range is a candidate. Thus, the time required for the basic algorithm is $O(n^3 L)$.

We now show how to decrease the time of the basic algorithm by an order of magnitude, i.e., by reducing the amortized time for each execution of the innermost step to $O(1)$.

The following lemma generalizes Theorem 1 of [1]. (See Yao [9] for a general discussion of "quadrangle conditions.")

LEMMA 4.1 (Quadrangle Lemma). *Suppose* $1 \leqq i_0 \leqq i_1 \leqq j_0 \leqq j_1 \leqq n$. *Then*

$$\chi_{i_0 j_0}(d)+\chi_{i_1 j_1}(d) \leqq \chi_{i_0 j_1}(d)+\chi_{i_1 j_0}(d).$$

Proof. By induction on $j_1 - i_0$. For $i_0 = j_1$, the result is trivial. Suppose, then, that $i_0 < j_1$. We consider two cases: $i_1 < j_0$ and $i_1 = j_0$.

If $i_1 < j_0$, let $r = \rho_{i_1 j_0}(d)$, and let $s = \rho_{i_0 j_1}(d)$. Without loss of generality, $r \leqq s$. We have

$$\chi_{i_0 j_0}(d) \leqq \chi_{i_0,r-1}(d+1)+\chi_{rj_0}(d+1),$$
$$\chi_{i_1 j_1}(d) \leqq \chi_{i_1,s-1}(d+1)+\chi_{sj_1}(d+1),$$
$$\chi_{i_0 j_1}(d) = \chi_{i_0,s-1}(d+1)+\chi_{sj_1}(d+1),$$
$$\chi_{i_1 j_0}(d) = \chi_{i_1,r-1}(d+1)+\chi_{rj_0}(d+1),$$

and by the inductive hypothesis:

$$\chi_{i_0,r-1}(d+1)+\chi_{i_1,s-1}(d+1) \leqq \chi_{i_0,s-1}(d+1)+\chi_{i_1,r-1}(d+1).$$

The rest of the proof of this case consists of straightforward manipulation of inequalities.

On the other hand, suppose $i_1 = j_0$. Note that $r = i_1 = j_0$. Let $s = \rho_{i_0 j_1}(d)$. Without loss of generality, $r < s$. We have

$$\chi_{i_0 j_1}(d) = \chi_{i_0,s-1}(d+1)+\chi_{sj_1}(d+1), \qquad \chi_{i_1 j_1}(d) \leqq \chi_{i_1,s-1}(d+1)+\chi_{sj_1}(d+1),$$

and by the inductive hypothesis:

$$\chi_{i_0 r}(d+1)+\chi_{r,s-1}(d+1)\leqq\chi_{i_0,s-1}(d+1)+\chi_{rr}(d+1).$$

Recall that $\chi_{rr}(d+1)=w_r(d+1)$, and $\chi_{rr}(d)=w_r d$. If $(l_{i_0},\cdots,l_r)$ is a subtree list corresponding to the optimal subtree of depth $(d+1)$ for the weights $(w_{i_0},\cdots,w_r)$, then $(l_{i_0}-1,\cdots,l_r-1)$ is a partial tree list of depth d, hence

$$\chi_{i_0 r}(d)\leqq\sum_{i_0\leqq k\leqq r}w_k(l_k-1)=\chi_{i_0 r}(d+1)-\sum_{i_0\leqq k\leqq r}w_k\leqq\chi_{i_0 r}(d+1)-w_r.$$

Thus

$$\chi_{i_0 r}(d)+\chi_{rr}(d+1)\leqq\chi_{i_0 r}(d+1)+\chi_{rr}(d).$$

The result follows from straightforward manipulation of these inequalities.

We can now derive a generalization of Garey's monotonicity lemma (Corollary 2 of [1]). This generalization is also present in [6] and [8].

LEMMA 4.2 (Monotonicity Lemma). *Let* $d\leqq L$. *Suppose* $1\leqq i_0\leqq i_1\leqq j_0\leqq j_1\leqq n$, $i_0<j_0$, *and* $i_1<j_1$. *Then* $\rho_{i_0 j_0}(d)\leqq\rho_{i_1 j_1}(d)$.

Proof. Let $r=\rho_{i_0 j_0}(d)$ and $s=\rho_{i_1 j_1}(d)$ and suppose that $r>s$. By definition of the ρ, we have

$$\chi_{i_0 j_0}(d)=\chi_{i_0,r-1}(d+1)+\chi_{r j_0}(d+1),$$

$$\chi_{i_0 j_0}(d)<\chi_{i_0,s-1}(d+1)+\chi_{s j_0}(d+1),$$

$$\chi_{i_1 j_1}(d)=\chi_{i_1,s-1}(d+1)+\chi_{s j_1}(d+1),$$

$$\chi_{i_1 j_1}(d)\leqq\chi_{i_1,r-1}(d+1)+\chi_{r j_1}(d+1),$$

while by Lemma 4.1, we have:

$$\chi_{i_0,s-1}(d+1)+\chi_{i_1,r-1}(d+1)\leqq\chi_{i_0,r-1}(d+1)+\chi_{i_1,s-1}(d+1),$$

$$\chi_{s j_0}(d+1)+\chi_{r j_1}(d+1)\leqq\chi_{r j_0}(d+1)+\chi_{s j_1}(d+1).$$

A contradiction is obtained by straightforward manipulation of the inequalities.

The speed-up of the basic algorithm is then as described in [1]. The key special case of Lemma 4.2 needed is that, for any $i<j$ such that $j-i\geqq 2$, and for any d, $\rho_{i,j-1}(d)\leqq\rho_{ij}(d)\leqq\rho_{i+1,j}(d)$. For each d, the values of $\rho_{ij}(d)$ are computed in order of increasing $gap=j-i$. For a particular value of d and a particular value of *gap*, there are at most $2n-gap-1$ candidates that need to be examined to compute all values of $\rho_{ij}(d)$. This method is essentially the same as used by Knuth in [7].

5. The Hybrid Algorithm. In this section, we introduce a new algorithm which combines features of Garey's and Hu and Tan's, to find an optimal height-restricted binary tree in the nonalphabetic case. The algorithm has one parameter g, and is in two parts. The first part uses the methods of § 4 (i.e., [1]), and computes minimum weighted path lengths of subtrees up to size g in $O(ngL)$-time. The second part uses the methods of § 3 (i.e., [3]) to finish the problem in $O((n^2L/g)\log n)$-time. Choosing g to be $\Theta(n^{1/2}\log^{1/2} n)$ causes the overall time complexity to be minimized.

We will assume throughout this section that a maximum height L is given, as are weights $w_1\cdots w_n$, such that $w_i\geqq w_j$ for $i<j$ for $i<j$. We also assume that $L\geqq\log_2 n$. The notation of §§ 2, 3 and 4 will be freely used.

The extended graph. We define a weighted directed acyclic graph $G^+ = (V^+, E^+)$ as follows. $V^+ = V$, the set of ordered pairs (k, q) such that k is an integer in the range 0 to n, and $q = a^2 - L$ where a is an interger in the range 0 to 2^L. E^+ consists of edges $\zeta_{ij}[q, d]$ from $(i, q-2^{-d})$ to (j, q), of weight $\chi_{i,j}(d)$. Note that G, the Hu-Tan graph, is a subgraph of G^+, which has the same vertices but only some of the edges. In particular $\varepsilon_k[q, d) = \zeta_{k-1,k}[q, d]$. We call G^+ the *extended* Hu-Tan graph. It follows immediately from the definition of $\chi_{ij}(d)$ given in § 4 that:

LEMMA 5.1. *The minimum weight of a path in G^+ between any two vertices is equal to the minimum weight of a path in G between the same two vertices.*

The useful graph $\bar{G}$. If $q \geqq 0$ is a diadic rational and $d \leqq L$, we say that (q, d) is a *useful pair* if $q = 1 - a2^{-d}$ for some integer $0 \leqq a \leqq 2n/g$, and we say that q is a *useful width* if (q, d) is a useful pair for some d. We say that (k, q) is a *useful* vertex if q is a useful width, and that $\zeta_{ij}[q, d]$ is a *useful edge* if $0 \leqq j - i + 1 \leqq g$ and (q, d) is a useful pair. Note that there are $\Theta(nL/g)$ useful pairs, $\Theta(nL/g)$ useful widths, $(\Theta(n^2L/g)$ useful vertices, and $\Theta(n^2L)$ useful edges. We define $\bar{G} = (\bar{V}, \bar{E})$ to be the subgraph of G^+ consisting of all useful vertices and edges. If $(k, q) \in \bar{V}$, define $\bar{\psi}_k(q)$ to be the minimum weight of any source-based path in $\bar{G}$ to (k, q). Clearly $\bar{\psi}_k(q) \geqq \psi_k(q)$.

LEMMA 5.2. $\bar{\psi}_n(1) = \psi_n(1)$.

Proof. Let $(l_1, \cdots, l_n)$ be a monotone decreasing tree list corresponding to an optimal tree, T. Let $q_k = \text{width}\,(l_1, \cdots, l_k)$ for all k; note that $\sum_{1 \leqq i \leqq k} w_i l_i = \psi_k(q_k)$. Let m be the largest index such that (m, q_m) is a useful pair and $\bar{\psi}_m(q_m) = \psi_m(q_m)$. If $m = n$ we are done. We will show by contradiction that m cannot be less than n.

Suppose $m < n$. Let α be a path in $\bar{G}$ from $(0, 0)$ to (m, q_m) of weight $\psi_m(q_m)$. Since q_m is a useful width, $q_m = 1 - a2^{-d}$ for integers $d \leqq L$ and $0 < a \leqq 2n/g$. Without loss of generality, a is odd, thus $d \leqq l_m \leqq l_{m+1}$. If $l_{m+1} = d$, $q_{m+1} = 1 - (a-1)2^{-d}$, which implies that $\varepsilon_{m+1}[q_{m+1}, d]$ is a useful edge. Let $\alpha' = \alpha\varepsilon_{m+1}[q_{m+1}, d]$. Then weight $(\alpha') =$ weight $(\alpha) + w_{m+1}d = \psi_m(q_m) + w_{m+1}l_{m+1} = \psi_{m+1}(q_{m+1})$, and we are done.

On the other hand, suppose $l_{m+1} > d$. Let S be the subtree of T whose root is at depth d which contains A_{m+1}. Since width $(l_1, \cdots, l_m)$ is an integral multiple of 2^{-d}, A_{m+1} is the leftmost leaf of S. We consider two cases: S has at most g leaves, and S has more than g leaves. If S has at most g leaves, let A_r be the rightmost leaf of S. Then $\zeta_{m+1,r}[q_r, d]$ is a useful edge, and $\alpha' = \alpha\zeta_{m+1,r}[q_r, d]$ is a path in $\bar{G}$ from $(0, 0)$ to (r, q_r), and weight $(\alpha') = \psi_r(q_r)$, which contradicts the maximiality of m. If S has more than g leaves, let S' be the smallest subtree of S which contains A_{m+1} and which has more than g leaves, and let S'' be the left subtree of S'. Let A_r be the rightmost leaf of S'', A_s the rightmost leaf of S', and let e be the depth of the root of S''. We have $r - m \leqq g$ and $s - m > g$. We will show that $\zeta_{m+1,r}[q_r, e]$ is a useful edge, by verifying two conditions. The first condition is that $r - m \leqq g$, which we already known. The second is that (q_r, e) is a useful pair. For some integer b, $q_r = 1 - b2^{-e}$; thus $q_m = 1 - (b+1)2^{-e}$. Since the l_i are monotone increasing:

$$1 - q_m = \text{width}\,(l_{m+1}, \cdots, l_n) \leqq ((n-m)/(s-m))\ \text{width}\,(l_{m+1}, \cdots, l_s) \leqq (n/g)2^{-e+1},$$

i.e., $b + 1 \leqq 2n/g$, so (q_r, e) is useful. Finally, $\alpha' = \alpha\zeta_{m+1,r}[q_r, e]$ is a path in $\bar{G}$ from $(0, 0)$ to (r, q_r) such that weight $(\alpha') = (\psi_r(q_r)$, contradicting the maximality of m.

This concludes the proof of Lemma 5.2.

The second monotonicity lemma. For any $(k, q) \in \bar{V}$, $\bar{\psi}_k(q) \leqq \bar{\psi}_{s-1}(q - 2^{-d}) + \chi_{sk}(d)$ for all choices of $d \leqq L$ and all choices of s such that $\max\{0, k-g+1, k-2^{L-d}+1\} \leqq s < k$. If $\bar{\psi}_k(q) < \infty$, then there exists some $d \leqq L$ and some $\max\{0, k-g+1, k-2^{L-d}+1\} \leqq s < k$ such that the inequality becomes an equality. Let $\delta_k[q]$ be that choice of d. In case of a tie, choose d arbitrarily among the candidates. For any useful pair (q, d)

and any $0 \leqq k \leqq n$, define $\sigma_k[q, d]$ to be that choice of s in the range $\max\{0, k-g+1, k-2^{L-d}+1\} \leqq s < k$ such that $\bar{\psi}_{s-1}(q-2^{-d})+\chi_{sk}(d)$ is minimized. If no such choice exists for which $\bar{\psi}_{s-1}(q-2^{-d})$ is finite, let $\sigma_k[q, d] = \max\{0, k-g+1, k-2^{L-d}+1\}$. In case of tie, choose $\sigma_k[q, d]$ to be as small as possible.

LEMMA 5.3 (Second Monotonicity Lemma). *If q is a useful width and if $0 \leqq d \leqq L$ and $0 \leqq k < k' \leqq n$ are integers, then $\sigma_k[q, d] \leqq \sigma_{k'}[q, d]$.*

Proof. Let $r = \sigma_k[q, d]$ and $s = \sigma_{k'}[q, d]$. Assume $r > s$. Since $s \geqq \max\{0, k'-g+1, k'-2^{L-d}+1\}$, $r > \max\{0, k-g+1, k-2^{L-d}+1\}$, which implies that $\bar{\psi}_{r-1}(q-2^{-d})$ is finite. By definition of the σ,

$$\bar{\psi}_{r-1}(q-2^{-d})+\chi_{rk}(d) < \bar{\psi}_{s-1}(q-2^{-d})+\chi_{sk}(d),$$

$$\bar{\psi}_{s-1}(q-2^{-d})+\chi_{sk'}(d) \leqq \bar{\psi}_{r-1}(q-2^{-d})+\chi_{rk'}(d),$$

while by the Quadrangle Lemma (Lemma 4.1):

$$\chi_{sk}(d)+\chi_{rk'}(d) \leqq \chi_{sk'}(d)+\chi_{rk}(d).$$

A contradiction follows immediately. (Hint: we have established the finitude of all terms.)

THE HYBRID ALGORITHM.

$g \leftarrow \lceil n^{1/2} \log n \rceil$
Compute $\chi_{i,j}(d)$ for all $1 \leqq i < j \leqq n$, $j-1 < g$, $d \leqq L$
$\bar{\psi}_0(0) \leftarrow 0$
$\bar{\psi}_k(0) \leftarrow \infty$ for all $0 < k \leqq n$
$\bar{\psi}(q) \leftarrow \infty$ for all useful widths q
$\sigma_0[q, d] \leftarrow 0$ for all useful pairs (q, d)
For all useful widths $q > 0$ in increasing order to
    begin
        For all d such that (q, d) is a useful pair do
            For $p \leftarrow \lfloor \log_2 n \rfloor$ down to 0 do
                For all $0 < k \leqq n$ such that k is an odd multiple of 2^p do
                    Compute $\sigma_k[q, d]$
        Compute $\bar{\psi}_k(q)$, $\delta_k(q)$
    end

Analysis of the Hybrid Algorithm. The first portion of the hybrid algorithm, which computes the χ, consists of simply running the subtree algorithm of [1] (described in § 4) part way, which requires $O(1)$ amortized time for each value computed. The time requirement for this portion is thus $O(ngL)$. We will focus attention on the remaining portion of the algorithm.

The innermost step, which computes $\sigma_k[q, d]$, appears to require $O(g)$ time in each instance. Making use of the Second Monotonicity Lemma (Lemma 5.3), we will show how to reduce the amortized time requirement for each such instance to $O(\log n)$.

Fix a useful pair (q, d) and an integer $0 \leqq p \leqq \lfloor \log_2 n \rfloor$. Let $m = \lceil n 2^{-p-1} \rceil$. Let $s_0 = 1$, $s_m = n$, and for all $0 < i < m$, let $s_i = \sigma_{2^{p+1}i}[q, d]$. At the time the iteration indexed by p is executed, the values of all s_i have been computed, and by Lemma 5.3, $s_0 \leqq s_1 \leqq \cdots \leqq s_m$. Again by Lemma 5.3, $s_{i-1} \leqq \sigma_k[q, d] \leqq s_i$ for $k = (2i-1)2^p$, and so the number of candidates that must be considered to compute $\sigma_k[q, d]$ for all k which are odd multiples of 2^p does not exceed $\sum_{1 \leqq i \leqq m}(s_i - s_{i-1}+1) = n+m-1 < 2n$. It therefore takes $O(n)$ time to execute the iteration of the innermost loop which is indexed by p, and thus $O(n^2 \log nL/g)$ time to compute all values of the σ.

For each q and each k, $\delta_k[q]$ is simply that member of $\{\sigma_k[q, d] | (q, d) \text{ useful}\}$ which minimizes $\bar{\psi}_k(q)$. Selecting the correct choice for all values of k and q requires examining each $\sigma_k[q, d]$ just once, for a total time of $O(n^2L/g)$.

The total time for the Hybrid Algorithm is thus $O(ngL+n^2(\log n)L/g) = O(n^{3/2}L\log^{1/2} n)$.

6. Open problems. It seems unlikely that the results of this paper will be the last word for either the alphabetic or nonalphabetic case. In particular, we conjecture that the time complexity for the nonalphabetic case can be improved to $O(n^{3/2}L^{1/2}\log^{1/2} n)$.

The alphabetic case seems considerably more difficult. An obvious question is, can the methods used for the Hybrid Algorithm be applied? If no reduction in time complexity can be achieved in the worst case, it should be possible to obtain an improvement for a wide set of cases, for example, with the restriction that $\min\{w_i\}_{1\leqq i\leqq n} = \Theta(1/n)$. (See [2].)

REFERENCES

[1] M. R. GAREY *Optimal binary search trees with restricted maximal depth*, this Journal, 3 (1974), pp. 101-110.

[2] J. H. HESTER, D. S. HIRSCHBERG AND L. L. LARMORE, *Construction of optimal binary split trees in the presence of bounded access probabilities*, to appear.

[3] T. C. HU AND K. C. TAN, *Path length of binary search trees*, SIAM J. Appl. Math., 22 (1972), pp. 225-234.

[4] T. C. HU AND A. C. TUCKER, *Optimal computer search trees and variable length alphabetic codes*, SIAM J. Appl. Math., 21 (1971), pp. 514-532.

[5] D. A. HUFFMAN, *A method for the construction of minimum redundancy codes*, Proc. Inst. Radio Engineers, 40 (1952), pp. 1098-1101.

[6] A. ITAI, *Optimal alphabetic trees*, this Journal, 5 (1976), pp. 9-18.

[7] D. E. KNUTH, *Optimum binary search trees*, Acta Informatica, 1 (1971), pp. 14-25.

[8] R. L. WESSNER, *Optimal alphabetic search trees with restricted maximal height*, Inform. Process. Lett., 4 (1976), pp. 90-94.

[9] F. F. YAO, *Efficient dynamic programming using quadrangle inequalities*, Proc. 12th Annual ACM Symp. on Theory of Comput., 1980, pp. 429-435.

[10] Y. ZHU AND J. WANG, *On alphabetic-extended binary trees with restricted path length*, Scientia Sinica, 22 (1979), pp. 1362-1371.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
010

ALMOST SURE FAULT TOLERANCE IN RANDOM GRAPHS*

EDWARD R. SCHEINERMAN†

Abstract. We analyze the PMC [10] model for fault tolerant systems by means of random directed graphs. Previous work has shown that the minimum in-degree of the testing digraph had to exceed the expected number of faulty units (vertices). We show, for much sparser digraphs than those described above, that the asymptotic probability of correct diagnosis of a faulty system tends to 1. Specifically we show that if our directed graph has n vertices, arc probability $p=(c \log n)/n$ and vertex failure probability $q<\frac{1}{2}$ then a simple algorithm will correctly diagnose the system "almost always" when $c>1/(1-q)$. However, when $c<1/(1-q)$ no algorithm can correctly diagnose the system. We present a similar analysis for a family of directed graphs which are not random. We conclude with analogous results for undirected graphs.

Key words. connection assignment, diagnosis, fault tolerance, permanent fault, PMC model, random graph, self-diagnosable systems, syndrome

AMS(MOS) subject classifications. 68M15, 05C80

1. Introduction.

We consider directed graphs whose vertices represent devices, such as computers, which are capable of testing each other. Arcs in these digraphs represent links over which the devices test one another. Each device can be in one of two states: faulty or nonfaulty. The test results of a nonfaulty unit are reliable, but the test results of a faulty device are not. We want to determine the set of faulty processors based on the results of the tests.

Previous work in this area has concentrated on finding the maximum number of faulty units which can be determined unambiguously by the test results. (See [6], [10].) The results say (essentially) that this maximum equals the minimum in-degree of the digraph. Our aim is to show that even if the number of faulty units is greater than the minimum in-degree, the probability of being able to diagnose the system can still be quite high. To this end, we employ the methods of random digraphs.

In the next section we introduce our probability model. In § 3 we introduce the notion of almost sure diagnosability and find models in which almost all random digraphs can be (or cannot be) diagnosed. In § 4 we present a probabilistic analysis of a typical fault tolerant scheme. In § 5 we show that our results are asymptotically best possible in both the random and deterministic cases. Finally, in § 6 we consider diagnosis of undirected graphs.

2. The model.

By a *directed graph* (or *digraph*) we mean a pair $D=(V, A)$ where V is a finite set and A is a set of ordered pairs $(v, w) \in V \times V$ with $v \neq w$, i.e., a directed graph without loops and at most one arc in each direction joining two vertices. We define a *faulted digraph* to be a pair (D, V_f) where $D=(V, A)$ is a digraph and $V_f \subset V$ represents the set of faulty units. D is called the *underlying digraph* (or *testing digraph*) of the faulted digraph and V_f is called the *fault set.* A *syndrome* of a faulted digraph (D, V_f) is a function $\sigma: A(D) \to \{0, 1\}$ which satisfies for all $(v, w) \in A(D)$:

(1) If $v, w \in V - V_f$ then $\sigma(vw)=0$, and

(2) If $v \in V - V_f$ and $w \in V_f$ then $\sigma(vw)=1$.

Here 0 stands for "okay" and 1 stands for "faulty." In short, a nonfaulty unit's reports must be reliable, while no assumption is placed on the value of $\sigma(vw)$ if v is faulty.

* Received by the editors November 7, 1985; accepted for publication (in revised form) February 27, 1987. This research was supported in part by Office of Naval Research contract N00014-85-K0622.

† Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, Maryland 21218.

Hence we are considering the "permanent fault" case: units are presumed to be in exactly one of the two specified states. In other models, *intermittent* faults are not permitted. We do not consider such "soft" faults.

Denote by $\Sigma(D, V_f)$ the set of all possible syndromes of (D, V_f). Ideally we wish to determine V_f given D and one of the syndromes $\sigma \in \Sigma(D, V_f)$. This is not possible without some restriction on V_f for if $V_f = V(D)$, then $\Sigma(D, V_f)$ contains all functions $\sigma: A(D) \to \{0, 1\}$. One approach to this problem has been to bound $|V_f|$: A digraph D is called (*one-step*) *t-diagnosable* provided $\Sigma(D, V_f) \cap \Sigma(D, W_f) = \varnothing$ whenever $V_f \neq W_f$ where $V_f, W_f \subset V(D)$ with $|V_f| \leqq t$ and $|W_f| \leqq t$. Let $\tau(D)$ denote the greatest t for which D is t-diagnosable.

LEMMA 1 [6], [10]. *Let D be a directed graph with minimum in-degree δ^-; then $\tau(D) \leqq \delta^-$.* □

Indeed, for many digraphs this inequality is an equality (for example if for any pair of vertices at most one arc between them is present, as in [10]).

Now we introduce our probability model which depends on two parameters: p, the probability that one vertex is joined to a second by an arc, and q, the probability of a vertex being faulty. Let $\mathbf{D}(n, p, q)$ denote the sample space consisting of all faulted labeled digraphs (D, V_f) for which $V(D) = \{1, \cdots, n\}$. Probabilities in this finite space are assigned pointwise as follows: Let $(D, V_f) \in \mathbf{D}(n, p, q)$ and assign to it the probability

$$\mathbf{P}(D, V_f) = p^m (1-p)^{n(n-1)-m} q^f (1-q)^{n-f}$$

where

$$n = |V(D)|, \quad m = |A(D)|, \quad f = |V_f|.$$

In other words, for each pair of distinct vertices v and w, the probability that there is an arc from v to w is p, and for each vertex, the probability it is in V_f is q. All these events are independent of one another.

3. Almost sure diagnosability. Suppose one were constructing a system consisting of n devices (vertices) in which one expected to see qn faults. By Lemma 1, one would have to include enough arcs so that $\delta^- \geqq qn$; this would require at least qn^2 arcs. However, it is always possible that more faults will be seen, and so more arcs would be necessary. Indeed, all devices might fail at once and it would be impossible to diagnose any design. One is forced to admit at this point that the probability of *all* units failing is q^n which is vanishingly small. The best one can ever hope for is "almost certain" diagnosability which we formalize below.

A *diagnosis algorithm* is a function A which takes as its "input" two arguments: D, a digraph, and σ, a syndrome (edge labeling by 0's and 1's) so that $\sigma \in \Sigma(D, V_f)$ for some $V_f \subset V(D)$ where $|V_f| < n/2$. The "output" of A is a subset $W_f \subset V(D)$ for which $\sigma \in \Sigma(D, W_f)$. In other words, the algorithm reads in a digraph and 0-1 fault labeling on its arcs and returns a subset of the vertex set which could have been the faulty set of vertices. Note that the "true" set of faulty vertices (V_f) is not part of the input.

(The restriction $|V_f| < n/2$ is necessary to our discussion. Suppose that $|V_f| \geqq n/2$. Even if D had every possible arc between every pair of vertices, the following syndrome is impossible to diagnose: Label each arc between units in V_f and between units in $V - V_f$ with a 0 and label each arc between V_f and $V - V_f$ with a 1. Note that this syndrome is compatible with the fault set being either V_f or $V - V_f$. One cannot

determine the true fault set unless it is known that V_f contains fewer than half the vertices.)

For example, consider the algorithm TRIVIAL which for any D and σ we have

$$\text{TRIVIAL}(D, \sigma) = V(D).$$

TRIVIAL always accuses all units of being faulty. Since every syndrome is compatible with $V_f = V(D)$, TRIVIAL is a diagnosis algorithm (of admittedly limited usefulness).

A more useful algorithm is based on the following observation: A digraph D is called *strongly connected* if for every pair of vertices $x, y \in V(D)$ there exists a directed path from x to y (and likewise, from y to x). The following is (essentially) proved in [10]:

LEMMA 2. *Let σ be a syndrome of a faulted digraph (D, V_f) and let C be a strongly connected subgraph of D in which every arc is labeled* 0 ("*okay*") *by σ. Then either $V(C) \subset V_f$ or $V(C) \subset V(D) - V_f$.*

Proof. If there were vertices x, y in C for which $x \in V(D) - V_f$ and $y \in V_f$ then along a directed path from x to y there would be a good unit testing a faulty unit. The arc between them would have to be labeled "1". □

ALGORITHM FINDFAULT.

(1) Let D^* denote the (spanning) subgraph of D consisting of all arcs labeled "0" (i.e., delete from D all arcs labeled "1"). Find a strongly connected subgraph C of D^* containing more than half the vertices. If none exists, put $W_g \leftarrow \varnothing$ and go to step (4). Otherwise, put $W_g \leftarrow V(C)$. (Here, W_g denotes a set of vertices which the algorithm "knows" must be nonfaulty.)

(2) Find a vertex $v \in V(D) - W_g$ for which (a) there exists a vertex $w \in W_g$ such that (b) $wv \in A(D)$ and (c) $\sigma(wv) = 0$. If none exists, go to step (4).

(3) Put $W_g \leftarrow W_g \cup \{v\}$. Go to step (2).

(4) Let $W_f \leftarrow V(D) - W_g$ and stop.

We now verify that this algorithm is a diagnosis algorithm and that it is "conservative," that is, if σ is a syndrome of (D, V_f) then, provided $|(V_f)| < |V(D)|/2$, FINDFAULT will find all the vertices in V_f (and perhaps some others).

LEMMA 3. *Let (D, V_f) be a faulted digraph with n vertices and $|V_f| < n/2$. Let $\sigma \in \Sigma(D, V_f)$. If $W_f = \text{FINDFAULT}(D, \sigma)$ then $W_f \supset V_f$.*

Proof. If no "large" strongly connected subgraph is found in step (1), then $W_f = V(D) \supset V_f$. Otherwise, let $V_g = V(D) - V_f$ and we observe that it is enough to show that $W_g \subset V_g$ at all times in the algorithm. This is certainly true at the end of step (1) by Lemma 2 since $|V(C)| > n/2$ and $|V_f| < n/2$. Let $v \in V(D) - W_g$ be a vertex satisfying the conditions of step (2). Since there exists a $w \in W_g \subset V_g$ and $\sigma(wv) = 0$ it must be the case that $v \in V_g$, and it follows that, through each pass of steps (2) and (3), $W_g \subset V_g$. □

LEMMA 4. *Let (D, V_f) be a faulted digraph with n vertices and $|V_f| < n/2$. If $\sigma \in \Sigma(D, V_f)$ and $W_f = \text{FINDFAULT}(D, \sigma)$, then $\sigma \in \Sigma(D, W_f)$.*

Proof. As no assumptions are made about the results of a test carried out by a bad unit, we need only consider the value of σ on arcs of the form vw with $v \in W_g = V(D) - W_f$. By Lemma 3, $W_g \subset V_g = V(D) - V_f$. Suppose $w \in W_g \subset V_g$, then $\sigma(vw) = 0$ since $v, w \in V_g$. Next, suppose $w \in W_f$. We must have $\sigma(vw) = 1$ for otherwise w would have been added to W_g by step (2) of the algorithm. □

The following definition is central to our discussion: Let A be a diagnosis algorithm and let (D, V_f) be a faulted digraph. We say that (D, V_f) is *A-diagnosable* provided $V_f = A(D, \sigma)$ for all $\sigma \in \Sigma(D, V_f)$.

(It is possible for (D, V_f) to be A-diagnosable and for there to exist (D, W_f) such that $\Sigma(D, V_f) \cap \Sigma(D, W_f) \neq \emptyset$. For example let D be the digraph with $V = \{v_1, v_2, v_3, v_4, u\}$ and $A = \{v_1v_2, v_2v_3, v_3v_4, v_4v_1\}$. Observe that $(D, \emptyset)$ is *not* FINDFAULT-diagnosable but $(D, \{u\})$ is. Moreover $\Sigma(D, \emptyset) \cap \Sigma(D, \{u\}) \neq \emptyset$ as both sets contain the syndrome σ which labels every arc 0.)

How likely is it that a faulted digraph is A-diagnosable? The answer depends on our choice of sample space $\mathbf{D}(n, p, q)$. If the probability that a faulted digraph in $\mathbf{D}(n, p, q)$ is A-diagnosable tends to 1 as $n \to \infty$, then we say *almost every* (*a.e.*) faulted digraph is A-diagnosable. If, however, this probability tends to 0, we say a.e. faulted digraph is A-undiagnosable. Our principal result is the following.

THEOREM 1. *Suppose q is a constant, $q < \frac{1}{2}$, and p depends on n with $c = \lim_{n\to\infty}(np/\log n)$ (where $c = 0$ and $c = \infty$ are acceptable values). If $c > 1/(1-q)$ then a.e. faulted digraph is* FINDFAULT*-diagnosable. However, if $c < 1/(1-q)$ then for every diagnosis algorithm A, a.e. faulted digraph is A-undiagnosable.*

This theorem has some interesting implications. Suppose we wish to design a system in which each device failed with probability $q < \frac{1}{2}$. Using "absolute" fault detection, we require $O(n^2)$ arcs. However, Theorem 1 implies that if our random directed graph has $O(n \log n)$ arcs, then we can still expect to diagnose almost all fault patterns.

The proof of Theorem 1 rests on the following results. Recall that a *Hamiltonian circuit* is a directed circuit which visits every vertex exactly once. Digraphs which contain Hamiltonian circuits are called *Hamiltonian.* In the random directed graph model in which each of the $n(n-1)$ arcs in a digraph with n vertices can appear with probability p (which may depend on n) we have the following lemma.

LEMMA 5 [8]. *Suppose $\lim_{n\to\infty}(pn/\log n) = c$. If $c < 1$ then almost every digraph is not Hamiltonian and if $c > 1$ then almost every digraph is Hamiltonian.* □

For any subset $X \subset V(D)$ denote by $D - X$ the subgraph of D induced by $V(D) - X$, i.e., one deletes all vertices in X (and all arcs incident with vertices in X).

LEMMA 6. *Let (D, V_f) be a faulted digraph with n vertices and $|V_f| < n/2$. If $D - V_f$ is Hamiltonian, then (D, V_f) is* FINDFAULT*-diagnosable.*

Proof. One checks that for any $\sigma \in \Sigma(D, V_f)$, the FINDFAULT algorithm puts $W_g \leftarrow V - V_f$ immediately in step (1) and finds no further vertices in step (2). □

LEMMA 7. *For fixed q, almost all faulted digraphs in $\mathbf{D}(n, p, q)$ satisfy $|V_f| = nq + o(n)$. In particular, for any $\varepsilon > 0$, we have*

$$\mathbf{P}(nq - n^{(\varepsilon+1)/2} \leqq |V_f| \leqq nq + n^{(\varepsilon+1)/2}) \to 1$$

as $n \to \infty$.

Proof. Let X be the random variable giving the cardinality of V_f in $(D, V_f) \in \mathbf{D}(n, p, q)$. Since memberships in V_f are independent, one readily computes $E(X) = nq$ and $\operatorname{Var}(X) = nq(1-q)$. By Chebyshev's inequality

$$\mathbf{P}(|X - nq| \geqq n^{(1+\varepsilon)/2}) \leqq \frac{\operatorname{Var}(X)}{n^{1+\varepsilon}} = o(1)$$

and the result follows. □

Proof of Theorem 1. Choose $\varepsilon > 0$. Let $x = x(n) = n^{0.6}$. Let Q be the event that $||V_f| - nq| \leqq x$. By Lemma 7, we choose n sufficiently large that $\mathbf{P}(Q) \geqq 1 - \varepsilon$. Suppose the event Q holds. The cardinality of $V_g = V - V_f$ is $N = (1-q)n + o(n)$. For any two vertices in V_g the probability that they are joined by an arc is p and observe that

$$(*) \qquad \frac{pN}{\log N} = \frac{p(1-q)n + po(n)}{\log[(1-q)n + o(n)]} \sim \frac{(1-q)pn}{\log n} \to c(1-q).$$

Now suppose $c > 1/(1-q)$ and let k satisfy $c(1-q) > k > 1$. By Lemma 5 we may suppose that n is sufficiently large that digraphs with $N = (1-q)n + o(n)$ vertices and arc probability $k \log N/N$ are Hamiltonian with probability at least $1-\varepsilon$. Denote by R the event that $D - V_f$ is Hamiltonian. Using (*) we see that $\mathbf{P}(R|Q) \geqq 1-\varepsilon$. Therefore, by Lemma 6,

$$\mathbf{P}((D, V_f) \text{ is FINDFAULT-diagnosable}) \geqq \mathbf{P}(R) \geqq \mathbf{P}(R \cap Q) = \mathbf{P}(R|Q)\mathbf{P}(Q) \geqq (1-\varepsilon)^2.$$

Now suppose $c < 1/(1-q)$. For a vertex v denote by $N^-(v)$ the set of all vertices which have an arc directed to v, i.e., the set of units which test v. Let X denote the number of good units which are tested only by faulty units, that is,

$$X = |\{v \in V_g \colon N^-(v) \subset V_f\}|.$$

We claim that in almost all faulted digraphs $X > ((q-1)/2)n^{1-(1-q)c}$. Let X_i be the 0-1 random variable which is 1 exactly when all units testing v_i are in V_f and $v_i \in V_g$. Observe that

$$E(X_i) = (1-q)[1-(1-q)p]^{n-1} \sim (1-q)n^{-(1-q)c}.$$

Hence,

$$E(X) = \sum_{i=1}^{n} E(X_i) \sim (1-q)n^{1-(1-q)c} \to \infty.$$

Next we compute $E(X^2)$ by considering $E(X_iX_j)$:

$$E(X_iX_j) = (1-q)^2(1-p)^2[1-(1-q)(2p-p^2)]^{n-2}.$$

The $(1-q)^2$ is the probability both vertices are "good," the $(1-p)^2$ term is the probability there is no arc between them. The expression $1-(1-q)(2p-p^2)$ gives the probability that a third vertex is not both good and has an arc to either of v_i or v_j. This gives

$$E(X^2) = \sum_{i,j} E(X_iX_j) \sim n^2(1-q)^2(1-p)^2\left[1 - \frac{2(1-q)c \log n}{n}\right]^n \sim (1-q)^2 n^{2-2c(1-q)}$$
$$\sim E(X)^2.$$

This gives $\operatorname{Var}(X) = o(n^{2-2c(1-q)})$ and finally Chebyshev's inequality yields

$$\mathbf{P}\left(X < \frac{1}{2}(1-q)n^{1-c(1-q)}\right) \leqq \frac{\operatorname{Var}(X)}{(1-q)n^{2-2c(1-q)}/4} \to 0.$$

Simlarly, let Y denote the number of faulty units which are tested only by faulty units. Similar calculations show that in almost all faulted digraphs, $Y > (q/2)n^{1-(1-q)c}$.

Note that for any diagnosis algorithm A, there is no way A can determine from any syndrome the state of the $X+Y$ vertices which are tested only by vertices in V_f. (Indeed, the probability that A correctly assigns these $X+Y$ vertices to the proper sets (V_f and V_g) is no more than $\binom{X+Y}{X}^{-1}$ which is vanishingly small.) Hence, almost all faulted digraphs are A-undiagnosable. Moreover, the probability that A correctly identifies V_f from a syndrome tends to 0 for almost all faulted digraphs. □

Theorem 1 can be interpreted as saying that a "typical" digraph with "more than" $n \log n/(1-q)$ arcs can be accurately diagnosed by the FINDFAULT algorithm almost all the time. This is a great improvement over the qn^2 arcs required in the PMC model which, in practice, cannot diagnose *all* fault possibilities, but which also can only be diagnosed with probability tending to 1. Moreover, in the proof of Theorem 1, it becomes clear that for the "typical" digraph, algorithm FINDFAULT terminates after step (1).

The way in which diagnosis algorithms fail when $c<1/(1-q)$ bears emphasis. The failure is due to the almost certain existence of many units which are tested only by faulty units. It is inherently impossible for *any* algorithm to correctly diagnose such units.

The fact that our testing arcs are random makes it easy to incorporate failed links into our model. Let r denote the probability that a link from one unit to another fails to function (failure meaning the arc gives no label). Then so long as

$$\frac{p(1-r)n}{\log n} \to c > \frac{1}{1-q}$$

almost all faulted digraphs are FINDFAULT-diagnosable.

Finally we consider the implementation of algorithm FINDFAULT. In step (1) we must identify the largest strongly connected subgraph using arcs labeled 0 by the syndrome. This can be accomplished by depth-first search in $O(|V|+|A|)$ time. Since we have $O(n \log n)$ arcs, step (1) takes $O(n \log n)$ time. Our analysis shows that the second step is not executed (almost always) and so, we can expect to diagnose almost all faulted diagraphs in $O(n \log n)$ time, an improvement over $O(n^{2.5})$ in [5].

4. A specific design. In the previous sections our digraphs were *random.* It is unlikely that real systems will be designed at random, and perhaps a nonrandom design would be better. We now restrict our attention to digraphs of a very specific form. Denote by $D_{n,k}$ the directed graph with n vertices $\{v_1, v_2, \cdots, v_n\}$ and arc set

$$A(D_{n,k}) = \{v_i v_j : j = i+\delta, \delta = 1, \cdots, k\}$$

where the addition $i+\delta$ is taken modulo n. One imagines the n units in a circle and each is connected by an arc to the next k in line. For example, $D_{n,1}$ is just a directed cycle. By [10], $\tau(D_{n,k}) = k$. If the unit failure probability is q then one would expect to put $k = nq$ giving qn^2 arcs in the digraph. However, a quadratic number of arcs is unnecessary to guarantee us almost certain diagnosability.

LEMMA 8 [11]. *If* $(D_{n,k}, V_f)$ *has the properties that* (a) $|V_f| < n/2$ *and* (b) *no* k *consecutive vertices are in* V_f, *then* $(D_{n,k}, V_f)$ *is* FINDFAULT-*diagnosable.*

Proof. It is easy to see that the set of nonfaulty processors, $V_g = V(D_{n,k}) - V_f$, induces a Hamiltonian subgraph of $D_{n,k}$ and so, by Lemma 6, $(D_{n,k}, V_f)$ is diagnosable. □

For positive integers $n > k$ and positive real $q < \frac{1}{2}$, define $\mathbf{C}(n, k, q)$ to be the sample space consisting of the faulted digraphs $(D_{n,k}, V_f)$ with probabilities given by:

$$\mathbf{P}(D_{n,k}, V_f) = q^f (1-q)^{n-f}$$

where $f = |V_f|$. Thus the only randomization we consider is in the fault set; the underlying digraph is fixed to be $D_{n,k}$. Each vertex is in V_f with probability q.

Given $q < \frac{1}{2}$, for what k (where k can depend on n) can we expect almost all digraphs in $\mathbf{C}(n, k, q)$ to be (not) diagnosable?

THEOREM 2. *Let* $q < \frac{1}{2}$ *and* $\lim_{n\to\infty}(k/\log n) = c$. *In* $\mathbf{C}(n, k, q)$ *if* $c > (\log q^{-1})^{-1}$ *then almost all faulted digraphs are* FINDFAULT-*diagnosable. However, if* $c < (\log q^{-1})^{-1}$ *then for any diagnosis algorithm A, almost all faulted digraphs are A-undiagnosable.*

Proof. Define a *run* in $(D_{n,k}, V_f)$ to be a set of vertices $\{v_i, v_{i+1}, \cdots, v_j\} \subset V_f$ for which neither v_{i-1} nor v_{j+1} is in V_f. The length of the run is $j-i+1$. (All arithmetic modulo n.) Let X_i be the random variable which is 1 if there is a run of length k beginning at vertex v_i, and 0 otherwise. Put $X = \sum X_i$, the number of runs of length k in $(D_{n,k}, V_f)$.

Suppose $c > (\log q^{-1})^{-1}$. Since $P(X_i = 1) = q^k(1-q)^2$ we have

$$E(X) = nq^k(1-q)^2 \sim nq^{c \log n}(1-q)^2 = n^{1-c \log (1/q)}(1-q)^2 \to 0.$$

Hence, $\mathbf{P}(X = 0) \to 1$ and almost all faulted digraphs have no runs of length k. Therefore, by Lemma 8, almost all faulted digraphs are FINDFAULT-diagnosable.

Suppose now that $c < (\log q^{-1})^{-1}$. We compute $E(X^2)$. Let $d(i,j)$ denote the "mod n difference" of i and j, i.e., $d(i,j) = \min\{|i-j|, n-|i-j|\}$. Then $E(X^2)$ equals

$$\sum_{d(i,j)=0} E(X_iX_j) = E(X) = nq^k(1-q)^2$$

plus

$$\sum_{0<d(i,j)<k+1} E(X_iX_j) = 0$$

plus

$$\sum_{d(i,j)=k+1} E(X_iX_j) = 2nq^{2k}(1-q)^3$$

plus

$$\sum_{d(i,j)>k+1} E(X_iX_j) = [n(n-2k-3)]q^{2k}(1-q)^4$$

from which we have

$$\frac{E(X^2)}{E(X)^2} \leqq \frac{1}{nq^k(1-q)^2} + \frac{2}{n(1-q)} + 1.$$

Observe that $nq^k \sim nq^{c \log n} = n^{1-c \log (1/q)} \to \infty$ thus $E(X^2) \sim E(X)^2$ and so $\operatorname{Var}(X) = o(n^{1-c \log (1/q)})$. By Chebyshev's inequality,

$$\mathbf{P}\left(X < \frac{1}{2}E(X)\right) \leqq \frac{\operatorname{Var}(X)}{E(X)^2}$$

showing that almost all fault patterns give many (order n^α where $\alpha = 1 - c \log (1/q) > 0$) runs of length k. Likewise there are many runs of length $k+1$. Observe that the last vertex in a run of length $k+1$ is a faulty unit tested only by faulty units. Further, the first unit after a run of length k is a good unit tested only by faulty units. As in the proof of Theorem 1, it is impossible for any algorithm to assess the state of these units which are tested only by faulty units. Hence, almost all faulted digraphs are undiagnosable. □

Thus, rather than the roughly qn^2 arcs required in the PMC formulation, we see that $c_q n \log n$ arcs suffice "almost always." It is interesting to note that our random and our deterministic designs both require $O(n \log n)$ arcs. Thus the $D_{n,k}$ digraphs "out perform" the random designs by only a constant factor in the number of arcs. Can we do better?

5. No further improvement. We now consider a third model which generalizes the $\mathbf{C}(n, k, q)$ model above. Suppose we are given an infinite sequence of directed graphs $D_1, D_2, \cdots$ with $|V(D_n)| = n$. Let $a(n) = |A(D_n)|$. For $0 < q < \frac{1}{2}$, let $\mathbf{S}(n, q)$ denote the sample space $\{(D_n, V_f)\colon V_f \subset V(D_n)\}$ with

$$\mathbf{P}(D_n, V_f) = q^f(1-q)^{n-f}$$

where $f = |V_f|$. This generalizes $\mathbf{C}(n, k, q)$ with $D_n = D_{n,k}$.

THEOREM 3. *Let $0 < q < \frac{1}{2}$ and let $D_1, D_2, \cdots$ be a fixed sequence of directed graphs with $|V(D_n)| = n$ and $|A(D_n)| = a(n)$. If $a(n) = o(n \log n)$, then for any diagnosis algorithm A, almost all digraphs in $\mathbf{S}(n, q)$ are A-undiagnosable.*

Proof. Denote by ω_n a function of n which goes to infinity, thus we can write $a(n) = n \log n/\omega_n$. In D_n let V^+ denote the set of vertices with (total) degree exceeding $\log n/\sqrt{\omega_n}$ and let V^- denote the remaining vertices. Observe that

$$2\frac{n \log n}{\omega_n} = 2a(n) \geqq \sum_{v \in V^+} d(v) \geqq |V^+| \frac{\log n}{\sqrt{\omega_n}}$$

and so $|V^+| \leqq 2n/\sqrt{\omega_n}$ and, more importantly,

$$|V^-| \geqq n\left(1 - \frac{2}{\sqrt{\omega_n}}\right) = n(1 - o(1)).$$

We focus our attention on V^- which we assume consists of the vertices $v_1, v_2, \cdots, v_{|V^-|}$. Let X_i denote the random variable which is 1 if $N^-(v_i) \subset V_f$ and $v_i \in V_g$, and is 0 otherwise. Let $X = \sum X_i$. Since $\mathbf{P}(X_i = 1) = (1-q)q^{d^-(v_i)}$, we see that

$$E(X) \geqq n\left(1 - \frac{2}{\sqrt{\omega_n}}\right)(1-q)q^{(\log n)/\sqrt{\omega_n}} \to \infty.$$

Next we compute $E(X^2) = \sum_{i,j} E(X_iX_j)$. We break up this sum into two parts as follows: Let $N_*(i)$ denote the "closed neighborhood" of v_i, i.e., $\{v\} \cup \{w: wv \in A(D_n)$ or $vw \in A(D_n)\}$. If $N_*(i) \cap N_*(j) = \varnothing$ then X_i and X_j are independent, so $E(X_iX_j) = E(X_i)E(X_j)$. Thus we can bound

$$\sum_{N_*(i) \cap N_*(j) = \varnothing} E(X_iX_j) \leqq E(X)^2.$$

If the neighborhoods intersect, we can certainly say that $E(X_iX_j) \leqq 1$. Fortunately the intersection of neighborhoods is a "rare" occurrence. Let $x = \log n/\sqrt{\omega_n}$. For each i, there are at most $1 + x + x^2$ j's for which $N_*(i) \cap N_*(j) \neq \varnothing$. The total number of pairs i, j with intersecting neighborhoods is therefore less than $(n)(2x^2)$. Thus,

$$\sum_{N_*(i) \cap N_*(j) \neq \varnothing} E(X_iX_j) \leqq \frac{2n(\log n)^2}{\omega_n}.$$

Therefore,

$$\frac{E(X^2)}{E(X)^2} \leqq 1 + \frac{2n(\log n)^2/\omega_n}{n^2[1-o(1)]^2(1-q)^2q^{(2\log n)/\sqrt{\omega_n}}} = 1 + o(1).$$

Chebyshev's inequality gives

$$\mathbf{P}\left(X < \frac{E(X)}{2}\right) \leqq \frac{\mathrm{Var}(X)}{E(X)^2/4} \to 0.$$

In short, there are many good units tested only by faulty units. The exact same analysis shows that there are many faulty units tested only by faulty units. Hence, for any diagnosis algorithm, almost all faulted digraphs are undiagnosable. □

We may conclude that $O(n \log n)$ arcs are "almost necessary" and "almost sufficient" in a directed graph with n vertices for diagnosability; and this is true whether the arcs are put in at random or by design.

6. Undirected models. We have not specified *how* one device tests another in our digraphs. This naturally depends on the type of device we are considering and seems to be more of an "engineering" problem than a "mathematical" one. However, the following method seems reasonable for units which are computers. When unit i wishes to test unit j it presents unit j with a job (i.e., program) which it also performs itself.

The test job which unit j performs is assumed to be sufficiently involved that if unit j were malfunctioning it would not execute the program properly. Unit i compares its own output with that of unit j and if the outputs agree, then unit i declares unit j to be healthy. Notice, however, that this output comparison is symmetric: unit j is effectively testing unit i as well. Because of this symmetry, we consider in this section diagnostic problems on undirected graphs. This approach was first considered in [3].

An *undirected graph* (or just *graph*) is a pair (V, E) where V is a finite set and E is a set of two element subsets of V. Thus in this section we only consider graphs which are undirected and have no loops or multiple edges.

Our testing procedure, as described above, consists of two units joined by an edge performing identical, extensive tasks. If the units' outputs agree, then the edge between them is labeled "0" for "okay," but if the outputs disagree the edge is labeled "1". It is clear that if two healthy units test one another we expect a 0 outcome, while if a healthy and a faulty unit test each other we expect a 1. What happens when two faulty units test each other? At this point we have a choice in our model. First, we may suppose that it is improbable that two units fail in the same way and we may safely assume that the outcome must be 1. Or, second, we make no such assumption and consider both 0 and 1 outcomes possible. This gives us two types of syndromes:

A *faulted (undirected) graph* is a pair (G, V_f) with $V_f \subset V(G)$ representing the set of faulty units. A *type* 1 *syndrome* for (G, V_f) is a function $\sigma: E(G) \to \{0, 1\}$ which satisfies, for all $vw \in E(G)$, $\sigma(vw) = 0$ if and only if $v, w \in V(G) - V_f$. A *type* 2 *syndrome* is a function $\sigma: E(G) \to \{0, 1\}$ which satisfies, for all $vw \in E(G)$,

(1) if $v, w \in V(G) - V_f$ then $\sigma(vw) = 0$, and
(2) if $v \in V(G) - V_f$ and $w \in V_f$ then $\sigma(vw) = 1$.

The set of all type 1 and type 2 syndromes for (G, V_f) are denoted by $\Sigma_1(G, V_f)$ and $\Sigma_2(G, V_f)$, respectively. Clearly $\Sigma_1(G, V_f) \subset \Sigma_2(G, V_f)$. A graph G is called *t-diagnosable (type i)* provided whenever $V_f, W_f \subset V(G)$ satisfy $|V_f| \leqq t$, $|W_f| \leqq t$, and $V_f \neq W_f$, then $\Sigma_i(G, V_f) \cap \Sigma_i(G, W_f) = \varnothing$.

It is easy to see that for a graph to be t-diagnosable (in either the type 1 or type 2 sense) its minimum degree must be at least t. Thus a graph with n vertices in which we expect to see qn faulty units must have $qn^2/2$ edges. As in the directed graph case, a quadratic number of edges is unnecessary for "almost sure" diagnosability.

In this section we present results analogous to Theorems 1 and 3 for undirected graphs. As many of the ideas and details are the same we will be more terse in our exposition. We present two algorithms for diagnosis of faulted graphs analogous to the FINDFAULT algorithm of the previous discussion.

ALGORITHM FF1.
Input: A graph G and a type 1 syndrome σ.
Output: A set W_f for which $\sigma \in \Sigma_1(G, W_f)$.
(1) Let W_f be the set of all vertices which are not incident with an edge labeled 0 by σ. Output W_f.

ALGORITHM FF2.
Input: A graph G and a type 2 syndrome σ.
Output: A set W_f for which $\sigma \in \Sigma_2(G, W_f)$.
(1) Find the largest connected component in the spanning subgraph of G consisting of the edges labeled 0. If this includes more than half the vertices of G, let W_g be those vertices; otherwise put $W_g \leftarrow \varnothing$.
(2) Let $W_f \leftarrow V(G) - W_g$ and output W_f.

Just like the FINDFAULT algorithm, one readily checks that these algorithms are "conservative" and correct: Suppose $\sigma \in \Sigma_i(G, V_f)$ and, in case $i=2, |V_f| \leqq |V(G)|/2$. Let W_f be the output of FFi with inputs G and σ. Then $W_f \supset V_f$ and $\sigma \in \Sigma_i(G, W_f)$.

Given a diagnosis algorithm A, a faulted graph (G, V_f) is called *type i A-diagnosable* provided $V_f = A(G, \sigma)$ for all $\sigma \in \Sigma_i(G, V_f)$.

We now introduce a sample space analogous to $\mathbf{D}(n, p, q)$. Let n be a positive integer and $p, q \in (0, 1)$. The sample space $\mathbf{G}(n, p, q)$ consists of all labeled faulted random graphs (G, V_f) with $V(G) = \{1, \cdots, n\}$. Edges in the random graph are present with probability p (and absent with probability $1-p$) and vertices are in V_f with probability q. Typically p depends on n while q is a constant. We adopt the convention that if we are dealing with type 2 syndromes then $0 < q < \frac{1}{2}$. For type 1 syndromes we only assume $0 < q < 1$.

THEOREM 4. *Suppose* $\lim_{n\to\infty} (pn/\log n) = c$. *For* $i = 1, 2$, *if* $c > 1/(1-q)$ *then a.e. faulted graph is type* i FFi*-diagnosable. If* $c < 1/(1-q)$ *then for any diagnosis algorithm* A, *a.e. faulted graph is type* i A*-undiagnosable.*

Proof. Almost surely the cardinality of V_f is $qn + o(n)$ and therefore the cardinality of $V_g = V(G) - V_f$ is $N = (1-q)n + o(n)$. The probability of two vertices in V_g being joined by an edge is p and $pN/\log N \to c(1-q)$ (as in (*) of the proof of Theorem 1). Note that when $c(1-q) > 1$ then $G - V_f$ is almost surely connected. One now checks that both FF1 and FF2 correctly identify V_f.

However, when $c(1-q) < 1$ one shows that there are many good and many fault units tested only by faulty units, and so it is impossible for any diagnosis algorithm to determine their status. □

Theorem 4 is essentially two theorems depending upon the syndrome type (1 or 2) we are considering. The two results in Theorem 4 are similar to those of Theorem 1. That the threshold in all three cases is $p = \log n/n(1-q)$ arises because in the evolution of random graphs the threshold for the disappearance of isolated vertices and the onset of connectivity and Hamiltonicity all occur at $p = \log n/n$.

Theorem 4 shows that $O(n \log n)$ edges are "almost sufficient" for diagnosability. It is also almost necessary.

THEOREM 5. *Let* $G_1, G_2, \cdots$, *be a sequence of graphs for which* $|V(G_n)| = n$ *and* $|E(G_n)| = e(n) = o(n \log n)$. *Suppose* q *is the probability that a vertex of* G_n *is in* V_f. *Then for any diagnosis algorithm* A, *almost all faulted graphs* (G_n, V_f) *are type* i A*-undiagnosable for* $i = 1, 2$.

Proof. The proof of Theorem 3 may be used with virtually no modifications. □

Acknowledgments. The author would like to thank D. Mechtel and G. Masson for introducing him to the field of fault tolerance, and A. Karr and J. Wierman for their helpful comments on this work. The author gives special thanks to Doug Blough whose close reading and comments were invaluable in improving the exposition.

REFERENCES

[1] F. J. ALLAN, T. KAMEDA AND S. TOIDA, *An approach to the diagnosability analysis of a system*, IEEE Trans. Comput., C-25 (1975), pp. 1040–1042.

[2] B. BOLLOBÁS, *Vertices of given degree in a random graph*, J. Graph Theory, 6 (1982), pp. 147–155.

[3] K. Y. CHWA AND S. L. HAKIMI, *Schemes for fault-tolerant computing: a comparison of modularly redundant and t-diagnosable systems*, Inform. and Control, 49 (1981), pp. 212–238.

[4] ———, *On fault identification in diagnosable systems*, IEEE Trans. Comput., C-30 (1981), pp. 414–422.

[5] A. T. DAHBURA AND G. M. MASSON, *An $O(n^{2.5})$ fault identification algorithm for diagnosable systems*, IEEE Trans. Comput., C-33 (1984), pp. 486–492.

[6] S. L. HAKIMI AND A. T. AMIN, *Characterization of connection assignment of diagnosable systems*, IEEE Trans. Comput., C-23 (1974), pp. 86–88.

[7] T. KAMEDA, S. TOIDA AND F. J. ALLAN, *A diagnosing algorithm for networks*, Inform. and Control, 29 (1975), pp. 141–148.

[8] C. MCDIARMID, *General percolation and random graphs*, Adv. in Appl. Probab., 13 (1981), pp. 40–60.

[9] E. M. PALMER, *Graphical Evolution: An Introduction to the Theory of Random Graphs*, John Wiley, New York, 1985.

[10] F. PREPARATA, G. METZE AND R. CHIEN, *On the connection assignment problem of diagnosable systems*, IEEE Trans. Comput., EC-16 (1967), pp. 848–854.

[11] E. R. SCHEINERMAN, *Fault tolerance of graphs with consecutive failure restrictions*, Technical Report No. 444, Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, MD, 1985.

[12] G. SULLIVAN, *A polynomial time algorithm for fault diagnosability*, in Proc. 25th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Publications, October, 1984, pp. 148–156.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987

© 1987 Society for Industrial and Applied Mathematics
011

ON ONE-WAY CELLULAR ARRAYS*

OSCAR H. IBARRA† AND TAO JIANG†

Abstract. There are two simple models of parallel language recognizer: one-way cellular array (OCA) and one-way iterative array (OIA). For inputs of length n, both arrays consist of n identical finite-state machines (cells). The communication between cells is one way, from left to right. The difference in the two models is in the manner in which the input is applied. For the OCA, the input is applied to the cells in parallel. For the OIA, the input is applied serially to the leftmost processor. An input string is accepted if the rightmost cell eventually enters an accepting state. We show that OCA's accept exactly the same class of languages as OIA's. It is relatively easy to show that OIA's can simulate OCA's. The difficult part is the converse, i.e., that OCA's can simulate OIA's. This is rather surprising, since in an OIA, *every* cell of the array has access to each symbol of the input string, whereas in an OCA, the ith cell can only access the first i symbols of the input. This result, when combined with known results concerning OIA's, answers some open questions concerning the computational complexity of OCA's. We also prove some new results concerning linear-time OCA's and OIA's. For example, we show: (1) linear-time OCA's are equivalent to $2n$-time OIA's (note that $2n$-time is optimal for OIA's); (2) the concatenation of a linear-time OCA language with a real-time (i.e. n-time) OCA language is a linear-time OCA language; (3) every bounded language accepted by a one-way multihead nondeterministic pushdown automaton is a linear-time OCA language.

Key words. one-way cellular array, one-way iterative array, real-time, linear-time, alternating Turing machine, computational complexity

AMS(MOS) subject classifications. 68Q80, 68Q15, 68Q45

1. Introduction. One of the simplest models of parallel computation is the one-way cellular array (OCA). An OCA is a one-dimensional array of n identical finite-state machines (cells) that operate synchronously at discrete time steps by means of a common clock (see Fig. 1). The input $a_1a_2 \cdots a_n$, where a_i is in the finite alphabet Σ, is applied to the array in parallel at time 0 by setting the states of the cells to $a_1, a_2, \cdots, a_n$. The state of a cell at time t is a function of its state and the state of its left neighbor at time $t-1$. We can assume that the leftmost cell has an "imaginary" left neighbor whose state is \$ at all times. Thus the leftmost cell can only tell that it is the leftmost cell at time 1. We say that $a_1a_2 \cdots a_n$ is accepted by the OCA if, when given the input $a_1a_2 \cdots a_n$, the rightmost cell eventually enters an accepting state. The OCA has time complexity $T(n)$ if and only if it accepts inputs of length n within $T(n)$ steps. Note that for a nontrivial computation, $T(n) \geqq n$.

Although OCA's have been studied extensively (see, e.g., [CHAN86a], [CHOF84], [DYER80], [HENN61], [IBAR85a], [IBAR85b], [IBAR86], [SEID79], [SMIT70], [SMIT71], [UMEO82]), a precise characterization of their computing power with

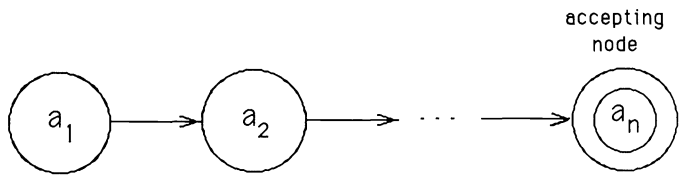


FIG. 1. *An* OCA.

* Received by the editors December 11, 1986; accepted for publication (in revised form) March 16, 1987. This research was supported in part by National Science Foundation grants MCS-8304756, DCR-8420935 and DCR-8604603.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

respect to space- and/or time-bounded Turing machines is not known. For example, it is not known whether or not linear space-bounded deterministic TM's are strictly more powerful than OCA's, although a positive answer seems very likely. On the other hand, it was also not known (until now) whether OCA's could accept languages accepted by polylog space-bounded deterministic TM's. In fact, a question left open in [DYER80] asks whether or not every context-free language can be accepted by an OCA. The main contribution of this paper is a proof of the equivalence of OCA to another simple array model, called one-way iterative array (OIA), whose computing power with respect to TM's has recently been investigated [CHAN86b]. Using the results in [CHAN86b] we are then able to resolve the problems mentioned above as well as other open questions in [DYER80].

The structure of an OIA is very similar to an OCA (see Fig. 2). The only difference between an OIA and an OCA is that in an OIA the input (which has an endmarker) $a_1a_2 \cdots a_n\$$ is fed serially to the leftmost cell. Symbol a_i, $1 \leqq i \leqq n$, is received by the leftmost cell at time $i-1$; after time $n-1$, it receives the symbol \$ (thus \$ is not consumed and available for reading). At time 0, each cell is in a distinguished quiescent state q_0. As in the OCA, the state of a cell at time t is a function of its state and the state of its left neighbor at time $t-1$. For the leftmost cell, the next state depends on its present state and the input symbol. (Thus, the input at time $t-1$ can be considered as the state of an imaginary left neighbor of the leftmost cell at time $t-1$.) We say that $a_1a_2 \cdots a_n$ is accepted by the OIA if, when given the input $a_1a_2 \cdots a_n\$$, the rightmost cell eventually enters an accepting state. The OIA has time complexity $T(n)$ if and only if it accepts inputs of length n within $T(n)$ steps. Again for a nontrivial computation, $T(n) \geqq 2n$.

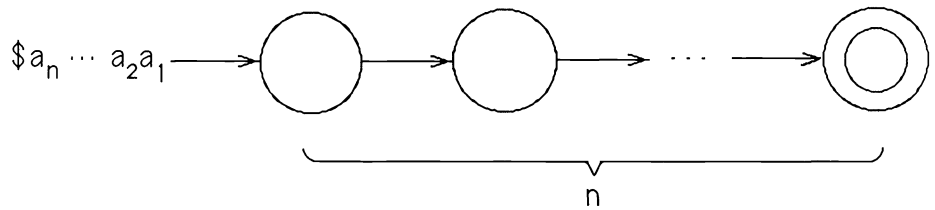


FIG. 2. *An* OIA.

In [CHAN86b], several interesting results concerning OIA's were obtained. We list some of them below.

(1) OIA's can accept PSPACE-complete languages, e.g., the language QBF = set of true quantified Boolean formulas. Thus, the membership problem for OIA's is PSPACE-complete. (See [HOPC79] for the definition of PSPACE-complete.)

(2) Every language accepted by a linear time-bounded alternating TM [CHAN81] can be accepted by an OIA.

(3) Every language accepted by an $n^{1/2}$ space-bounded nondeterministic TM can be accepted by an OIA.

(4) Every language accepted by a two-way multihead nondeterministic pushdown automaton operating in $c^{n/\log n}$ time (for some constant c) can be accepted by an OIA. Thus, in particular, every context-free language can be accepted by an OIA.

(5) The class of languages accepted by OIA's is an AFL closed under intersection, complementation, and reversal. (An AFL is a family of languages containing at least one nonempty language which is closed under the operations of union, concatenation,

Kleene+, ε-free homomorphism, inverse homomorphism, and intersection with regular sets [HOPC79].)

In this paper we show that OCA's accept exactly the same class of languages as OIA's. It is relatively easy to show that OIA's can simulate OCA's. The hard part is the converse, i.e., that OCA's can simulate OIA's. The difficulty arises from the fact that in an OIA, *every* cell of the array has access to each symbol of the input string, whereas in an OCA, the ith cell can only access the first i symbols of the input. It follows that results (1)–(5) above also hold for OCA's. These results answer in the affirmative some open questions in [DYER80], e.g., whether or not OCA languages are closed under operations such as concatenation and reversal, and whether or not OCA's can accept all context-free languages.

We also prove new results concerning linear-time OCA's and OIA's. For example, we show: (1) linear-time OCA's are equivalent to $2n$-time OIA's (note that $2n$-time is optimal for OIA's); (2) the concatenation of a linear-time OCA language with a real-time (i.e. n-time) OCA language is a linear-time OCA language; (3) every bounded language accepted by a one-way multihead nondeterministic pushdown automaton [IBAR74] is a linear-time OCA language. (A language is bounded if it is a subset of $a_1^* a_2^* \cdots a_k^*$ for some distinct symbols $a_1, a_2, \cdots, a_k$.) The second result partially answers the open question of whether or not the class of real-time (linear-time) OCA languages is closed under concatenation [DYER80].

2. Characterizing sequential machines. To avoid getting into unnecessary details, we will only give high-level descriptions of the array operations. There is a nice way to represent the computation of an iterative array or a cellular array in terms of its time-space diagram (or unrolling). The representation can be described in terms of a sequential machine. Sequential machine characterizations of various types of iterative and cellular arrays have been given in [IBAR85a], [IBAR85b], [IBAR86]. In particular, there is a sequential machine, called SMI, which is equivalent to an OIA, and a sequential machine, called SMC, which accepts the reverse of the language accepted by an OCA.

An SMI (see Fig. 3) is a restricted type of on-line single-tape Turing machine. It consists of a semi-infinite worktape (bounded at the left by a special marker $\not\in$) and a finite-state control with an input terminal from which it receives the serial input $a_1 a_2 \cdots a_n\$$. Each a_i, $1 \leqq i \leqq n$, comes from a finite input alphabet Σ, which does not contain \$. The symbol \$ is used as the endmarker for the input.

The SMI operates as follows. Initially, all cells of the worktape (to the right of $\not\in$) contain λ's (where λ is the blank symbol). The read-write head (RWH) makes left-to-right sweeps of the worktape as follows. A left-to-right sweep begins with the RWH scanning $\not\in$ and the machine in a distinguished start state q_0. The machine then reads the input symbol a_i (starting with $i = 1$) and moves right (of $\not\in$) into some state

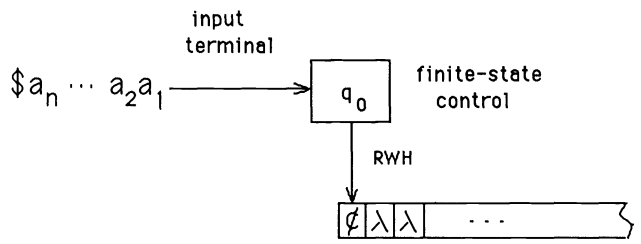


FIG. 3. *An* SMI.

different from q_0. It continues to move right, rewriting symbols scanned by non-λ symbols and changing states (except into q_0), until the RWH scans a λ. When the RWH reaches λ, it rewrites λ by a non-λ symbol different from \$ and resets to the left boundary marker $\not\subset$ in state q_0. Then it begins the next left-to-right sweep. When the endmarker \$ is read, the machine operates just like on any input a_i, except that at the end of the left-to-right sweep, the machine rewrites λ by a right boundary marker \$. The machine then continues making left-to-right sweeps between $\not\subset$ and \$ as before, but without expanding the work space. (The \$ is never rewritten.) We assume that the input endmarker \$ is not consumed when read by the machine, and is always available for reading. The worktape profile of the SMI for 6 (left-to-right) sweeps is shown in Fig. 4. The string $a_1a_2 \cdots a_n$ is accepted by an SMI A if, after reading the input $a_1a_2 \cdots a_n\$$, A eventually enters an accepting state at the end of some left-to-right sweep, i.e., when the RWH reaches the right boundary marker \$. The sweep complexity $S(n)$ of an SMI on string $a_1a_2 \cdots a_n$ is the least number of sweeps needed to accept it. Clearly, if the computation is nontrivial, $S(n) \geqq n+1$.

The structure of an SMC is similar to that of the SMI, except that the worktape of an SMC is bounded at the right (see Fig. 5).

The SMC operates as follows. Initially, all cells of the worktape (to the left of $\not\subset$) contain λ's, and the RWH is on the rightmost λ. The RWH makes left-to-right sweeps of the worktape as follows. A left-to-right sweep begins with the RWH scanning λ and the machine in a distinguished start state q_0. The machine then reads the input symbol a_i (starting with $i=1$), rewrites λ by a non-λ symbol, and moves right into some state different from q_0. When the RWH reaches $\not\subset$, it resets to the rightmost cell which contains λ in state q_0. Then it begins the next left-to-right sweep. When the input endmarker \$ is read, the machine rewrites λ by the left boundary marker \$. The machine then continues making left-to-right sweeps between \$ and $\not\subset$ without expanding

Sweep	Input	Worktape
0		$\not\subset$
1	a_1	$\not\subset Z_1^1$
2	a_2	$\not\subset Z_1^2 Z_2^1$
3	a_3	$\not\subset Z_1^3 Z_2^2 Z_3^1$
4	\$	$\not\subset Z_1^4 Z_2^3 Z_3^2\$$
5	\$	$\not\subset Z_1^5 Z_2^4 Z_3^3\$$
6	\$	$\not\subset Z_1^6 Z_2^5 Z_3^4\$$

FIG. 4. *The worktape profile of an* SMI *on input* $a_1a_2a_3\$$.

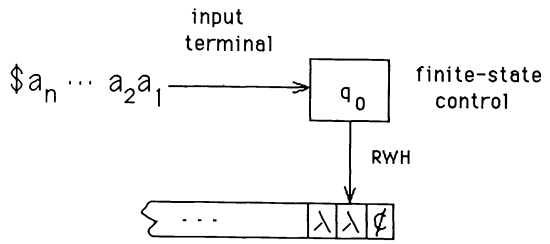


FIG. 5. *An* SMC.

the work space. The worktape profile of the SMC for 6 sweeps is shown in Fig. 6. Like the SMI, the string $a_1a_2 \cdots a_n$ is accepted by the SMC if, when given the input $a_1a_2 \cdots a_n\$$, the machine eventually enters an accepting state at the end of some left-to-right sweep. The sweep complexity $S(n)$ of an SMC on string $a_1a_2 \cdots a_n$ is the least number of sweeps needed to accept it.

The relationships between the arrays and the sequential machines are given by the following theorem.

THEOREM 1. *Let* $S(n) \geqq n+1$.

(1) *A language is accepted by an* OIA *in time* $S(n)+n-1$ *if and only if it can be accepted by an* SMI *with sweep complexity* $S(n)$.

(2) *A language is accepted by an* OCA *in time* $S(n)-1$ *if and only if its reverse can be accepted by an* SMC *with sweep complexity* $S(n)$.

Proof. The proof for (1) is given in [CHAN86b] (see also [IBAR85a]). The proof for (2) uses the same technique and is briefly sketched in the Appendix. □

Sweep	Input	Worktape
0		$\mathrm{Z}_1^1\mathrm{¢}$
1	a_1	$\mathrm{Z}_2^1\mathrm{Z}_1^2\mathrm{¢}$
2	a_2	$\mathrm{Z}_3^1\mathrm{Z}_2^2\mathrm{Z}_1^3\mathrm{¢}$
3	a_3	$\$\mathrm{Z}_3^2\mathrm{Z}_2^3\mathrm{Z}_1^4\mathrm{¢}$
4	\$	$\$\mathrm{Z}_3^3\mathrm{Z}_2^4\mathrm{Z}_1^5\mathrm{¢}$
5	\$	$\$\mathrm{Z}_3^4\mathrm{Z}_2^5\mathrm{Z}_1^6\mathrm{¢}$
6	\$	

FIG. 6. *The worktape profile of an* SMC *on input* $a_1a_2a_3\$$.

3. Equivalence of OCA and OIA. The following theorem, which is rather surprising, is the main result of the paper. As mentioned earlier, although it is easy to show that OIA's can simulate OCA's, it is not obvious how OCA's can simulate OIA's. The difficulty arises from the fact that in an OIA, every cell has access to each symbol of the input string, whereas in an OCA, the ith cell can only access the first i symbols of the input.

THEOREM 2. OCA*'s accept the same class of languages as* OIA*'s.*

Proof. One direction is easy, i.e., an OCA can be simulated by an OIA. By Theorem 1, it is sufficient to show that an SMI can simulate an OCA. The simulation is straightforward. On input $a_1a_2 \cdots a_n\$$, the SMI reads the input and then simulates the state changes in the cells of the OCA on subsequent sweeps.

For the converse, we show that if a language is accepted by an SMI, then its reversal is accepted by an SMC. We first motivate the construction.

Let M_1 be an SMI and M_2 be the SMC to simulate M_1. Suppose the size of the tape alphabet of M_1 is k. Imagine that each cell of M_2 is split into three subcells. M_2 could begin by using markers to partition its subcells into three contiguous regions A, B and C, of equal size. If M_2 had knowledge of n no matter where its RWH is, it could work by moving the input string x into region C, and then using region A as a k^n counter to assist in systematically generating each possible input string y, of size n, in region B. Region C is used to first simulate M_1 on y and then to check if y is the given input string x. However, M_2 can know n only when its RWH is on the right endmarker. This problem can be resolved by iterating the above procedure n times.

In the ith iteration, M_2 works on the first i symbols of x in regions A, B and C of size i. That is, in the ith iteration, all possible strings of length i are generated, simulated and checked against the first i symbols of x.

We now describe the details of the simulation. Let $L \subseteq \Sigma^*$ be a language accepted by an SMI M_1, where Σ is the input alphabet. Let Γ_1 be the worktape alphabet of M_1. Without loss of generality, we can assume that $|\Sigma| = |\Gamma_1| = k$ for some integer $k > 2$. (Note that Γ_1 includes $\not\in$ and \$.) Now we construct an SMC M_2 which accepts L^R (the reverse of L).

The worktape of M_2 has two tracks. Each cell is divided into six subcells, three on the upper track and three on the lower track. The structure of M_2's worktape is shown in Fig. 7.

For a given input $a_n \cdots a_2 a_1\$$ (note that M_2 accepts the reversal of L), M_2 will go through three stages: input stage, simulation stage and dummy stage. M_2 accepts the string $a_n \cdots a_2 a_1$ if and only if it enters an accepting state at the end of the simulation stage. We now describe these stages.

Input stage. This stage consists of the first $n+1$ left-to-right sweeps. On sweep i, $1 \leqq i \leqq n$, M_2 writes the input symbol a_{n-i+1} on the right upper subcell of the corresponding cell. On sweep $n+1$ when it receives the input endmarker \$, M_2 rewrites the blank symbol λ by the left boundary marker \$ and writes the composite marker $(\#_1, \#_2)$ (this will be split into two separate markers $\#_1$ and $\#_2$ later) to the left of a_1 and the marker $\#_3$ to the right of a_1. The worktape configuration of M_2 after the $(n+1)$st sweep is shown in Fig. 8.

Simulation stage. This stage is divided into n phases: phase 1, phase 2, $\cdots$, phase n. It begins with phase 1 (right after the input stage) and ends with phase n. M_2 accepts the input $a_n \cdots a_2 a_1\$$ if and only if it enters an accepting state at the end of phase n. We first give a general description of the operation of M_2 during phase i, $1 \leqq i \leqq n$, and then show that M_2 does indeed go through phase 1, phase 2, $\cdots$, phase n correctly.

In phase i, M_2 checks whether or not M_1 accepts the prefix $a_1 a_2 \cdots a_i$ (of the string $a_1 a_2 \cdots a_n$), by using the following strategy: systematically generate all strings (in Σ^*) of length i, which are called input patterns, and for each of these verify that (1) it is accepted by M_1 and (2) it is $a_1 a_2 \cdots a_i$. It is clear that $a_1 a_2 \cdots a_i$ is accepted by M_1 if and only if there exists some input pattern which satisfies the two conditions.

We now describe the details of phase i. Phase i is divided into k^i subphases, each consisting of k^i left-to-right sweeps (so phase i has a total of k^{2i} left-to-right sweeps).

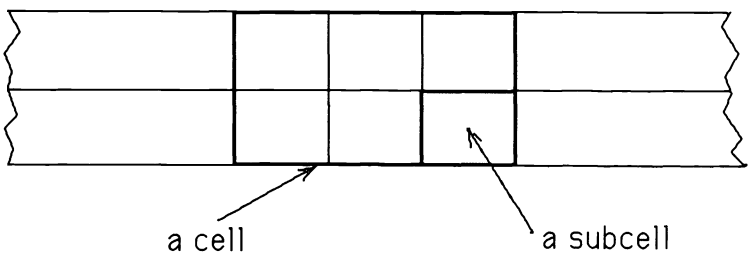


FIG. 7. *The structure of M_2's worktape.*

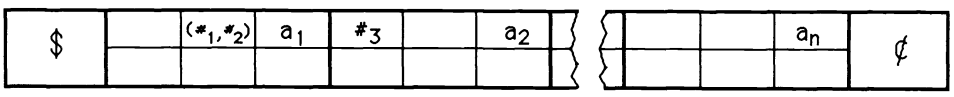


FIG. 8. *The worktape configuration of M_2 after the $(n+1)$st sweep.*

During phase i, the worktape of M_2 is divided into several regions by the markers $\#_1$, $\#_2$ and $\#_3$ as shown in Fig. 9. The regions satisfy: $|\alpha| = |\beta| = |\sigma_1| = |\gamma| = |\sigma_2| = i$, where $|x|$ is the number of subcells contained in region x. Note that region σ_1 includes the subcells containing the markers $\#_1$ and $\#_2$. The functions of these regions are discussed in the following paragraphs.

Region α is used as a counter (called the phase counter) to count the number of left-to-right sweeps M_2 has made since it entered phase i. The phase counter is also used to control the movements of the markers $\#_1$, $\#_2$ and $\#_3$ so that they are always in the "correct" positions. When a carry from the phase counter occurs, M_2 enters phase $i+1$ and $\#_1$ is moved one subcell to the right, $\#_2$ (along with $a_1 a_2 \cdots a_i$) is moved two subcells to the right, and $\#_3$ is moved three subcells to the right (while a_{i+1} remains at the same subcell). Thus the regions α, β, γ, σ_1 and σ_2 are all increased by one when a carry from the phase counter occurs in a left-to-right sweep and the increases are done in the same sweep.

Region β is used as a counter (called the subphase counter) to count the number of left-to-right sweeps in which M_2 has simulated M_1 on the input pattern contained in region γ. When a carry from the subphase counter occurs, M_2 enters the next subphase and the input pattern in region γ is updated. At the same time, M_2 starts to verify that (1) the new input pattern in region γ is accepted by M_1 and (2) the new input pattern is equal to $a_1 a_2 \cdots a_i$.

Region γ is used to systematically generate all input patterns as stated above. Note that the number of strings of length i is k^i, exactly the same as the number of subphases in phase i.

Regions σ_1 and σ_2 are used as the work space for simulating M_1 on the input pattern contained in region γ and checking if the input pattern is equal to $a_1 a_2 \cdots a_i$.

Since there are k^{2i} left-to-right sweeps in phase i, the period of the phase counter is k^{2i}, i.e., the phase counter is a k^2-ary counter. Similarly, the subphase counter is a k-ary counter.

Now we describe in detail how M_2 verifies that the input pattern contained in region γ is accepted by M_1 and the input pattern is equal to $a_1 a_2 \cdots a_i$ in a subphase. Although these two jobs are done simultaneously, we will discuss them separately. Supppose $a_1' a_2' \cdots a_i'$ is the current input pattern in region γ. The worktape of M_2 now looks like the one shown in Fig. 10.

The simulation of M_1 on the input pattern $a_1' a_2' \cdots a_i'$ is as follows. In the first sweep of the subphase (i.e. the sweep in which M_2 enters the subphase), M_2 reads a_1', remembers it in the finite control, and writes a marker, say $\downarrow$, in the subcell above a_2' (thus $\downarrow$ is written in region σ_1). Then it simulates the first sweep of M_1 by using region σ_2 as the work space. In the second sweep, M_2 reads the symbol under the marker $\downarrow$, which is a_2' now, remembers it in the finite control, and moves the marker $\downarrow$ one subcell

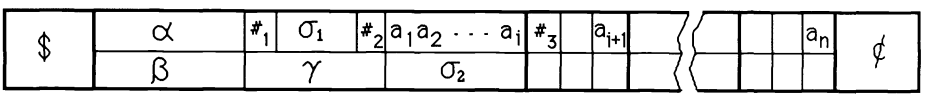


FIG. 9. *The regions on M_2's worktape during phase i.*

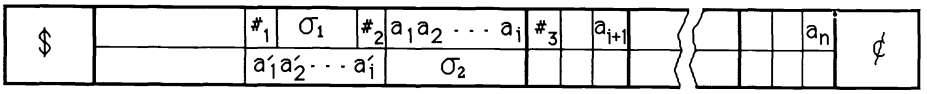


FIG. 10. *The worktape of M_2 during a subphase.*

to the right (so now it is above a_3'). Then it simulates the second sweep of M_1, using region σ_2 as the work space, etc. When the marker $\downarrow$ reaches the marker $\#_2$, $\#_2$ is rewritten by the composite marker $(\downarrow, \#_2)$. In the ith sweep, M_2 reads a_i', remembers it in the finite control, and rewrites $(\downarrow, \#_2)$ by $\#_2$ (i.e. the marker $\downarrow$ is deleted). Then it simulates the ith sweep of M_1 in region σ_2. From now on M_2 will just faithfully simulate the sweeps of M_1 in region σ_2, until the subphase is finished. If M_1 enters an accepting state at the end of some left-to-right sweep, M_2 remembers this fact by writing some special symbol into the rightmost subcell of σ_2. Since $|\Gamma_1| = k$, M_1 accepts $a_1'a_2' \cdots a_i'$ if and only if it enters an accepting state at the end of the jth sweep for some $i+1 \leqq j \leqq k^i$. (Note that $j < k^i$ since Γ_1 includes \$ and $\not\in$ which are not used in generating the patterns.) So, whether or not M_1 accepts $a_1'a_2' \cdots a_i'$ will be remembered by M_2 before the last sweep of the subphase. The special symbol will be erased when the subphase is finished.

The equality of $a_1'a_2' \cdots a_i'$ and $a_1a_2 \cdots a_i$ can be checked in i sweeps, by using the marking technique also. In the first sweep, M_2 checks if $a_1' = a_1$ and marks the symbols a_2' and a_2. In the second sweep, M_2 checks if $a_2' = a_2$ and moves the markers one subcell to the right, etc. In the ith sweep, M_2 checks if $a_i' = a_i$ and, if all these equalities hold, M_2 remembers this fact by writing another special symbol into the rightmost subcell of σ_2. This special symbol will also be erased when the subphase is finished.

Thus at the end of the subphase, M_2 will know (1) whether or not M_1 accepts $a_1'a_2' \cdots a_i'$ and (2) whether or not $a_1'a_2' \cdots a_i'$ is equal to $a_1a_2 \cdots a_i$, by simply checking the existence of the two special symbols described above. If they exist, M_2 replaces them by a new special symbol which will not be erased until phase i is finished. This ensures that M_2 will know whether or not M_1 accepts the prefix $a_1a_2 \cdots a_i$ at the end of phase i, by checking the existence of the new special symbol. When M_2 knows that M_1 accepts $a_1a_2 \cdots a_i$ (i.e. the new special symbol is read), it will remain in an accepting state until the marker $\#_3$ is read. It is easy to see that if marker $\#_3$ is on the worktape, then M_2 will not be in an accepting state at the end of any left-to-right sweep. Thus M_2 could be in an accepting state at the end of a left-to-right sweep only after $\#_3$ is deleted. (We shall see that $\#_3$ will eventually be deleted.)

Suppose the phase counter and the subphase counter are both set to 0 before phase 1 is entered. Then after executing k^2 sweeps of phase 1, a carry from the phase counter occurs, i.e., M_2 enters phase 2. Similarly, M_2 enters phase 3 after k^4 sweeps of phase 2 are executed, etc. M_2 enters phase n after k^{2n-2} sweeps of phase $n-1$ are executed. Thus M_2 goes through phase 1, phase 2, ..., and phase n correctly. When M_2 enters phase n from phase $n-1$, the marker $\#_3$ reaches the right boundary marker $\not\in$ and $\#_3$ is deleted. M_2 will then use the right boundary marker $\not\in$ as $\#_3$. If M_1 accepts $a_1a_2 \cdots a_n$, M_2 will be in an accepting state at the end of the last left-to-right sweep of phase n, i.e., M_2 accepts $a_n \cdots a_2a_1$. On the other hand, if M_1 does not accept $a_1a_2 \cdots a_n$, M_2 will not be in an accepting state at the end of any left-to-right sweep.

Dummy stage. When phase n is finished, the number of subcells in region γ keeps increasing, while the number of subcells in region σ_2 keeps decreasing. So the numbers of subcells in these two regions are no longer equal. M_2 will detect this when it checks if the input pattern in γ matches the string in the region above σ_2 and will not enter an accepting state. It is easy to see that after the marker $\#_2$ reaches the right boundary marker $\not\in$, M_2 will also not be able to enter an accepting state since region σ_2 no longer exists. Thus M_2 will never enter an accepting state during the dummy stage.

From the description, we see that M_2 accepts $a_n \cdots a_2a_1$ if and only if M_1 accepts $a_1a_2 \cdots a_n$. □

Using the results in [CHAN86b] and Theorem 2, we have the following corollary.

COROLLARY 1.

(1) OCA's *can accept* PSPACE*-complete languages. Thus the membership problem for* OCA's *is* PSPACE*-complete.*

(2) *Every language accepted by a linear time-bounded alternating* TM *can be accepted by an* OCA.

(3) *Every language accepted by an* $n^{1/2}$ *space-bounded nondeterministic* TM *can be accepted by an* OCA.

(4) *Every language accepted by a two-way multihead nondeterministic pushdown automaton operating in* $c^{n/\log n}$ *time (for some constant* c*) can be accepted by an* OCA. *Thus, in particular, every context-free language can be accepted by an* OCA.

(5) *The class of languages accepted by* OCA's *is an* AFL *closed under intersection, complementation and reversal.*

One can generalize the OIA and OCA to allow for the arrays to use more cells. Let $S(n) \geqq n$. An $S(n)$-cell OIA [CHAN86b] is shown in Fig. 11. For the case of OCA's, there are two possible generalizations. The first model is called $S(n)$-cell OCA_1 and the second is called $S(n)$-cell OCA_2 (see Fig. 12). It may seem at first that these two models are not equivalent. However, by using Theorem 2, we can show that $S(n)$-cell OCA_1's and $S(n)$-cell OCA_2's have the same power as $S(n)$-cell OIA's. (Note that the language accepted by each of these models is the set of strings accepted with the padding symbols #'s deleted.)

THEOREM 3. $S(n)$*-cell* OCA_1*'s,* $S(n)$*-cell* OCA_2*'s and* $S(n)$*-cell* OIA*'s accept the same class of languages.*

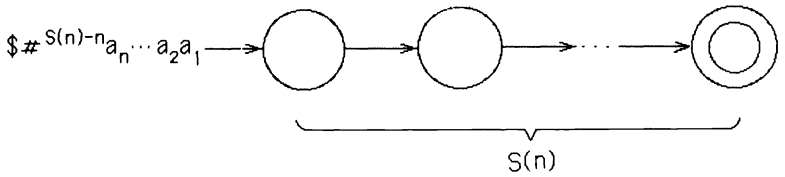


FIG. 11. *An* $S(n)$*-cell* OIA.

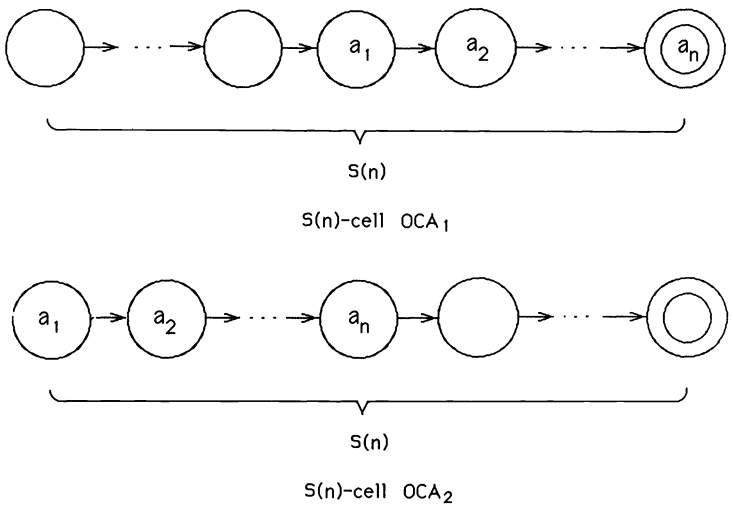


FIG. 12. *The two models of* $S(n)$*-cell* OCA.

Proof. We first show that $S(n)$-cell OCA_2's accept the same class of languages as $S(n)$-cell OIA's. It is easy to see that every language accepted by an $S(n)$-cell OIA can be accepted by an $S(n)$-cell OCA_2. Let L_1 be a language accepted by an $S(n)$-cell OIA. Define another language L_2 as follows: $L_2 = \{a_1a_2 \cdots a_n \#^{S(n)-n} \mid a_1a_2 \cdots a_n \in L_1\}$. Then it is easy to see that L_2 is accepted by an OIA. By Theorem 2, L_2 can be accepted by an OCA, i.e., L_1 can be accepted by an $S(n)$-cell OCA_2.

Now we show that $S(n)$-cell OCA_1's accept the same class of languages as $S(n)$-cell OIA's. Again, we only show that every language accepted by an $S(n)$-cell OIA can be accepted by an $S(n)$-cell OCA_1 since the converse is straightforward. Let L_1 be a language accepted by an $S(n)$-cell OIA. Then $L_2 = \{a_1a_2 \cdots a_n \#^{S(n)-n} \mid a_1a_2 \cdots a_n \in L_1\}$ is accepted by an OIA. Let $L_3 = \{a_na_{n-1} \cdots a_1 \#^{S(n)-n} \mid a_1a_2 \cdots a_n \in L_1\}$. It is easy to show that L_3 can be accepted by an OIA. Then by Theorem 1, L_3 can be accepted by an OCA. Since the class of OCA languages is closed under reversal by Corollary 1, L_3^R is accepted by an OCA, i.e., L_1 is accepted by an $S(n)$-cell OCA_1. □

COROLLARY 2. *Let $S(n) \geqq n$ and $T(n) \geqq n$.*

(1) *Every language accepted by a $T(n)$ time-bounded alternating* TM *can be accepted by a $T(n)$-cell* OCA_1 (OCA_2).

(2) *Every language accepted by an $S(n)$ space-bounded nondeterministic* TM *can be accepted by an $S^2(n)$-cell* OCA_1 (OCA_2).

(3) *Every language accepted by a nondeterministic auxiliary pushdown automaton* [COOK71] *which is simultaneously $S(n)$ space-bounded and $T(n)$ time-bounded can be accepted by an $S(n)$ log $T(n)$-cell* OCA_1 (OCA_2).

Proof. From Theorem 3 and the fact that these results hold for $S(n)$-cell OIA's [CHAN86b]. □

COROLLARY 3. *The class of languages accepted by n^r-cell OCA_1's (OCA_2's) is properly contained in the class of languages accepted by n^s-cell OCA_1's (OCA_2's) for any real constants $1 \leqq r < s$.*

Proof. This hierarchy is true for polynomial-cell OIA's [CHAN86b]. □

Theorem 2 can also be generalized to two-dimensional analogues of OCA's and OIA's. A one-way two-dimensional iterative array (OTIA) and a one-way two-dimensional cellular array (OTCA) are shown in Fig. 13.

THEOREM 4. OTCA's *accept the same class of languages as* OTIA's.

Proof. Similar to the proof of Theorem 2, using the sequential machine characterizations of OTIA's and OTCA's in [IBAR85a]. □

4. Linear-time (real-time) OCA's and linear-time (pseudo-real-time) OIA's. A linear-time OCA is one which accepts inputs of length n within time cn for some fixed real constant $c \geqq 1$. When $c = 1$, the OCA is called real-time. Thus, for real-time computation, the sequential machine, SMC, characterizing the OCA makes exactly $n+1$ sweeps. It is easy to show that in this case, the SMC does not need the right endmarker \$, and therefore can operate in exactly n sweeps—thus, acceptance/rejection occurs at the end of the sweep after reading the last input symbol. (The idea is for the SMC to assume that the current input symbol is followed by \$.) It is easy to show that linear-time OCA's are more powerful than real-time OCA's. For example, the language $\{a^{2^n} \mid n \geqq 1\}$ can be accepted by a linear-time OCA but not by a real-time OCA [CULI84]. Another example of a linear-time OCA language which does not appear to be real-time recognizable is $\{xx \mid x \text{ in } \{0, 1\}^+\}$. This language was shown to be linear-time recognizable by an OCA in [DYER80] by direct construction of the OCA. A construction using an

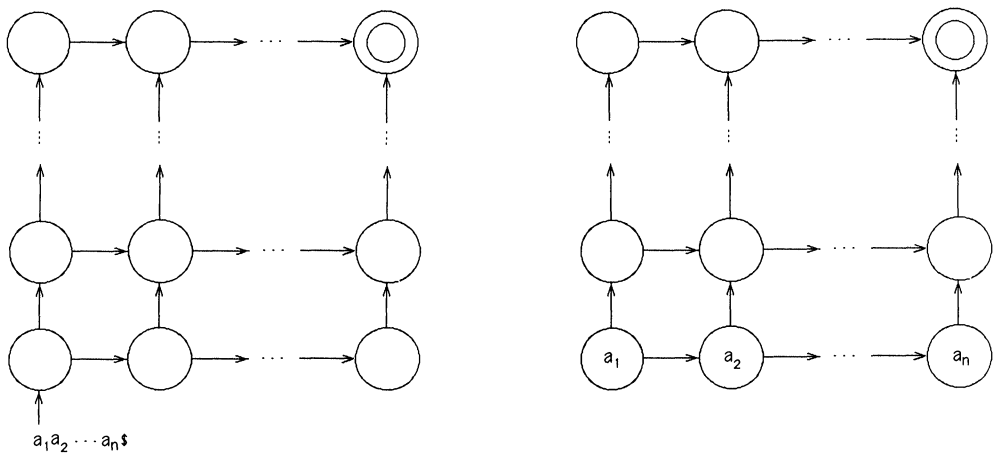


FIG. 13. *An* OTIA (*left*) *and an* OTCA (*right*).

SMC is relatively simpler. The SMC when given input $a_6a_5a_4a_3a_2a_1\$$, checks that $a_6a_5a_4$ is equal to $a_3a_2a_1$ as follows. First, it copies the string $a_6a_5a_4a_3a_2a_1$ in reverse. Then the operation of the SMC on subsequent sweeps (after the first 6 sweeps) is shown in Fig. 14. The machine accepts if the string in the upper track is equal to that of the lower track.

A linear-time OIA is one which accepts inputs of length n within time cn for some fixed real constant $c \geqq 2$. When $c = 2$, the OIA is called pseudo-real-time. Again, for the pseudo-real-time case, the right endmarker on the input for the characterizing

Sweep	Worktape							
6		a_1	a_2	a_3	a_4	a_5	a_6	¢
7	\$	-	$\begin{pmatrix} a_2 \\ a_1 \end{pmatrix}$	a_3	a_4	a_5	a_6	¢
8	\$	-	$\begin{pmatrix} - \\ a_1 \end{pmatrix}$	$\begin{pmatrix} a_3 \\ a_2 \end{pmatrix}$	a_4	a_5	a_6	¢
9	\$	-	-	$\begin{pmatrix} a_3 \\ a_1 \end{pmatrix}$	$\begin{pmatrix} a_4 \\ a_2 \end{pmatrix}$	a_5	a_6	¢
10	\$	-	-	$\begin{pmatrix} - \\ a_1 \end{pmatrix}$	$\begin{pmatrix} a_4 \\ a_2 \end{pmatrix}$	$\begin{pmatrix} a_5 \\ a_3 \end{pmatrix}$	a_6	¢
11	\$	-	-	-	$\begin{pmatrix} a_4 \\ a_1 \end{pmatrix}$	$\begin{pmatrix} a_5 \\ a_2 \end{pmatrix}$	$\begin{pmatrix} a_6 \\ a_3 \end{pmatrix}$	¢

FIG. 14. *The operation of the* SMC *on input* $a_6a_5a_4a_3a_2a_1\$$.

sequential machine SMI is not necessary. Thus, the number of sweeps of the SMI reduces to n.

In this section, we derive some new results concerning linear-time (real-time) OCA's and linear-time (pseudo-real-time) OIA's. We show that linear-time OCA's are equivalent to pseudo-real-time OIA's. It is an open question whether the class of linear-time (real-time) OCA languages is closed under concatenation [DYER80]. Here, we show that the concatenation of a linear-time (hence, also real-time) OCA language with a real-time OCA language is a linear-time OCA language. We also show that the class of semilinear languages [HARR78], [IBAR74] can be accepted by linear-time OCA's and hence also by pseudo-real-time OIA's. It follows that, e.g., every bounded language accepted by a one-way nondeterministic multihead pushdown automaton can be accepted by a linear-time OCA. The proofs are facilitated by the use of the characterizing sequential machines.

We begin with the following speed-up theorem for OCA's and OIA's.

THEOREM 5. *Let M be an* OCA (OIA) *operating in time $n+R(n)$ $(2n+R(n))$, $R(n)\geqq 0$. We can effectively construct an* OCA (OIA) *equivalent to M which operates in time $n+R(n)/k$ $(2n+R(n)/k)$ for any positive integer k.*

Proof. The sequential machine SMC (SMI) characterizing M can be speeded-up by making the machine simulate k sweeps in one sweep, after the first n sweeps. □

It follows from the above theorem that any linear-time OCA (OIA) can be converted to one which operates in time $(1+\varepsilon)n$ $((2+\varepsilon)n)$ for any positive real constant ε.

THEOREM 6. *Linear-time* OCA*'s are equivalent to pseudo-real-time* OIA*'s.*

Proof. Let M_1 be an SMC which makes a linear number of sweeps. By the speed-up theorem, we may assume that M_1 makes at most $2n-1$ sweeps on inputs of length n. We can easily construct a pseudo-real-time SMI M_2 accepting the reverse of the language accepted by M_1. We illustrate the construction by an example. Figure 15 shows the worktape profile of M_1 on input $a_1a_2a_3a_4\$$. M_2 simulates M_1 along the dashed lines. The worktape profile of M_2 on input $a_4a_3a_2a_1$ is shown in Fig. 16. To simplify the figures, the state changes of M_1 (M_2) are not shown.

For the converse, assume that M_2 is a pseudo-real-time SMI operating in n sweeps (i.e. there is no input endmarker). Figure 17 shows the worktape profile of M_2 on input $a_1a_2a_3a_4$. Then the corresponding $2n$-sweep SMC M_1 simulating M_2 is shown in Fig. 18. □

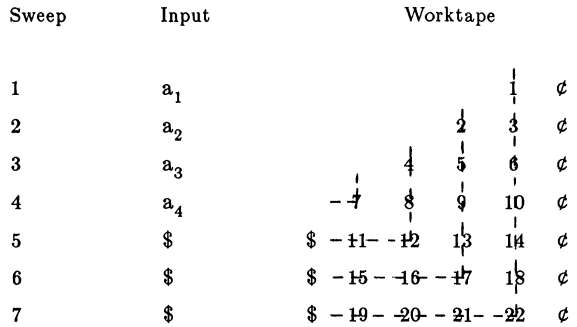


FIG. 15. *The worktape profile of M_1 on input $a_1a_2a_3a_4\$$.*

Sweep	Input	Worktape				
1	a_4	¢	(-,-,7)			
2	a_3	¢	(4,8,11)	(-,-,12)		
3	a_2	¢	(2,5,15)	(9,13,16)	(-,-,17)	
4	a_1	¢	(1,3,19)	(6,10,20)	(14,18,21)	(-,-,22)

FIG. 16. *The worktape profile of M_2 on input $a_4a_3a_2a_1$.*

Sweep	Input	Worktape				
1	a_1	¢	1			
2	a_2	¢	2	3		
3	a_3	¢	4	5	6	
4	a_4	¢	7	8	9	10

FIG. 17. *The worktape profile of M_2 on input $a_1a_2a_3a_4$.*

Sweep	Input	Worktape					
1	a_4					a_4	¢
2	a_3				a_3	a_4	¢
3	a_2			a_2	a_3	a_4	¢
4	a_1		a_1	a_2	a_3	a_4	¢
5	\$	\$	1	2	4	7	¢
6	\$	\$	-	3	5	8	¢
7	\$	\$	-	-	6	9	¢
8	\$	\$	-	-	-	10	¢

FIG. 18. *The worktape profile of M_1 on input $a_4a_3a_2a_1$\$.*

Next, we prove the concatenation result.

THEOREM 7. *The concatenation of a linear-time* OCA *language with a real-time* OCA *language is a linear-time* OCA *language.*

Proof. Let L_1 be a linear-time OCA language and L_2 be a real-time OCA language. Then by Theorem 1 and the speed-up theorem, L_1^R can be accepted by a $2n$-space SMC M_1 and L_2^R can be accepted by an n-sweep SMC M_2 (i.e., there is no input endmarker). We show that $L_2^R L_1^R$ can be accepted by a $3n$-sweep SMC M_3 and, hence, by Theorem 1 L_1L_2 is a linear-time OCA language. Without loss of generality, we assume that M_1 accepts strings of length n in exactly $2n$ sweeps. Again, we illustrate the construction by an example. Figure 19 shows the operation of M_1 when given input $a_1a_2a_3a_4a_5$\$. Figure 20 shows the operation of M_2 on input $a_1a_2a_3a_4a_5$. In these figures a circled symbol means that the machine enters an accepting state after writing the symbol.

The operation of M_3 consists of two stages (see Fig. 21). In the first stage (before the input endmarker \$ is received), M_3 simulates M_1 faithfully and records the string $a_5a_4a_3a_2a_1$ on its worktape. When \$ is received, it continues simulating M_1 and, at the

Sweep	Input	Worktape						
1	a_1						1′	¢
2	a_2					2′	3′	¢
3	a_3				4′	5′	6′	¢
4	a_4			7′	8′	9′	10′	¢
5	a_5		11′	12′	13′	14′	15′	¢
6	\$	\$	16′	17′	18′	19′	20′	¢
7	\$	\$	21′	22′	23′	24′	25′	¢
8	\$	\$	26′	27′	28′	29′	30′	¢
9	\$	\$	31′	32′	33′	34′	35′	¢
10	\$	\$	36′	37′	38′	39′	40′	¢

FIG. 19. *The worktape profile of M_1 on input $a_1a_2a_3a_4a_5$\$.*

Sweep	Input	Worktape
1	a_1	1′ ¢
2	a_2	2′ 3′ ¢
3	a_3	4′ 5′ 6′ ¢
4	a_4	7′ 8′ 9′ 10′ ¢
5	a_5	11′ 12′ 13′ 14′ 15′ ¢

FIG. 20. *The worktape profile of M_2 on input $a_1a_2a_3a_4a_5$.*

Sweep	Input						
1	a_1						$(a_1,1')$ ¢
2	a_2					$(a_2,2')$	$(a_1,3')$ ¢
3	a_3				$(a_3,4')$	$(a_2,5')$	$(a_1,6')$ ¢
4	a_4			$(a_4,7')$	$(a_3,8')$	$(a_2,9')$	$(a_1,10')$ ¢
5	a_5		$(a_5,11')$	$(a_4,12')$	$(a_3,13')$	$(a_2,14')$	$(a_1,15')$ ¢
6	\$	\$	(11,-,16′)	$(a_4,17')$	$(a_3,18')$	$(a_2,19')$	$(a_1,20')$ ¢
7	\$	\$	(-,11,21′)	(7,-,22′)	$(a_3$*,23′)	$(a_2,24')$	$(a_1,25')$ ¢
8	\$	\$	(-,-,26′)	(12,7,27′)	(4*,-,28′)	$(a_2,29')$	$(a_1,30')$ ¢
9	\$	\$	(-,-,31′)	(-,12,32′)	(8,4*,33′)	(2,-,34′)	$(a_1$*,35′) ¢
10	\$	\$	(-,-,36′)	(-,-,37′)	(13,8,38′)	(5*,2,39′)	(1*,-,40′) ¢
11	\$	\$	(-,-,41′)	(-,-,42′)	(-,13,43′)	(9,5*,44′)	(3,1*,45′) ¢
12	\$	\$	(-,-,46′)	(-,-,47′)	(-,-,48′)	(14,9,49′)	(6*,3,50′) ¢
13	\$	\$	(-,-,51′)	(-,-,52′)	(-,-,53′)	(-,14,54′)	(10,6*55′) ¢
14	\$	\$	(-,-,56′)	(-,-,57′)	(-,-,58′)	(-,-,59′)	(15,10,60′) ¢
15	\$	\$	(-,-,61′)	(-,-,62′)	(-,-,63′)	(-,-,64′)	(-,15, 65′) ¢

FIG. 21. *The worktape profile of M_3 on input $a_1a_2a_3a_4a_5$\$.*

same time, starts simulating M_2 along the dashed lines. Since the strings a_4a_5 and $a_2a_3a_4a_5$ are accepted by M_1, a_3 and a_1 are marked by asterisks during the 7th and 9th sweeps, respectively. In the following sweeps, the asterisks are shifted as shown in the figure. M_3 enters an accepting state if M_2 enters an accepting state and the current symbol scanned by M_1 is marked by an asterisk. □

For the next result, we need the notion of a semilinear language.

Let **N** be the set of natural numbers and k be a positive integer. A set $Q \subseteq \mathbf{N}^k$ is a semilinear set [HARR78] if there exist positive integers $m, r_1, \cdots, r_m$ and vectors v_j^i in $\mathbf{N}^k$ for $1 \leqq i \leqq m$, $0 \leqq j \leqq r_i$ such that

$$Q = \bigcup_{i=1}^{m} \left\{ v_0^i + \sum_{j=1}^{r_i} t_j v_j^i \,\middle|\, t_j \text{ in } \mathbf{N} \right\}.$$

The language, L_Q, defined by Q is the set $\{a_1^{x_1} \cdots a_k^{x_k} | (x_1, \cdots, x_k) \text{ in } Q\}$, where $a_1, \cdots, a_k$ are distinct symbols. L_Q is called a semilinear language.

THEOREM 8. *Every semilinear language can be accepted by a pseudo-real-time* OIA (*and hence also by a linear-time* OCA).

Proof. Let

$$Q = \bigcup_{i=1}^{m} \left\{ v_0^i + \sum_{j=1}^{r_i} t_j v_j^i \,\middle|\, t_j \geqq 0 \right\}$$

where $v_j^i \in \mathbf{N}^k$. Suppose we want to know if $(x_1, \cdots, x_k)$ is in Q. Then we have to check whether there is an $i \in \{1, \cdots, m\}$ and $t_1, \cdots, t_{r_i} \in \mathbf{N}$ such that $(x_1, \cdots, x_k) = v_0^i + \sum_{j=1}^{r_i} t_j v_j^i$. This process involves trying, for each i, to solve a diophantine system (in the nonnegative integer variables $t_1, \cdots, t_{r_i}$). The efficient solution of such a system depends on the following result from [GURA79] (see also [CHAN83]).

Let $A\bar{y} = \bar{b}$ be a system of linear equations, where A is an $m \times n$ integral matrix, $\bar{y} = (y_1, \cdots, y_n)^T$ is a column vector of variables, and $\bar{b} = (b_1, \cdots, b_m)^T$ is an integral column vector. Let $r \leqq m$ be the rank of A. Denote by Ψ the maximum of the absolute values of all $r \times r$ subdeterminants of A. If the system has a nonnegative integral solution, then it has a nonnegative integral solution $(\hat{y}_1, \cdots, \hat{y}_n)^T$ such that for some set of indices $I = \{l_1, \cdots, l_r\} \subseteq \{1, \cdots, n\}$, $\hat{y}_i < \Psi$ for each $i \notin I$. Moreover, the submatrix formed by columns $l_1, \cdots, l_r$ of A is nonsingular (i.e. it has rank r).

Thus, with the semilinear set Q is associated a finite set of nonsingular systems each of which arises by "predetermining" some of the t_j's. For each $\bar{x} = (x_1, \cdots, x_k)^T \in \mathbf{N}^k$, one need only try each such system to see if the remaining t_j's are solvable in nonnegative integers.

The solution of each of the nonsingular systems can be effected by applying Cramer's rule to a square nonsingular subsystem. Thus each "nonpredetermined" t_j can be written as

$$t_j = \frac{\sum a_p y_p - \sum b_p z_p + c}{\Delta} \tag{1}$$

where y_p, z_p are components of $\bar{x}$, a_p, b_p and Δ are positive integers, and c is an integer. Here a_p, b_p and Δ depend only on the vectors defining the underlying semilinear set, whereas c depends on the predetermined t_j's as well. However, the t_j's so obtained constitute a solution if and only if they are nonnegative integers which also satisfy any rows that were deleted to obtain the square nonsingular subsystem. Thus, by

substituting the "predetermined" t_j's and the t_j's of (1) for the remaining equations, we get equations of the form

$$\sum a_p y_p - \sum b_1 z_q + c = 0 \tag{2}$$

where y_p, z_q, a_p, b_q and c are as in (1).

Now, clearly, the class of pseudo-real-time OIA languages is closed under union and intersection. Hence, to prove that a pseudo-real-time OIA can accept L_Q, it is sufficient to show that a pseudo-real-time OIA when given input $a_1^{x_1} \cdots a_k^{x_k}$ can verify equations of forms (2) and (3) below (derived from (1)).

$$\frac{\sum a_p y_p - \sum b_q z_q + c}{\Delta} \text{ is a nonnegative integer.} \tag{3}$$

We only consider (3). Equations of form (2) can be handled similarly. Equation (3) can be verified by checking that

(a) $\sum a_p y_p - \sum b_q z_q + c$ is divisible by Δ;

(b) $\sum a_p y_p - \sum b_q z_q + c$ is nonnegative.

The integral test (a) can be done by a finite automaton computing mod Δ, and hence can be done by an SMI (characterizing a pseudo-real-time OIA) in $n+1$ sweeps. As for (b), an SMI can compute the positive combination on one track of the worktape and the negative combination on another track. This computation can be done as the input is read, packing d unary symbols per cell of the worktape for some fixed integer d. When the input endmarker is read, the SMI checks in one sweep whether (b) is satisfied. □

A one-way h-head NPDA (h-NPDA) is a nondeterministic pushdown automaton with h independent one-way (read-only) input heads [HARR68]. The heads are "nonsensing" in that the heads cannot sense the presence of other heads on the same input position. As is well known, 1-NPDA's accept exactly the context-free languages [HOPC79].

COROLLARY 4. *Let L be a language accepted by an h-*NPDA, *and $L \subseteq a_1^* \cdots a_k^*$ for distinct symbols $a_1, \cdots, a_k$. Then L can be accepted by a pseudo-real-time* OIA (*and hence also by a linear-time* OCA).

Proof. It is known that a language $L \subseteq a_1^* \cdots a_k^*$ is accepted by an h-NPDA if and only if it is a semilinear language [IBAR74]. □

5. Concluding remarks. It remains an open problem whether or not OCA's have the same power as two-way cellular arrays (CA's). In [DYER80], it was conjectured that every language accepted by an n^2-cell OCA_1 can be accepted by a CA (thus OCA's are strictly weaker than CA's). But from Corollary 2 this would imply that $\text{NSPACE}(n) = \text{DSPACE}(n)$, which is a long-standing open problem in complexity theory, and many conjecture the answer to be negative.

Another open problem is whether or not the class of real-time (linear-time) OCA languages is closed under concatenation. Although it may seem that the answer is positive for the real-time case (from Theorem 7), we conjecture that it is not. For let $L = \{0^n 1^n \mid n > 0\}$. Then both L and L^R can be accepted by real-time OCA's. However, it seems unlikely that LL^R can be accepted by a real-time OCA. From Theorems 6 and 7, we know that the class of linear-time OCA languages is closed under concatenation if and only if the class of pseudo-real-time OIA languages is closed under concatenation. Currently, we do not even know if the concatenation of two pseudo-real-time OIA languages is a linear-time OIA language.

Appendix.

Proof of Theorem 1 (2). Figure A1 shows an OCA A with input $a_1a_2a_3$. Its time-space unrolling is shown in Fig. A2. The corresponding worktape profile of an SMC simulating this unrolling is shown in Fig. A3.

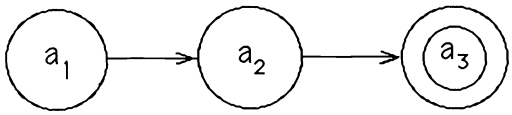


FIG. A1. OCA *A*.

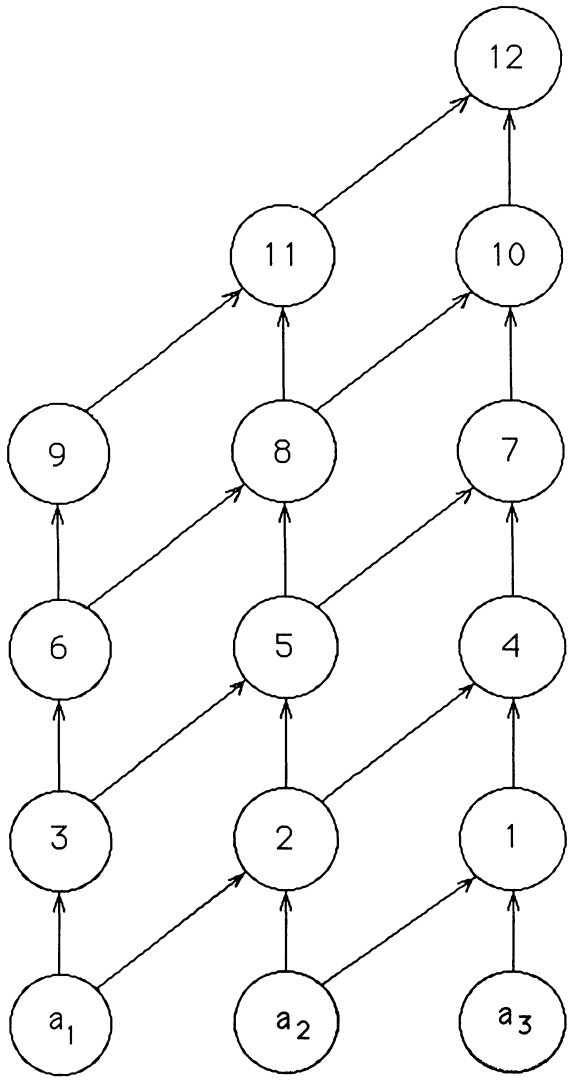


FIG. A2. *Time-space unrolling of A.*

For the converse, given the worktape profile of an SMC M in Fig. A4, the corresponding time-space unrolling is shown in Fig. A5. In Fig. A5, q_i is the state the machine M enters after writing symbol i on the tape. □

Acknowledgment. We thank the referee for his comments which improved the presentation of Theorem 2.

Input	Worktape
	¢
a_3	a_3 ¢
a_2	a_2 1 ¢
a_1	a_1 2 4 ¢
\$	\$ 3 5 7 ¢
\$	\$ 6 8 10 ¢
\$	\$ 9 11 12 ¢

FIG. A3. *Corresponding worktape profile of the simulating* SMC.

Input	Worktape
	¢
a_1	1 ¢
a_2	2 3 ¢
a_3	4 5 6 ¢
\$	\$ 7 8 9 ¢
\$	\$ 10 11 12 ¢
\$	\$ 13 14 15 ¢

FIG. A4. *Worktape profile of M.*

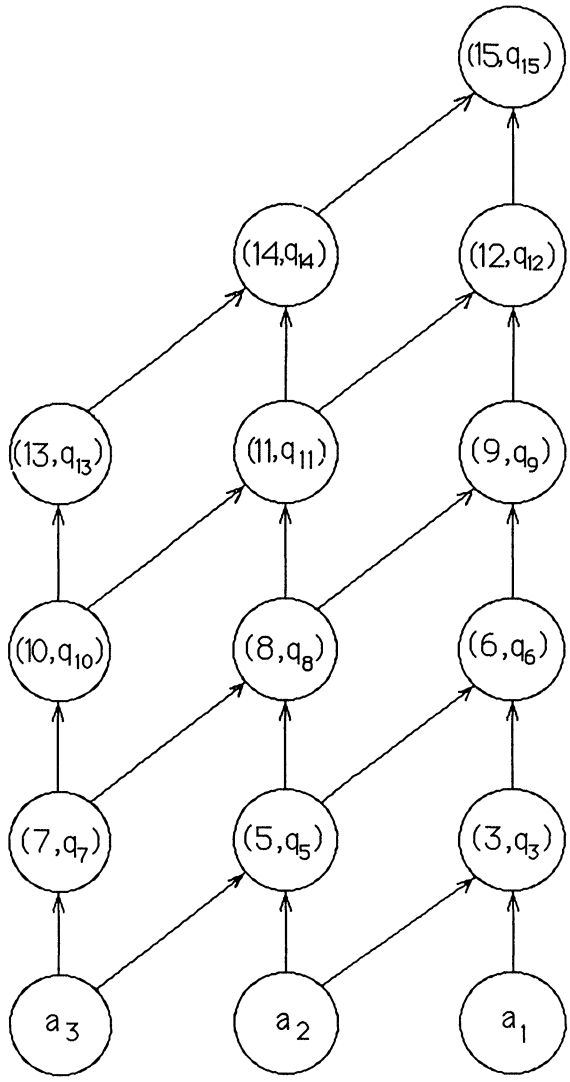


FIG. A5. *Corresponding time-space unrolling of the simulating* OCA.

REFERENCES

[CHAN81] A. CHANDRA, D. KOZEN AND L. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[CHAN83] T. CHAN AND O. IBARRA, *On the space and time complexity of functions computable by simple programs*, this Journal, 12 (1983), pp. 708-716.

[CHAN86a] J. CHANG, O. IBARRA AND M. PALIS, *Efficient simulations of simple models of parallel computation by space-bounded* TMs *and time-bounded alternating* TMs, Revision of Tech. Report 85-47, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN, Nov. 1985; submitted to Theoret. Comput. Sci.

[CHAN86b] J. CHANG, O. IBARRA AND A. VERGIS, *On the power of one-way communication*, Proc. 27th IEEE Annual Symp. on Foundations of Computer Science, 1986, pp. 455-464.

[CHOF84] C. CHOFFRUT AND K. CULIK III, *On real-time cellular automata and trellis automata*, Acta Inform., 21 (1984), pp. 393-409.

[COOK71] S. COOK, *Characterizations of pushdown machines in terms of time-bounded computers*, J. Assoc. Comput. Mach., 18 (1971), pp. 4-18.

[CULI84] K. CULIK II, J. GRUSKA AND A. SALOMAA, *Systolic trellis automata, Part* I, Internat. J. Comput. Math., 15 (1984), pp. 195–212.

[DYER80] C. DYER, *One-way bounded cellular automata*, Inform. Control, 44 (1980), pp. 54–69.

[GURA79] E. GURARI AND O. IBARRA, *The complexity of the equivalence problem for two characterizations of Presburger sets*, Theoret. Comput. Sci., 13 (1979), pp. 295–314.

[HARR68] M. HARRISON AND O. IBARRA, *Multitape and multihead pushdown automata*, Inform. Control, 13 (1968), pp. 433–470.

[HARR78] M. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[HENN61] F. HENNIE, *Iterative Arrays of Logical Circuits*, MIT Press, Cambridge, MA, 1961.

[HOPC79] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[IBAR74] O. IBARRA, *A note on semilinear sets and bounded-reversal multihead pushdown automata*, Inform. Process. Lett., 3 (1974), pp. 25–28.

[IBAR85a] O. IBARRA AND M. PALIS, *Two-dimensional systolic arrays: characterizations and applications*, Tech. Report 85-1, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN, Jan. 1985; Theoret. Comput. Sci., to appear.

[IBAR85b] O. IBARRA, M. PALIS AND S. KIM, *Some results concerning linear iterative (systolic) arrays*, J. Parallel Distributed Computing, 2 (1985), pp. 182–218.

[IBAR86] O. IBARRA, S. KIM AND M. PALIS, *Designing systolic algorithms using sequential machines*, IEEE Trans. Comput., C35-6 (June 1986), pp. 31–42; extended abstract, Proc. 25th IEEE Symp. on Foundations of Computer Science, 1984, pp. 46–55.

[SEID79] S. SEIDEL, *Language recognition and the synchronization of cellular automata*, Tech. Report 79-02, Dept. of Computer Science, Univ. of Iowa, Iowa City, IA, 1979.

[SMIT70] A. SMITH III, *Cellular automata and formal languages*, Proc. 11th IEEE Annual Symp. on Switching and Automata Theory, 1970, pp. 216–224.

[SMIT71] A. SMITH III, *Cellular automata complexity trade-offs*, Inform. Control, 18 (1971), pp. 466–482.

[UMEO82] H. UMEO, K. MORITA AND K. SUGATA, *Deterministic one-way simulation of two-way real-time cellular automata and its related problems*, Inform. Process. Lett., 14 (1982), pp. 159–161.

SIAM J. COMPUT.
Vol. 16, No. 6, December 1987
© 1987 Society for Industrial and Applied Mathematics
012

A QUEUEING ANALYSIS OF HASHING WITH LAZY DELETION*

JOHN A. MORRISON†, LARRY A. SHEPP† AND CHRISTOPHER J. VAN WYK†

Abstract. Hashing with lazy deletion is a simple method for maintaining a dynamic dictionary: items are inserted and sought as usual in a separate-chaining hash table; however, items that no longer need to be in the data structure remain until a later insertion operation stumbles on them and removes them from the table. Because hashing with lazy deletion does not delete items as soon as possible, it keeps more items in the dictionary than methods that use more careful deletion strategies. On the other hand, its space overhead is much smaller than those more careful methods, so if the number of extra items is not too large, hashing with lazy deletion can be a practical algorithm when space is scarce. In this paper, we analyze the expected amount of excess space used by hashing with lazy deletion.

Key words. dynamic dictionary, first passage times, storage

AMS(MOS) subject classifications. 60K30, 68A50

1. Introduction.

The strategy of hashing with lazy deletion was proposed by Van Wyk and Vitter [1] for use in the following situation:

> A sequence of items is given; each item includes a search key, a starting time, and an expiration time. The items arrive in the order of their starting times, and each item must be kept in a dynamic dictionary (available for searching) until the arrival of an item whose starting time is later than the item's expiration time.

Algorithms that delete items as soon as possible have unacceptably high overhead in space [1], even though they require less storage space for the items themselves. Hashing with lazy deletion means keeping the items hashed by search key in a table of linked lists (separate chains); each time an item is added to a list, any items on that list that the new item shows to be expired are deleted from that list. This deletion procedure is "lazy" because there is no separate operation associated with clearing expired items out of the table: expired items are only deleted when they are encountered during an insertion operation.

Hashing with lazy deletion was used to solve a geometric adjacency problem that arose in the analysis of integrated circuits [2]. The items were line segments in the plane: starting time corresponded to minimum x-coordinate, expiration time to maximum x-coordinate, and search key to y-intercept. At any abscissa x all segments that cross the vertical line at x must be in the dictionary. The artwork for a chip of modest size (around 100,000 transistors) can involve three to ten million line segments. But it is reasonable to expect that at most 50,000 segments cross any vertical line, so the dictionary only needs to store about 50,000 segments. Hashing with lazy deletion uses 24 bytes per segment, while hashing with careful deletion requires 36 bytes per segment. Thus, using careful deletion, the dictionary can fill around 1.8 megabytes; even if hashing with lazy deletion keeps 20% more segments in the dictionary than are necessary, and thus requires 1.44 megabytes plus the size of the hash table, it still uses less space than the careful deletion strategy.

Given a sequence of items, let $N(t)$ be the number of items that start at or before time t and expire at or after time t. The statement of the problem requires that these items be in the dictionary at time t, so any algorithm used to solve this problem must

* Received by the editors February 3, 1986; accepted for publication (in revised form) March 24, 1987.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

use space proportional to $N(t)$ at time t; in this sense, an algorithm whose space complexity is $N(t)$ for all t has *optimum* space complexity. Given the same sequence of items and the number H of buckets in a hash table being managed with lazy deletion, let $U_H(t)$ be the number of items present in the table at time t; $U_H(t)$ is the space complexity of hashing with lazy deletion. Note that for all H and all t, $U_H(t) \geqq N(t)$.

Under the assumptions that the starting times of items form a Poisson process, that item life-times are independent and exponentially distributed, and that the hash function is uniform on the search keys, Van Wyk and Vitter showed that the expected value of $U_H(t)$ is H more than the expected value of $N(t)$. This suggests that it is possible to choose H so that the expected search time is constant while space complexity is little more than optimum.

In this paper we obtain more information about the space complexity of hashing with lazy deletion by studying the values of $\max_t N(t)$ and $\max_t U_H(t)$, where $0 \leqq t \leqq T$. The quantity $\max_t N(t)$ is a lower bound on the space required by any algorithm that solves the problem, while $\max_t U_H(t)$ is the space required by hashing with lazy deletion using H buckets. Suppose the amount of space available to solve the problem is S. If $\max_t U_H(t) > S \geqq \max_t N(t)$, then hashing with lazy deletion cannot solve the problem even though an algorithm with a more careful deletion strategy could solve the problem in space S. Hence we wish to study the amount by which $\max_t U_H(t)$ exceeds $\max_t N(t)$; note that the expressions $N(t)$ and $U_H(t)$ may attain their maxima at different values of t.

In § 2 we derive an expression for $\max_t N(t)$ using techniques from queueing theory. In § 3 we derive an expression for $\max_t U_1(t)$ by studying the following queueing problem:

> The door of a bank is locked from the inside, so customers can enter at any time, but customers who have completed their transactions must wait in the lobby for an arriving customer before they can leave; assuming that customers are served immediately, what is the total number of customers—in service and waiting to leave—in the bank?

This queueing problem directly models the behavior of hashing with lazy deletion using a single bucket ($H = 1$). Section 4 shows how the results in §§ 2 and 3 provide bounds on the space complexity of hashing with lazy deletion using H buckets. Section 5 presents some numerical results.

2. Analysis of $N(t)$. We first consider the case of an M/M/∞ queue, with Poisson arrival rate λ, and mean service time $1/\mu$. The equilibrium probability p_j that there are j customers in the system is [3]

$$p_j = \frac{a^j}{j!} e^{-a}, \qquad a = \frac{\lambda}{\mu}. \tag{2.1}$$

Let $N(t)$ denote the number of customers in the system at time t. We are interested in the distribution of $\max_{0 \leqq t \leqq T} N(t)$, given that the system is in equilibrium at $t = 0$. Let $s_{j,k}(t)$ denote the density of the passage time from j to k. Now, $\max_{0 \leqq t \leqq T} N(t)$ will be less than k if, and only if, the system is initially in some state $j < k$, and the passage time from j to k is greater than T. Hence,

$$\begin{aligned} \Pi_k(T) &\equiv \Pr\left\{ \max_{0 \leqq t \leqq T} N(t) < k \,\middle|\, \text{equilibrium at } t = 0 \right\} \\ &= \sum_{j=0}^{k-1} p_j \int_T^\infty s_{j,k}(t)\, dt. \end{aligned} \tag{2.2}$$

We now turn our attention to the calculation of $s_{j,k}(t)$. We use the method of Siegert (cf. [4] and [5]), which is possible because $N(t)$ increases in unit steps. Let

$$\sigma_m(\xi) = \int_0^\infty e^{-\xi t} s_{m,m+1}(t)\, dt \tag{2.3}$$

be the Laplace transform of the upward passage time density at m. Then [6],

$$\int_0^\infty e^{-\xi t} s_{j,k}(t)\, dt = \prod_{m=j}^{k-1} \sigma_m(\xi), \qquad k > j. \tag{2.4}$$

Also [6], since the departure rate from state m is $m\mu$,

$$(\xi + \lambda + m\mu)\sigma_m = \lambda + m\mu\sigma_{m-1}\sigma_m, \qquad m = 0, 1, \cdots. \tag{2.5}$$

We define

$$\omega_{m+1} = \left(\prod_{l=0}^{m} \sigma_l \right)^{-1}, \qquad m = 0, 1, \cdots, \quad \omega_0 = 1, \tag{2.6}$$

and let

$$\eta = \frac{\xi}{\mu}, \qquad a = \frac{\lambda}{\mu}. \tag{2.7}$$

Then, from (2.5), we obtain

$$(\eta + a + m)\omega_m = a\omega_{m+1} + m\omega_{m-1}, \qquad m = 0, 1, \cdots. \tag{2.8}$$

Next, we let

$$\theta_m = a^m \omega_m / m!. \tag{2.9}$$

Hence, from (2.6) and (2.8), we find that

$$(\eta + a + m)\theta_m = (m+1)\theta_{m+1} + a(1 - \delta_{m0})\theta_{m-1}, \qquad \theta_0 = 1, \tag{2.10}$$

where δ denotes the Kronecker symbol. It follows from (2.10) that θ_m is a polynomial of degree m in both η and a, and it can be related to a Poisson-Charlier polynomial. In the notation of [7],

$$\theta_m = s_m(\eta, a). \tag{2.11}$$

From (2.6), (2.7), (2.9) and (2.11), we obtain

$$\prod_{m=j}^{k-1} \sigma_m(\xi) = \frac{j!\, a^k s_j(\xi/\mu, a)}{k!\, a^j s_k(\xi/\mu, a)}, \qquad 0 \leqq j < k. \tag{2.12}$$

The zeros of $s_k(\eta, a)$, regarded as a function of η, are negative and distinct [7], and we denote them by $\eta_l = -\kappa_l(a)$, $l = 1, \cdots, k$. Then

$$\lim_{\xi \to -\mu\kappa_l} (\xi + \mu\kappa_l) \frac{s_j(\xi/\mu, a)}{s_k(\xi/\mu, a)} = \frac{\mu s_j(-\kappa_l, a)}{s_k'(-\kappa_l, a)}, \tag{2.13}$$

where the prime denotes derivative with respect to the first argument. It follows, from (2.4), (2.12) and (2.13), that

$$s_{j,k}(t) = \mu \frac{j!\, a^k}{k!\, a^j} \sum_{l=1}^{k} \frac{s_j(-\kappa_l, a)}{s_k'(-\kappa_l, a)} e^{-\mu\kappa_l t}, \qquad 0 \leqq j < k. \tag{2.14}$$

But [7], for $|u| < 1$,

$$\sum_{k=0}^{\infty} s_k(x, a) u^k = e^{au}(1-u)^{-x} = e^{au} e^{-x \log(1-u)}. \tag{2.15}$$

Hence,

$$\sum_{k=0}^{\infty} s'_k(x, a)u^k = -\log(1-u)\, e^{au}\, e^{-x\log(1-u)} = \sum_{r=1}^{\infty} \frac{u^r}{r} \sum_{m=0}^{\infty} s_m(x, a)u^m. \tag{2.16}$$

This implies that

$$s'_k(x, a) = \sum_{m=0}^{k-1} \frac{s_m(x, a)}{(k-m)}. \tag{2.17}$$

We may now obtain an expression for $\Pi_k(T)$. From (2.1), (2.2) and (2.14), it is found that

$$\Pi_k(T) = \frac{a^k}{k!} e^{-a} \sum_{j=0}^{k-1} \sum_{l=1}^{k} \frac{s_j(-\kappa_l, a)}{\kappa_l s'_k(-\kappa_l, a)} e^{-\mu\kappa_l T}. \tag{2.18}$$

But [7],

$$\sum_{j=0}^{n} s_j(x, a) = s_n(1+x, a). \tag{2.19}$$

Hence

$$\Pi_k(T) = \frac{a^k}{k!} e^{-a} \sum_{l=1}^{k} \frac{s_{k-1}(1-\kappa_l, a)}{\kappa_l s'_k(-\kappa_l, a)} e^{-\mu\kappa_l T}. \tag{2.20}$$

Besides $\Pi_k(T)$ we are also interested in

$$\Lambda(T) \equiv E\left\{ \max_{0\leqq t\leqq T} N(t) \,\middle|\, \text{equilibrium at } t=0 \right\} = \sum_{k=1}^{\infty} k[\Pi_{k+1}(T) - \Pi_k(T)] = \sum_{k=1}^{\infty} [1-\Pi_k(T)], \tag{2.21}$$

where we have used (2.2) and the fact that $\lim_{k\to\infty} \Pi_k(T) = 1$.

3. Analysis of $U_1(t)$. We now investigate the single-bucket model of a dynamic dictionary considered by Van Wyk and Vitter [1]. This differs from the M/M/∞ queue in that customers whose service has been completed cannot leave until the next customer arrives. We let $N(t)$ and $W(t)$ denote the number of customers at time t which are in service, and waiting to exit (having been served), respectively. It is evident that $N(t)$ is the number of customers in the corresponding M/M/∞ queue. The equilibrium probability $p_{m,n}$ that $N(t) = m$ and $W(t) = n$ satisfies the recurrence relation [1]

$$(a+m)p_{m,n} = (m+1)p_{m+1,n-1} + a\delta_{n0} \sum_{r=0}^{\infty} p_{m-1,r}, \tag{3.1}$$

where $p_{m,n} = 0$ if $\min(m, n) < 0$. We have set $\beta = 1/\mu$ in [1], and $\lambda = a\mu$, as before.

If we set $n = 0$ in (3.1), we obtain

$$(a+m)p_{m,0} = a \sum_{r=0}^{\infty} p_{m-1,r} = ap_{m-1}. \tag{3.2}$$

Hence $p_{0,0} = 0$ and, from (2.1),

$$p_{m,0} = \frac{a^m e^{-a}}{(m-1)!(a+m)}, \qquad m = 1, 2, \cdots. \tag{3.3}$$

For $n \geqq 1$, (3.1) implies that

$$(a+m)p_{m,n} = (m+1)p_{m+1,n-1}. \tag{3.4}$$

It follows by induction on n, from (3.3) and (3.4), that

$$p_{m,n} = \frac{(m+n)a^{m+n}\Gamma(a+m)}{m!\Gamma(a+m+n+1)}\, e^{-a}. \tag{3.5}$$

Note that

$$\begin{aligned} \sum_{n=0}^{\infty} p_{m,n} &= \frac{a^m}{m!}\, e^{-a}\Gamma(a+m) \sum_{n=0}^{\infty} \left[\frac{a^n}{\Gamma(a+m+n)} - \frac{a^{n+1}}{\Gamma(a+m+n+1)}\right] \\ &= \frac{a^m}{m!}\, e^{-a} = p_m, \end{aligned} \tag{3.6}$$

as it should.

We are interested in the distribution of $\max_{0 \leqq t \leqq T} [N(t)+W(t)]$, given that the system is in equilibrium at $t=0$, namely

$$\begin{aligned} P_k(T) &\equiv \Pr\left\{ \max_{0 \leqq t \leqq T} [N(t)+W(t)] < k \,\middle|\, \text{equilibrium at } t=0 \right\} \\ &= \sum_{m+n<k} p_{m,n} \Pr\left\{ \max_{0 \leqq t \leqq T} [N(t)+W(t)] < k \,\middle|\, N(0)=m,\ W(0)=n \right\}. \end{aligned} \tag{3.7}$$

But $N(t)+W(t)$ remains constant between arrivals, and $W(t)=0$ immediately after an arrival. Hence, $N(t)+W(t)$ increases only when $N(t)$ does, and

$$\begin{aligned} P_k(T) &= \sum_{m+n<k} p_{m,n} \Pr\left\{ \max_{0 \leqq t \leqq T} N(t) < k \,\middle|\, N(0)=m \right\} \\ &= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1-m} p_{m,n} \int_T^{\infty} s_{m,k}(t)\, dt. \end{aligned} \tag{3.8}$$

Now, from (3.5),

$$\begin{aligned} \sum_{n=0}^{k-1-m} p_{m,n} &= \frac{a^m}{m!}\, e^{-a}\Gamma(a+m) \sum_{n=0}^{k-1-m} \left[\frac{a^n}{\Gamma(a+m+n)} - \frac{a^{n+1}}{\Gamma(a+m+n+1)}\right] \\ &= \frac{e^{-a}}{m!}\left[a^m - a^k \frac{\Gamma(a+m)}{\Gamma(a+k)}\right]. \end{aligned} \tag{3.9}$$

Hence, from (2.14), (3.8) and (3.9), we obtain

$$P_k(T) = \frac{a^k}{k!}\, e^{-a} \sum_{m=0}^{k-1} \left[1 - a^{k-m}\frac{\Gamma(a+m)}{\Gamma(a+k)}\right] \sum_{l=1}^{k} \frac{s_m(-\kappa_l, a)}{\kappa_l s_k'(-\kappa_l, a)}\, e^{-\mu\kappa_l T}. \tag{3.10}$$

Besides $P_k(T)$ we are also interested in

$$\begin{aligned} \Omega(T) &\equiv E\left\{ \max_{0 \leqq t \leqq T} [N(t)+W(t)] \,\middle|\, \text{equilibrium at } t=0 \right\} \\ &= \sum_{k=1}^{\infty} k[P_{k+1}(T) - P_k(T)] = \sum_{k=1}^{\infty} [1 - P_k(T)], \end{aligned} \tag{3.11}$$

where we have used (3.7) and the fact that $\lim_{k \to \infty} P_k(T) = 1$.

The expression in (3.10) should be compared with that in (2.18), which corresponds to the M/M/∞ queue. We note that $P_1(T)=0$. Also, from (3.7), we have

$$P_k(0)=\sum_{m+n<k} p_{m,n}=\sum_{j=0}^{k-1}\sum_{m+n=j} p_{m,n}. \tag{3.12}$$

Now $p_{0,0}=0$ and for $j\geqq 1$, from (3.5), we obtain

$$\begin{aligned}\sum_{m+n=j} p_{m,n} &= \frac{ja^{j-1}e^{-a}}{\Gamma(a+j+1)}\sum_{m=0}^{j}\left[\frac{\Gamma(a+m+1)}{m!}-\frac{m\Gamma(a+m)}{m!}\right] \\ &= \frac{a^{j-1}}{(j-1)!}e^{-a}=p_{j-1},\end{aligned} \tag{3.13}$$

from (2.1). Hence, from (2.2) and (3.12), for $k\geqq 2$,

$$P_k(0)=\sum_{j=1}^{k-1} p_{j-1}=\sum_{j=0}^{k-2} p_j=\Pi_{k-1}(0). \tag{3.14}$$

4. Application. In this section we show explicitly the queue parameters that were implicit in the last two sections. That is, we write $N(t, \lambda, \mu)$ to denote $N(t)$ of § 2 and $W(t, \lambda, \mu)$ to denote $W(t)$ of § 3. We also define the following:

$$\Pi_k(T, \lambda, \mu)\equiv \Pr\left\{\max_{0\leqq t\leqq T} N(t, \lambda, \mu)<k \,\middle|\, \text{equilibrium at } t=0\right\}, \tag{4.1}$$

$$\Lambda(T, \lambda, \mu)=E\left\{\max_{0\leqq t\leqq T} N(t, \lambda, \mu) \,\middle|\, \text{equilibrium at } t=0\right\}, \tag{4.2}$$

$$P_k(T, \lambda, \mu)\equiv \Pr\left\{\max_{0\leqq t\leqq T}[N(t, \lambda, \mu)+W(t, \lambda, \mu)]<k \,\middle|\, \text{equilibrium at } t=0\right\} \tag{4.3}$$

and

$$\Omega(T, \lambda, \mu)=E\left\{\max_{0\leqq t\leqq T}[N(t, \lambda, \mu)+W(t, \lambda, \mu)] \,\middle|\, \text{equilibrium at } t=0\right\}. \tag{4.4}$$

To apply the results of §§ 2 and 3 to the case $H>1$, we treat the processing in each bucket i as an independent M/M/∞ queue with Poisson arrival rate λ/H and mean service time $1/\mu$; for bucket i, we denote the number of items in service at time t by $N_i(t, \lambda/H, \mu)$ and the number of items waiting to be deleted at time t by $W_i(t, \lambda/H, \mu)$.

Under these assumptions, the space complexity of hashing with lazy deletion using H buckets is

$$U_H(t, \lambda, \mu)\equiv\sum_{i=1}^{H}[N_i(t, \lambda/H, \mu)+W_i(t, \lambda/H, \mu)]. \tag{4.5}$$

Let

$$\Phi_{H,k}(T, \lambda, \mu)\equiv \Pr\left\{\max_{0\leqq t\leqq T} U_H(t, \lambda, \mu)<k \,\middle|\, \text{equilibrium at } t=0\right\}, \tag{4.6}$$

and

$$\Psi_H(T, \lambda, \mu)\equiv E\left\{\max_{0\leqq t\leqq T} U_H(t, \lambda, \mu) \,\middle|\, \text{equilibrium at } t=0\right\}. \tag{4.7}$$

The value of $\Psi_H(T, \lambda, \mu)$ can be bounded from below and from above [1].

Since the operations in each hash bucket are independent, the input to the whole algorithm is the same as that of an M/M/∞ queue with Poisson arrival rate λ and mean service time $1/\mu$. Hashing with lazy deletion using H buckets inserts each item at the same time as hashing with lazy deletion using only one bucket, but the H-bucket algorithm cannot delete any item sooner than the single-bucket algorithm. Therefore U_1 is a lower bound on the space complexity of hashing with lazy deletion:

$$U_1(t, \lambda, \mu) \leqq U_H(t, \lambda, \mu). \tag{4.8}$$

It follows from (3.7), (4.6) and (4.8) that

$$P_k(T, \lambda, \mu) \geqq \Phi_{H,k}(T, \lambda, \mu); \tag{4.9}$$

and (3.11), (4.7) and (4.9) imply that

$$\Omega(T, \lambda, \mu) \leqq \Psi_H(T, \lambda, \mu). \tag{4.10}$$

The maximum space used by the algorithm is no larger than the sum over the buckets of the maximum space used in each:

$$\max_{0 \leqq t \leqq T} U_H(t, \lambda, \mu) \leqq \sum_{i=1}^{H} \max_{0 \leqq t \leqq T} [N_i(t, \lambda/H, \mu) + W_i(t, \lambda/H, \mu)]. \tag{4.11}$$

It follows from (4.4), (4.7) and (4.11) that

$$\Psi_H(T, \lambda, \mu) \leqq H\Omega(T, \lambda/H, \mu). \tag{4.12}$$

To summarize, (4.10) and (4.12) give the following upper and lower bounds on the space complexity of hashing with lazy deletion using H buckets:

$$\Omega(T, \lambda, \mu) \leqq \Psi_H(T, \lambda, \mu) \leqq H\Omega(T, \lambda/H, \mu). \tag{4.13}$$

5. Numerical results. We evaluated numerically formulas (2.18), (2.21), (3.10) and (3.11) for several choices of λ and μ. This required computing $\Pi_k(T)$ and $P_k(T)$ for $k = 0, 1, \cdots$ until they were close enough to one that they made no discernible contribution to $\Lambda(T)$ or $\Omega(T)$. Since the evaluation of $\Pi_k(T)$ or $P_k(T)$ requires finding the zeros of the kth degree polynomial (2.11), we chose λ and μ so that both $1 - \Pi_{20}(T) \ll 1$ and $1 - P_{20}(T) \ll 1$ for all relevant values of T. We used subroutine IMTQL2 [8] to determine the zeros of (2.11) by finding the eigenvalues of the appropriate symmetric tridiagonal matrix.

In Figs. 1 and 2 we have taken $\lambda = 10$, $\mu = 2$. Figure 1 shows $\Pi_k(T, 10, 2)$ and $P_k(T, 10, 2)$ for selected values of T. The curve for $T = 0$ shows that $P_k(0, 10, 2) = \Pi_{k-1}(0, 10, 2)$, illustrating (3.14). The graph shows that the difference between $P_k(T, 10, 2)$ and $\Pi_k(T, 10, 2)$ is already quite small at $T = 1$, and has almost vanished by $T = 10$.

Figure 2 shows $\Lambda(T, 10, 2)$ and $\Omega(T, 10, 2)$. Since the Markov chains studied in §§ 2 and 3 are irreducible, we know that both $\Lambda(T, \lambda, \mu)$ and $\Omega(T, \lambda, \mu)$ grow without bound as $T \to \infty$, for any choice of λ and μ; this figure shows that the growth is slow. As one would expect from Fig. 1, $\Lambda(T, 10, 2)$ and $\Omega(T, 10, 2)$ start out differing by one, but quickly become much closer. The data structure interpretation of this is that hashing with lazy deletion using one bucket has almost optimum space complexity.

Figure 3 shows $H\Omega(T, 10/H, 2)$ for several values of H. The curve for $H = 1$, $\Omega(T, \lambda, \mu)$, is a lower bound on $\Psi_H(T, \lambda, \mu)$ by (4.10). The upper bound given by

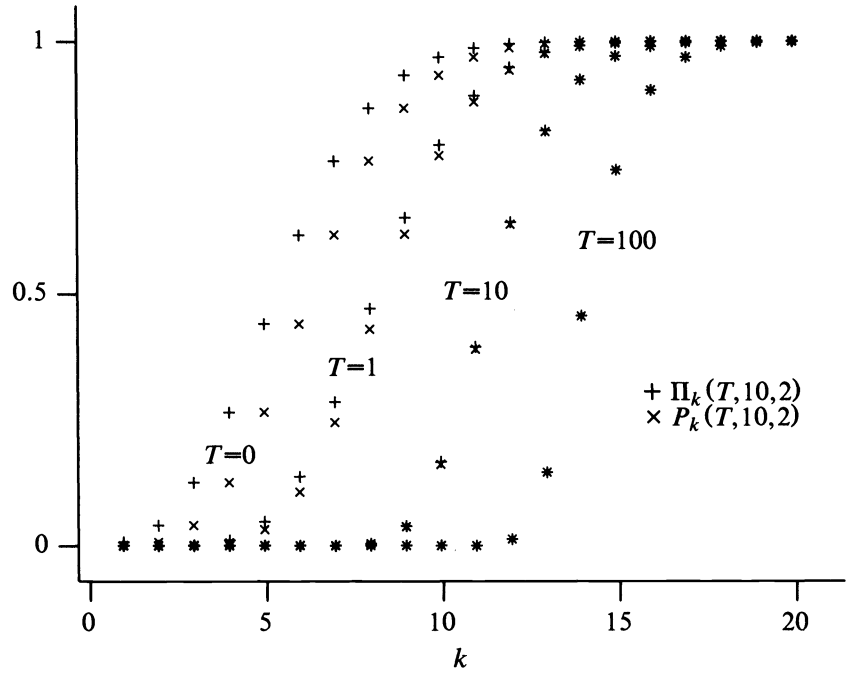


FIG. 1

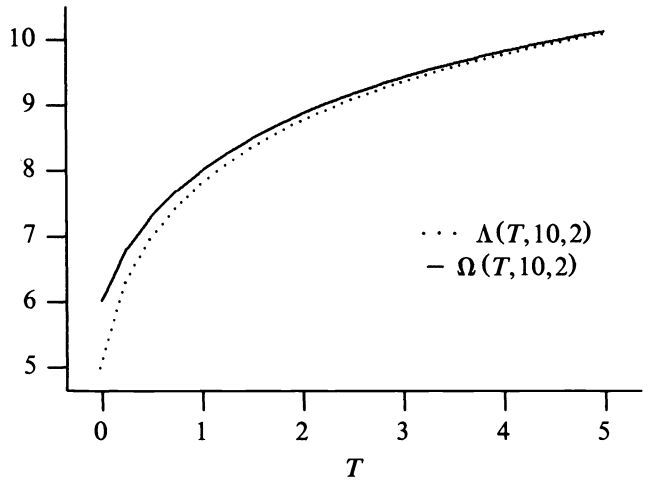


FIG. 2

(4.12) becomes progressively larger as H increases. It appears that $\Psi_H(T, \lambda, \mu)$ exceeds $\Omega(T, \lambda, \mu)$ by an amount that is at most $O(H)$. Since $\Omega(T, \lambda, \mu)$ is only a little larger than $\Lambda(T, \lambda, \mu)$, this suggests that $\Psi_H(T, \lambda, \mu) - \Lambda(T, \lambda, \mu) = O(H)$, which would imply that H can be chosen to be $O(\lambda/\mu)$ in order to give constant expected search time and almost optimum space complexity.

Observations about $\Lambda(T)$, $\Omega(T)$ and $\Psi_H(T)$ based on Figs. 1-3 can be applied to the case of constant λ/μ by an appropriate choice of scaling factors. Other models are appropriate for some applications. For example, to model the analysis of integrated

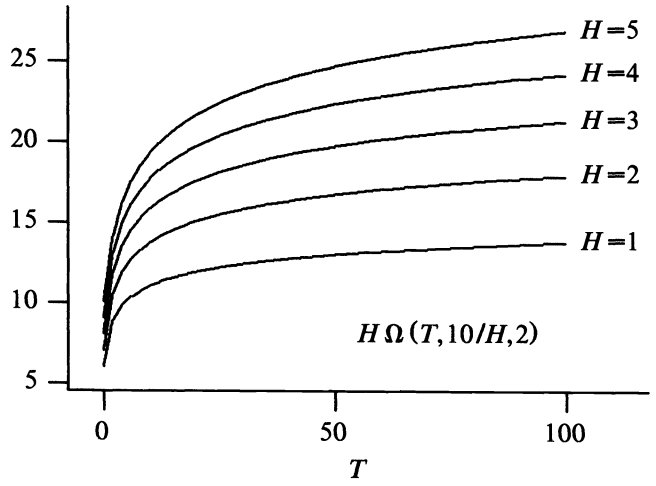


FIG. 3

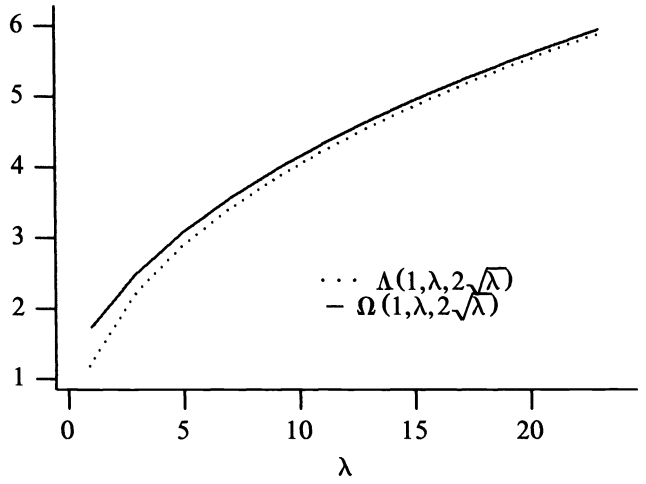


FIG. 4

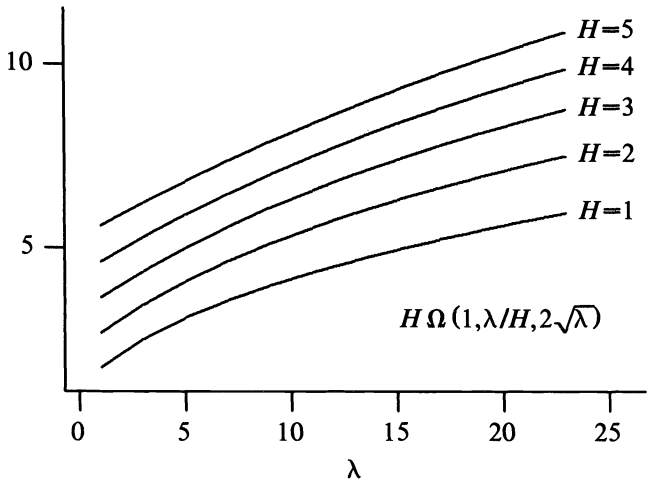


FIG. 5

circuits discussed in § 1, we take λ to be the chip size, $\mu = c\sqrt{\lambda}$, and study $\Lambda(1)$ and $\Omega(1)$ as λ increases [1]. Figures 4 and 5 show results for $c = 2$.

Acknowledgment. Thanks to Jeff Vitter for several thoughtful comments on the manuscript.

REFERENCES

[1] C. J. VAN WYK AND J. S. VITTER, *The complexity of hashing with lazy deletion*, Algorithmica, 1 (1986), pp. 17-29.

[2] T. G. SZYMANSKI AND C. J. VAN WYK, *Space efficient algorithms for VLSI artwork analysis*, Proc. 20th Design Automation Conference, 1983, pp. 734-739.

[3] L. KLEINROCK, *Queueing Systems, Vol.* I: *Theory*, John Wiley, New York, 1975.

[4] D. A. DARLING AND A. J. F. SIEGERT, *The first passage problem for a continuous Markov process*, Ann. Math. Statist., 24 (1953), pp. 624-639.

[5] A. J. F. SIEGERT, *On the first passage time probability problem*, Phys. Rev., 81 (1951), pp. 617-623.

[6] J. KEILSON, *Markov Chain Models—Rarity and Exponentiality*, Applied Mathematical Sciences, Vol. 28, Springer-Verlag, New York, 1979.

[7] J. A. MORRISON, *Analysis of some overflow problems with queueing*, Bell System Tech. J., 59 (1980), pp. 1427-1462.

[8] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gaussian quadrature rules*, Math. Comp., 23 (1969), pp. 221-230.